

Claude Code

Documentation Complète

By Anthropic

Date de production : 2026-03-17

Un outil de codage agentique dans votre terminal,
qui comprend votre code et vous aide à programmer plus vite.

Compilé depuis code.claude.com/docs/fr
Pour la dernière version, visitez le site web.

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

Table of Contents

Part 1: Getting Started	5
Aperçu de Claude Code	5
Démarrage rapide	
Démarrer avec l'application de bureau	
Configurer Claude Code	25
Comment fonctionne Claude Code	
Part 2: Core Usage	46
Étendre Claude Code	46
Mode interactif	59
Meilleures pratiques pour Claude Code	
Flux de travail courants	96
Part 3: Commands & Reference	
Référence CLI	121
Personnaliser les raccourcis clavier	
Part 4: Configuration	144
Configurer les autorisations	144
Comment Claude se souvient de votre projet	
Paramètres Claude Code	168
Configuration du modèle	219
Styles de sortie	228
Accélérez les réponses avec le mode rapide	
Gérer les coûts efficacement	236
Part 5: Extensibility	244
Référence des hooks	244
Automatiser les workflows avec les hooks	
Connecter Claude Code aux outils via MCP	
Étendre Claude avec des skills	
Créer des plugins	401
Référence des plugins	414
Créer et distribuer une place de marché de plugins	
Découvrir et installer des plugins prédéfinis via les marketplaces	
Part 6: Advanced & Automation	480
Créer des sous-agents personnalisés	

Orchestrer des équipes de sessions Claude Code	
Exécuter Claude Code par programmation	
Continuer les sessions locales depuis n'importe quel appareil avec Remote Control	
Exécuter des prompts selon un calendrier	
Révision de code	539

Part 7: CI/CD & Integrations

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

Part 8: IDE & Platform Integration 581

Utiliser Claude Code dans VS Code	
JetBrains IDEs	599
Utiliser Claude Code Desktop	604
Utiliser Claude Code avec Chrome (bêta)	
Claude Code dans Slack	638
Claude Code sur le web	645

Part 9: Security & Privacy

Sécurité	666
Sandboxing	672
Checkpointing	683
Utilisation des données	686
Légal et conformité	692
Zéro conservation des données	

Part 10: Enterprise & Monitoring 697

Suivre l'utilisation de l'équipe avec l'analytique	
Surveillance	705
Aperçu du déploiement en entreprise	
Configurer les paramètres gérés par le serveur (bêta publique)	

Part 11: Cloud Providers 735

Claude Code sur Amazon Bedrock	
Claude Code sur Google Vertex AI	
Claude Code sur Microsoft Foundry	
Configuration de la passerelle LLM	
Configuration réseau d'entreprise	

Part 12: Environment Setup 760

Conteneurs de développement	760
Optimisez votre configuration de terminal	

Personnalisez votre barre de statut

Part 13: Troubleshooting & Changelog

798

Dépannage

798

Part 1: Getting Started

Aperçu de Claude Code

Claude Code est un outil de codage agentique qui lit votre base de code, modifie les fichiers, exécute des commandes et s'intègre à vos outils de développement. Disponible dans votre terminal, IDE, application de bureau et navigateur.

Claude Code est un assistant de codage alimenté par l'IA qui vous aide à créer des fonctionnalités, corriger des bogues et automatiser les tâches de développement. Il comprend l'ensemble de votre base de code et peut travailler sur plusieurs fichiers et outils pour accomplir les tâches.

Commencer

Choisissez votre environnement pour commencer. La plupart des surfaces nécessitent un [abonnement Claude](#) ou un compte [Anthropic Console](#). Le CLI Terminal et VS Code prennent également en charge les [fournisseurs tiers](#).

Terminal

Le CLI complet pour travailler avec Claude Code directement dans votre terminal. Modifiez les fichiers, exécutez des commandes et gérez l'ensemble de votre projet à partir de la ligne de commande.

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.

Ensuite, démarrez Claude Code dans n'importe quel projet :

```
cd your-project  
claude
```

Vous serez invité à vous connecter lors de la première utilisation. C'est tout ! [Continuez avec le Démarrage rapide →](#)

Tip:

Consultez la [configuration avancée](#) pour les options d'installation, les mises à jour manuelles ou les instructions de désinstallation. Visitez la [résolution des problèmes](#) si vous rencontrez des problèmes.

VS Code

L'extension VS Code fournit des diffs en ligne, des mentions @, un examen du plan et l'historique des conversations directement dans votre éditeur.

- [Installer pour VS Code](#)
- [Installer pour Cursor](#)

Ou recherchez « Claude Code » dans la vue Extensions (`Cmd+Shift+X` sur Mac, `Ctrl+Shift+X` sur Windows/Linux). Après l'installation, ouvrez la Palette de commandes (`Cmd+Shift+P` / `Ctrl+Shift+P`), tapez « Claude Code » et sélectionnez **Ouvrir dans un nouvel onglet**.

[Commencer avec VS Code →](#)

Application de bureau

Une application autonome pour exécuter Claude Code en dehors de votre IDE ou terminal. Examinez les diffs visuellement, exécutez plusieurs sessions côte à côte, planifiez des tâches récurrentes et lancez des sessions cloud.

Téléchargez et installez :

- [macOS](#) (Intel et Apple Silicon)
- [Windows](#) (x64)
- [Windows ARM64](#) (sessions distantes uniquement)

Après l'installation, lancez Claude, connectez-vous et cliquez sur l'onglet **Code** pour commencer à coder. Un [abonnement payant](#) est requis.

[En savoir plus sur l'application de bureau →](#)

Web

Exécutez Claude Code dans votre navigateur sans configuration locale. Lancez des tâches longues et revenez quand elles sont terminées, travaillez sur des dépôts que vous n'avez pas localement ou exécutez plusieurs tâches en parallèle. Disponible sur les navigateurs de bureau et l'application Claude iOS.

Commencez à coder sur claude.ai/code.

[Commencer sur le web →](#)

JetBrains

Un plugin pour IntelliJ IDEA, PyCharm, WebStorm et autres IDE JetBrains avec visualisation interactive des diffs et partage du contexte de sélection.

Installez le [plugin Claude Code](#) depuis la Marketplace JetBrains et redémarrez votre IDE.

[Commencer avec JetBrains →](#)

Ce que vous pouvez faire

Voici quelques-unes des façons dont vous pouvez utiliser Claude Code :

Claude Code gère les tâches fastidieuses qui vous prennent du temps : écrire des tests pour le code non testé, corriger les erreurs de lint dans un projet, résoudre les conflits de fusion, mettre à jour les dépendances et écrire les notes de version.

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

Décrivez ce que vous voulez en langage naturel. Claude Code planifie l'approche, écrit le code sur plusieurs fichiers et vérifie qu'il fonctionne.

Pour les bogues, collez un message d'erreur ou décrivez le symptôme. Claude Code trace le problème dans votre base de code, identifie la cause racine et implémente une correction. Consultez les [flux de travail courants](#) pour plus d'exemples.

Claude Code fonctionne directement avec git. Il prépare les modifications, écrit les messages de commit, crée des branches et ouvre des demandes de tirage.

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

En CI, vous pouvez automatiser l'examen du code et le triage des problèmes avec [GitHub Actions](#) ou [GitLab CI/CD](#).

Le [Model Context Protocol \(MCP\)](#) est une norme ouverte pour connecter les outils d'IA aux sources de données externes. Avec MCP, Claude Code peut lire vos documents de conception dans Google Drive, mettre à jour les tickets dans Jira, extraire les données de Slack ou utiliser vos outils personnalisés.

`CLAUDE.md` est un fichier markdown que vous ajoutez à la racine de votre projet que Claude Code lit au début de chaque session. Utilisez-le pour définir les normes de codage, les décisions architecturales, les bibliothèques préférées et les listes de contrôle d'examen.

Claude construit également une [mémoire automatique](#) au fur et à mesure qu'il travaille, en sauvegardant les apprentissages comme les commandes de construction et les informations de débogage entre les sessions sans que vous ayez à écrire quoi que ce soit.

Créez des [commandes personnalisées](#) pour emballer les flux de travail répétables que votre équipe peut partager, comme `/review-pr` ou `/deploy-staging`.

Les [hooks](#) vous permettent d'exécuter des commandes shell avant ou après les actions de Claude Code, comme le formatage automatique après chaque modification de fichier ou l'exécution de lint avant un commit.

Lancez [plusieurs agents Claude Code](#) qui travaillent sur différentes parties d'une tâche simultanément. Un agent principal coordonne le travail, assigne les sous-tâches et fusionne les résultats.

Pour les flux de travail entièrement personnalisés, le [Agent SDK](#) vous permet de créer vos propres agents alimentés par les outils et les capacités de Claude Code, avec un contrôle total sur l'orchestration, l'accès aux outils et les permissions.

Claude Code est composable et suit la philosophie Unix. Canalez les journaux dedans, exécutez-le en CI ou enchaînez-le avec d'autres outils :

```
## 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"
```

Consultez la [référence CLI](#) pour l'ensemble complet des commandes et des drapeaux.

Les sessions ne sont pas liées à une seule surface. Déplacez le travail entre les environnements à mesure que votre contexte change :

- Éloignez-vous de votre bureau et continuez à travailler depuis votre téléphone ou n'importe quel navigateur avec [Contrôle à distance](#)
- Lancez une tâche longue sur le [web](#) ou l'[application iOS](#), puis tirez-la dans votre terminal avec `/teleport`

- Remettez une session de terminal à l'[application de bureau](#) avec `/desktop` pour un examen visuel des diffs
- Acheminez les tâches depuis le chat d'équipe : mentionnez `@Claude` dans [Slack](#) avec un rapport de bogue et récupérez une demande de tirage

Utiliser Claude Code partout

Chaque surface se connecte au même moteur Claude Code sous-jacent, donc vos fichiers CLAUDE.md, paramètres et serveurs MCP fonctionnent sur tous.

Au-delà des environnements [Terminal](#), [VS Code](#), [JetBrains](#), [Desktop](#) et [Web](#) ci-dessus, Claude Code s'intègre avec les flux de travail CI/CD, chat et navigateur :

Je veux...	Meilleure option
Continuer une session locale depuis mon téléphone ou un autre appareil	Contrôle à distance
Démarrer une tâche localement, continuer sur mobile	Web ou application Claude iOS
Automatiser les examens de PR et le triage des problèmes	GitHub Actions ou GitLab CI/CD
Obtenir un examen automatique du code sur chaque PR	Examen du code GitHub
Acheminer les rapports de bogues de Slack vers les demandes de tirage	Slack
Déboguer les applications web en direct	Chrome
Créer des agents personnalisés pour vos propres flux de travail	Agent SDK

Étapes suivantes

Une fois que vous avez installé Claude Code, ces guides vous aident à approfondir.

- [Démarrage rapide](#) : parcourez votre première tâche réelle, de l'exploration d'une base de code à la validation d'une correction

- [Stocker les instructions et les mémoires](#) : donnez à Claude des instructions persistantes avec les fichiers CLAUDE.md et la mémoire automatique
- [Flux de travail courants](#) et [meilleures pratiques](#) : modèles pour tirer le meilleur parti de Claude Code
- [Paramètres](#) : personnalisez Claude Code pour votre flux de travail
- [Résolution des problèmes](#) : solutions pour les problèmes courants
- code.claude.com : démos, tarification et détails du produit

Démarrage rapide

Bienvenue dans Claude Code !

Ce guide de démarrage rapide vous permettra d'utiliser l'assistance au codage alimentée par l'IA en quelques minutes. À la fin, vous comprendrez comment utiliser Claude Code pour les tâches de développement courantes.

Avant de commencer

Assurez-vous que vous avez :

- Un terminal ou une invite de commande ouvert
 - Si vous n'avez jamais utilisé le terminal auparavant, consultez le [guide du terminal](#)
- Un projet de code avec lequel travailler
- Un [abonnement Claude](#) (Pro, Max, Teams ou Enterprise), un compte [Claude Console](#), ou un accès via un [fournisseur cloud pris en charge](#)

Note:

Ce guide couvre le CLI du terminal. Claude Code est également disponible sur le [web](#), en tant qu'[application de bureau](#), dans [VS Code](#) et [les IDE JetBrains](#), dans [Slack](#), et en CI/CD avec [GitHub Actions](#) et [GitLab](#). Voir [toutes les interfaces](#).

Étape 1 : Installer 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.

Étape 2 : Se connecter à votre compte

Claude Code nécessite un compte pour être utilisé. Lorsque vous démarrez une session interactive avec la commande `claude`, vous devrez vous connecter :

```
claude
```

```
## Vous serez invité à vous connecter lors de la première utilisation
```

```
/login
```

```
## Suivez les invites pour vous connecter avec votre compte
```

Vous pouvez vous connecter en utilisant l'un de ces types de compte :

- [Claude Pro, Max, Teams ou Enterprise](#) (recommandé)
- [Claude Console](#) (accès API avec crédits prépayés). Lors de la première connexion, un espace de travail « Claude Code » est automatiquement créé dans la Console pour un suivi centralisé des coûts.
- [Amazon Bedrock, Google Vertex AI ou Microsoft Foundry](#) (fournisseurs cloud d'entreprise)

Une fois connecté, vos identifiants sont stockés et vous n'aurez pas besoin de vous reconnecter. Pour changer de compte ultérieurement, utilisez la commande `/login`.

Étape 3 : Démarrer votre première session

Ouvrez votre terminal dans n'importe quel répertoire de projet et démarrez Claude Code :

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

```
claude
```

Vous verrez l'écran de bienvenue de Claude Code avec les informations de votre session, les conversations récentes et les dernières mises à jour. Tapez `/help` pour les commandes disponibles ou `/resume` pour continuer une conversation précédente.

Tip:

Après vous être connecté (Étape 2), vos identifiants sont stockés sur votre système. En savoir plus dans [Gestion des identifiants](#).

Étape 4 : Posez votre première question

Commençons par comprendre votre base de code. Essayez l'une de ces commandes :

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

Claude analysera vos fichiers et fournira un résumé. Vous pouvez également poser des questions plus spécifiques :

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

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

```
explain the folder structure
```

Vous pouvez également demander à Claude ses propres capacités :

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

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

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

Note:

Claude Code lit vos fichiers de projet selon les besoins. Vous n'avez pas à ajouter manuellement du contexte.

Étape 5 : Effectuez votre première modification de code

Maintenant, faisons en sorte que Claude Code fasse du vrai codage. Essayez une tâche simple :

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

Claude Code va :

1. Trouver le fichier approprié
2. Vous montrer les modifications proposées
3. Demander votre approbation
4. Effectuer la modification

Note:

Claude Code demande toujours la permission avant de modifier les fichiers. Vous pouvez approuver les modifications individuelles ou activer le mode « Accepter tout » pour une session.

Étape 6 : Utiliser Git avec Claude Code

Claude Code rend les opérations Git conversationnelles :

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

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

Vous pouvez également demander des opérations Git plus complexes :

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

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

```
help me resolve merge conflicts
```

Étape 7 : Corriger un bug ou ajouter une fonctionnalité

Claude est compétent pour le débogage et l'implémentation de fonctionnalités.

Décrivez ce que vous voulez en langage naturel :


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

Ou corrigez les problèmes existants :

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

Claude Code va :

- Localiser le code pertinent
- Comprendre le contexte
- Implémenter une solution
- Exécuter les tests si disponibles

Étape 8 : Testez d'autres flux de travail courants

Il existe plusieurs façons de travailler avec Claude :

Refactoriser le code

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

Écrire des tests

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

Mettre à jour la documentation

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

Révision de code

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

Tip:

Parlez à Claude comme vous le feriez avec un collègue utile. Décrivez ce que vous voulez réaliser, et il vous aidera à y parvenir.

Commandes essentielles

Voici les commandes les plus importantes pour l'utilisation quotidienne :

Commande	Ce qu'elle fait	Exemple
<code>claude</code>	Démarrer le mode interactif	<code>claude</code>
<code>claude "task"</code>	Exécuter une tâche unique	<code>claude "fix the build error"</code>
<code>claude -p "query"</code>	Exécuter une requête unique, puis quitter	<code>claude -p "explain this function"</code>
<code>claude -c</code>	Continuer la conversation la plus récente dans le répertoire actuel	<code>claude -c</code>
<code>claude -r</code>	Reprendre une conversation précédente	<code>claude -r</code>
<code>claude commit</code>	Créer un commit Git	<code>claude commit</code>
<code>/clear</code>	Effacer l'historique des conversations	<code>/clear</code>
<code>/help</code>	Afficher les commandes disponibles	<code>/help</code>
<code>exit</code> ou Ctrl+C	Quitter Claude Code	<code>exit</code>

Voir la [référence CLI](#) pour une liste complète des commandes.

Conseils professionnels pour les débutants

Pour plus d'informations, voir [les meilleures pratiques](#) et [les flux de travail courants](#).

Au lieu de : ' corriger le bug '

Essayez : ' corriger le bug de connexion où les utilisateurs voient un écran vide après avoir entré des identifiants incorrects '

Divisez les tâches complexes en étapes :

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

Avant de faire des modifications, laissez Claude comprendre votre code :

analyze the database schema

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

- Appuyez sur `?` pour voir tous les raccourcis clavier disponibles
- Utilisez Tab pour la complétion des commandes
- Appuyez sur `↑` pour l'historique des commandes
- Tapez `/` pour voir toutes les commandes et skills

Prochaines étapes

Maintenant que vous avez appris les bases, explorez des fonctionnalités plus avancées :

- [Comment fonctionne Claude Code](#) Comprendre la boucle agentique, les outils intégrés et comment Claude Code interagit avec votre projet
- [Meilleures pratiques](#) Obtenez de meilleurs résultats avec un prompting efficace et une configuration de projet appropriée
- [Flux de travail courants](#) Guides étape par étape pour les tâches courantes
- [Étendre Claude Code](#) Personnalisez avec CLAUDE.md, skills, hooks, MCP et bien plus

Obtenir de l'aide

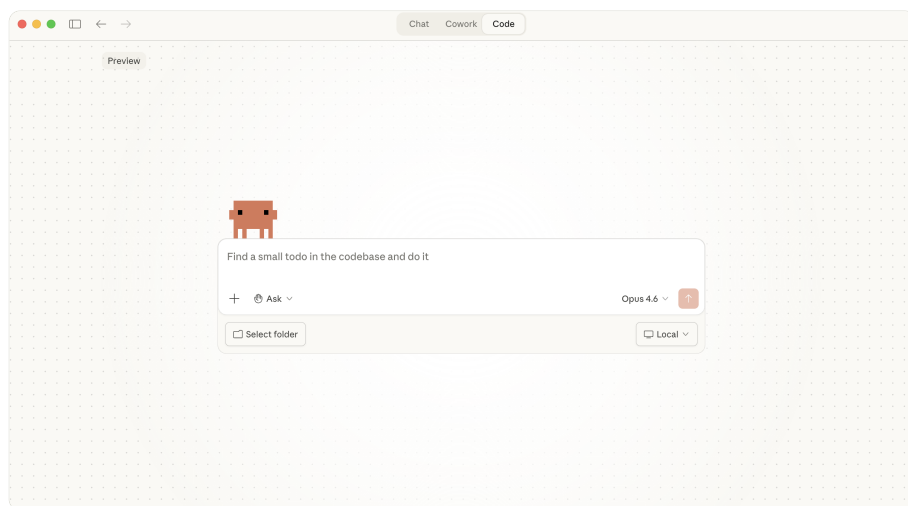
- **Dans Claude Code** : Tapez `/help` ou demandez « how do I... »
- **Documentation** : Vous êtes ici ! Parcourez les autres guides
- **Communauté** : Rejoignez notre [Discord](#) pour des conseils et du support

Démarrer avec l'application de bureau

Installez Claude Code sur le bureau et commencez votre première session de codage

L'application de bureau vous donne accès à Claude Code avec une interface graphique : examen des différences visuelles, aperçu en direct de l'application, surveillance des PR GitHub avec fusion automatique, sessions parallèles avec isolation Git worktree, tâches planifiées et la possibilité d'exécuter des tâches à distance. Aucun terminal requis.

Cette page vous guide dans l'installation de l'application et le démarrage de votre première session. Si vous êtes déjà configuré, consultez [Utiliser Claude Code Desktop](#) pour la référence complète.



L'interface Claude Code Desktop montrant l'onglet Code sélectionné, avec une boîte de saisie, un sélecteur de mode de permission défini sur Demander les permissions, un sélecteur de modèle, un sélecteur de dossier et l'option Environnement local

L'application de bureau a trois onglets :

- **Chat** : Conversation générale sans accès aux fichiers, similaire à claude.ai.
- **Cowork** : Un agent autonome en arrière-plan qui travaille sur des tâches dans une VM cloud avec son propre environnement. Il peut fonctionner indépendamment pendant que vous faites autre chose.

- **Code** : Un assistant de codage interactif avec accès direct à vos fichiers locaux. Vous examinez et approuvez chaque modification en temps réel.

Chat et Cowork sont couverts dans les [articles d'assistance Claude Desktop](#). Cette page se concentre sur l'onglet **Code**.

Note:

Claude Code nécessite un [abonnement Pro, Max, Teams ou Enterprise](#).

Installer

Step 1: Télécharger l'application

Téléchargez Claude pour votre plateforme.

- [macOS](#) Build universel pour Intel et Apple Silicon
- [Windows](#) Pour les processeurs x64

Pour Windows ARM64, [téléchargez ici](#).

Linux n'est actuellement pas supporté.

Step 2: Se connecter

Lancez Claude à partir de votre dossier Applications (macOS) ou du menu Démarrer (Windows). Connectez-vous avec votre compte Anthropic.

Step 3: Ouvrir l'onglet Code

Cliquez sur l'onglet **Code** en haut au centre. Si cliquer sur Code vous invite à mettre à niveau, vous devez d'abord [vous abonner à un plan payant](#). S'il vous invite à vous connecter en ligne, complétez la connexion et redémarrez l'application. Si vous voyez une erreur 403, consultez [dépannage de l'authentification](#).

L'application de bureau inclut Claude Code. Vous n'avez pas besoin d'installer Node.js ou la CLI séparément. Pour utiliser `claude` depuis le terminal, installez la CLI séparément. Consultez [Démarrer avec la CLI](#).

Commencer votre première session

Avec l'onglet Code ouvert, choisissez un projet et donnez à Claude quelque chose à faire.

Step 1: Choisir un environnement et un dossier

Sélectionnez **Local** pour exécuter Claude sur votre machine en utilisant vos fichiers directement. Cliquez sur **Sélectionner le dossier** et choisissez votre répertoire de projet.

Tip:

Commencez par un petit projet que vous connaissez bien. C'est le moyen le plus rapide de voir ce que Claude Code peut faire. Sur Windows, [Git](#) doit être installé pour que les sessions locales fonctionnent. La plupart des Mac incluent Git par défaut.

Vous pouvez également sélectionner :

- **Remote** : Exécutez les sessions sur l'infrastructure cloud d'Anthropic qui continue même si vous fermez l'application. Les sessions distantes utilisent la même infrastructure que [Claude Code sur le web](#).
- **SSH** : Connectez-vous à une machine distante via SSH (vos propres serveurs, VM cloud ou conteneurs de développement). Claude Code doit être installé sur la machine distante.

Step 2: Choisir un modèle

Sélectionnez un modèle dans la liste déroulante à côté du bouton d'envoi. Consultez [modèles](#) pour une comparaison d'Opus, Sonnet et Haiku. Vous ne pouvez pas changer le modèle après le démarrage de la session.

Step 3: Dire à Claude ce qu'il faut faire

Tapez ce que vous voulez que Claude fasse :

- Trouver un commentaire TODO et le corriger
- Ajouter des tests pour la fonction principale
- Créer un CLAUDE.md avec des instructions pour cette base de code

Une [session](#) est une conversation avec Claude sur votre code. Chaque session suit son propre contexte et ses modifications, vous pouvez donc travailler sur plusieurs tâches sans qu'elles n'interfèrent les unes avec les autres.

Step 4: Examiner et accepter les modifications

Par défaut, l'onglet Code démarre en [mode Demander les permissions](#), où Claude propose des modifications et attend votre approbation avant de les appliquer. Vous verrez :

1. Une [vue de différence](#) montrant exactement ce qui changera dans chaque fichier
2. Des boutons Accepter/Rejeter pour approuver ou refuser chaque modification

3. Des mises à jour en temps réel pendant que Claude travaille sur votre demande

Si vous rejetez une modification, Claude vous demandera comment vous aimeriez procéder différemment. Vos fichiers ne sont pas modifiés tant que vous n'acceptez pas.

Et maintenant ?

Vous avez fait votre première modification. Pour la référence complète sur tout ce que Desktop peut faire, consultez [Utiliser Claude Code Desktop](#). Voici quelques choses à essayer ensuite.

Interrompre et diriger. Vous pouvez interrompre Claude à tout moment. S'il prend la mauvaise direction, cliquez sur le bouton d'arrêt ou tapez votre correction et appuyez sur **Entrée**. Claude arrête ce qu'il fait et s'ajuste en fonction de votre entrée. Vous n'avez pas besoin d'attendre qu'il finisse ou de recommencer.

Donner à Claude plus de contexte. Tapez `@filename` dans la boîte de saisie pour extraire un fichier spécifique dans la conversation, joignez des images et des PDF en utilisant le bouton de pièce jointe, ou glissez-déposez des fichiers directement dans la saisie. Plus Claude a de contexte, meilleurs sont les résultats. Consultez [Ajouter des fichiers et du contexte](#).

Utiliser les skills pour les tâches répétables. Tapez `/` ou cliquez sur `+` → **Slash commands** pour parcourir les [commandes intégrées](#), les [skills personnalisés](#) et les skills de plugin. Les skills sont des invites réutilisables que vous pouvez invoquer chaque fois que vous en avez besoin, comme des listes de contrôle d'examen de code ou des étapes de déploiement.

Examiner les modifications avant de valider. Après que Claude modifie les fichiers, un indicateur `+12 -1` apparaît. Cliquez dessus pour ouvrir la [vue de différence](#), examinez les modifications fichier par fichier et commentez des lignes spécifiques. Claude lit vos commentaires et révisé. Cliquez sur **Examiner le code** pour que Claude évalue lui-même les différences et laisse des suggestions en ligne.

Ajuster le contrôle que vous avez. Votre [mode de permission](#) contrôle l'équilibre. Demander les permissions (par défaut) nécessite une approbation avant chaque modification. Auto accept edits accepte automatiquement les modifications de fichiers pour une itération plus rapide. Plan mode permet à Claude de cartographier une approche sans toucher à aucun fichier, ce qui est utile avant une grande refonte.

Ajouter des plugins pour plus de capacités. Cliquez sur le bouton `+` à côté de la boîte de saisie et sélectionnez **Plugins** pour parcourir et installer des [plugins](#) qui ajoutent des skills, des agents, des MCP servers et plus.

Prévisualiser votre application. Cliquez sur la liste déroulante **Preview** pour exécuter votre serveur de développement directement dans le bureau. Claude peut voir l'application en cours d'exécution, tester les points de terminaison, inspecter les journaux et itérer sur ce qu'il voit. Consultez [Prévisualiser votre application](#).

Suivre votre demande de tirage. Après avoir ouvert une PR, Claude Code surveille les résultats des vérifications CI et peut corriger automatiquement les défaillances ou fusionner la PR une fois que toutes les vérifications sont réussies. Consultez [Surveiller l'état de la demande de tirage](#).

Mettre Claude sur un calendrier. Configurez des [tâches planifiées](#) pour exécuter Claude automatiquement de manière récurrente : un examen de code quotidien chaque matin, un audit de dépendances hebdomadaire ou un briefing qui extrait de vos outils connectés.

Augmenter l'échelle quand vous êtes prêt. Ouvrez des [sessions parallèles](#) à partir de la barre latérale pour travailler sur plusieurs tâches à la fois, chacune dans son propre Git worktree. Envoyez des [travaux de longue durée vers le cloud](#) pour qu'ils continuent même si vous fermez l'application, ou [continuez une session sur le web ou dans votre IDE](#) si une tâche prend plus de temps que prévu. [Connectez des outils externes](#) comme GitHub, Slack et Linear pour réunir votre flux de travail.

Venant de la CLI ?

Desktop exécute le même moteur que la CLI avec une interface graphique. Vous pouvez exécuter les deux simultanément sur le même projet, et ils partagent la configuration (fichiers CLAUDE.md, MCP servers, hooks, skills et paramètres). Pour une comparaison complète des fonctionnalités, des équivalents de drapeaux et de ce qui n'est pas disponible dans Desktop, consultez [Comparaison CLI](#).

Prochaines étapes

- [Utiliser Claude Code Desktop](#) : modes de permission, sessions parallèles, vue de différence, connecteurs et configuration d'entreprise
- [Dépannage](#) : solutions aux erreurs courantes et problèmes de configuration
- [Meilleures pratiques](#) : conseils pour rédiger des invites efficaces et tirer le meilleur parti de Claude Code
- [Flux de travail courants](#) : tutoriels pour le débogage, la refonte, les tests et plus

Configurer Claude Code

Installez, authentifiez-vous et commencez à utiliser Claude Code sur votre machine de développement.

Configuration requise

- **Système d'exploitation :**
 - macOS 13.0+
 - Windows 10 1809+ ou Windows Server 2019+ ([voir les notes de configuration](#))
 - Ubuntu 20.04+
 - Debian 10+
 - Alpine Linux 3.19+ ([dépendances supplémentaires requises](#))
- **Matériel :** 4 Go+ de RAM
- **Réseau :** Connexion Internet requise (voir [configuration réseau](#))
- **Shell :** Fonctionne mieux avec Bash ou Zsh
- **Localisation :** [Pays supportés par Anthropic](#)

Dépendances supplémentaires

- **ripgrep :** Généralement inclus avec Claude Code. Si la recherche échoue, voir [dépannage de la recherche](#).
- **Node.js 18+ :** Requis uniquement pour [l'installation npm dépréciée](#)

Installation

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

Native Install (Recommended)

macOS, Linux, WSL:

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

Windows PowerShell:

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

Windows CMD:

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

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

Info:

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

Homebrew

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

Info:

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

WinGet

```
winget install Anthropic.ClaudeCode
```

Info:

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

Une fois le processus d'installation terminé, accédez à votre projet et démarrez Claude Code :

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

Si vous rencontrez des problèmes lors de l'installation, consultez le [guide de dépannage](#).

Tip:

Exécutez `claude doctor` après l'installation pour vérifier votre type d'installation et votre version.

Configuration spécifique à la plateforme

Windows : Exécutez Claude Code en mode natif (nécessite [Git Bash](#)) ou dans WSL. WSL 1 et WSL 2 sont tous deux supportés, mais WSL 1 a un support limité et ne supporte pas les fonctionnalités comme le sandboxing de l'outil Bash.

Alpine Linux et autres distributions basées sur musl/uClibc :

Le programme d'installation natif sur Alpine et autres distributions basées sur musl/uClibc nécessite `libgcc`, `libstdc++` et `ripgrep`. Installez-les à l'aide du gestionnaire de paquets de votre distribution, puis définissez `USE_BUILTIN_RIPGREP=0`.

Sur Alpine :

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

Authentification

Pour les particuliers

1. **Plan Claude Pro ou Max** (recommandé) : Abonnez-vous au [plan Pro ou Max](#) de Claude pour un abonnement unifié qui inclut à la fois Claude Code et Claude sur le web. Gérez votre compte en un seul endroit et connectez-vous avec votre compte Claude.ai.
2. **Claude Console** : Connectez-vous via la [Claude Console](#) et complétez le processus OAuth. Nécessite une facturation active dans la Console Anthropic. Un espace de travail « Claude Code » est automatiquement créé pour le suivi de l'utilisation et la gestion des coûts. Vous ne pouvez pas créer de clés API pour l'espace de travail Claude Code ; il est dédié exclusivement à l'utilisation de Claude Code.

Pour les équipes et organisations

1. **Claude for Teams ou Enterprise** (recommandé) : Abonnez-vous à [Claude for Teams](#) ou [Claude for Enterprise](#) pour une facturation centralisée, la gestion d'équipe et l'accès à la fois à Claude Code et Claude sur le web. Les membres de l'équipe se connectent avec leurs comptes Claude.ai.
2. **Claude Console avec facturation d'équipe** : Configurez une organisation [Claude Console](#) partagée avec facturation d'équipe. Invitez les membres de l'équipe et attribuez des rôles pour le suivi de l'utilisation.
3. **Fournisseurs cloud** : Configurez Claude Code pour utiliser [Amazon Bedrock](#), [Google Vertex AI](#) ou [Microsoft Foundry](#) pour les déploiements avec votre infrastructure cloud existante.

Installer une version spécifique

Le programme d'installation natif accepte soit un numéro de version spécifique, soit un canal de version (`latest` ou `stable`). Le canal que vous choisissez au moment de l'installation devient votre canal par défaut pour les mises à jour automatiques. Voir [Configurer le canal de version](#) pour plus d'informations.

Pour installer la dernière version (par défaut) :

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
```

Pour installer la version stable :

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
```

Pour installer un numéro de version spécifique :

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
```

Intégrité binaire et signature de code

- Les sommes de contrôle SHA256 pour toutes les plateformes sont publiées dans les manifestes de version, actuellement situés à <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json> (exemple : remplacez {VERSION} par 2.0.30)
- Les binaires signés sont distribués pour les plateformes suivantes :
 - macOS : Signé par ' Anthropic PBC ' et notarié par Apple
 - Windows : Signé par ' Anthropic, PBC '

Installation NPM (dépréciée)

L'installation NPM est dépréciée. Utilisez la méthode [d'installation native](#) si possible. Pour migrer une installation npm existante vers la version native, exécutez `claude install`.

Installation npm globale

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

Warning:

N'utilisez PAS `sudo npm install -g` car cela peut entraîner des problèmes de permissions et des risques de sécurité. Si vous rencontrez des erreurs de permission, voir [dépannage des erreurs de permission](#) pour les solutions recommandées.

Configuration Windows

Option 1 : Claude Code dans WSL

- WSL 1 et WSL 2 sont tous deux supportés
- WSL 2 supporte le [sandboxing](#) pour une sécurité renforcée. WSL 1 ne supporte pas le sandboxing.

Option 2 : Claude Code sur Windows natif avec Git Bash

- Nécessite [Git for Windows](#)
- Pour les installations Git portables, spécifiez le chemin vers votre `bash.exe` :

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

Mettre à jour Claude Code

Mises à jour automatiques

Claude Code se met à jour automatiquement pour vous assurer que vous disposez des dernières fonctionnalités et correctifs de sécurité.

- **Vérifications de mise à jour** : Effectuées au démarrage et périodiquement pendant l'exécution
- **Processus de mise à jour** : Télécharge et installe automatiquement en arrière-plan
- **Notifications** : Vous verrez une notification lorsque les mises à jour sont installées

- **Application des mises à jour** : Les mises à jour prennent effet la prochaine fois que vous démarrez Claude Code

Note:

Les installations Homebrew et WinGet ne se mettent pas à jour automatiquement. Utilisez `brew upgrade claude-code` ou `winget upgrade Anthropic.ClaudeCode` pour mettre à jour manuellement.

Problème connu : Claude Code peut vous notifier des mises à jour avant que la nouvelle version soit disponible dans ces gestionnaires de paquets. Si une mise à niveau échoue, attendez et réessayez plus tard.

Configurer le canal de version

Configurez le canal de version que Claude Code suit pour les mises à jour automatiques et `claude update` avec le paramètre `autoUpdatesChannel` :

- `"latest"` (par défaut) : Recevez les nouvelles fonctionnalités dès qu'elles sont publiées
- `"stable"` : Utilisez une version qui a généralement environ une semaine, en ignorant les versions avec des régressions majeures

Configurez ceci via `/config` → **Canal de mise à jour automatique**, ou ajoutez-le à votre [fichier settings.json](#) :

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

Pour les déploiements d'entreprise, vous pouvez appliquer un canal de version cohérent dans votre organisation en utilisant les [paramètres gérés](#).

Désactiver les mises à jour automatiques

Définissez la variable d'environnement `DISABLE_AUTOUPDATER` dans votre shell ou [fichier settings.json](#) :

```
export DISABLE_AUTOUPDATER=1
```

Mettre à jour manuellement

```
claude update
```

Désinstaller Claude Code

Si vous devez désinstaller Claude Code, suivez les instructions pour votre méthode d'installation.

Installation native

Supprimez le binaire Claude Code et les fichiers de version :

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"
```

Installation Homebrew

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

Installation WinGet

```
winget uninstall Anthropic.ClaudeCode
```


Installation NPM

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

Nettoyer les fichiers de configuration (optionnel)

Warning:

La suppression des fichiers de configuration supprimera tous vos paramètres, outils autorisés, configurations de serveur MCP et l'historique de session.

Pour supprimer les paramètres et données en cache de Claude Code :

macOS, Linux, WSL :

```
## Supprimer les paramètres utilisateur et l'état
rm -rf ~/.claude
rm ~/.claude.json

## Supprimer les paramètres spécifiques au projet (exécutez depuis votre
répertoire de projet)
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell :

```
## Supprimer les paramètres utilisateur et l'état
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

## Supprimer les paramètres spécifiques au projet (exécutez depuis votre
répertoire de projet)
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

Windows CMD :

REM Supprimer les paramètres utilisateur et l'état

```
rmdir /s /q "%USERPROFILE%\.claude"
```

```
del "%USERPROFILE%\.claude.json"
```

REM Supprimer les paramètres spécifiques au projet (exécutez depuis votre répertoire de projet)

```
rmdir /s /q ".claude"
```

```
del ".mcp.json"
```

Comment fonctionne Claude Code

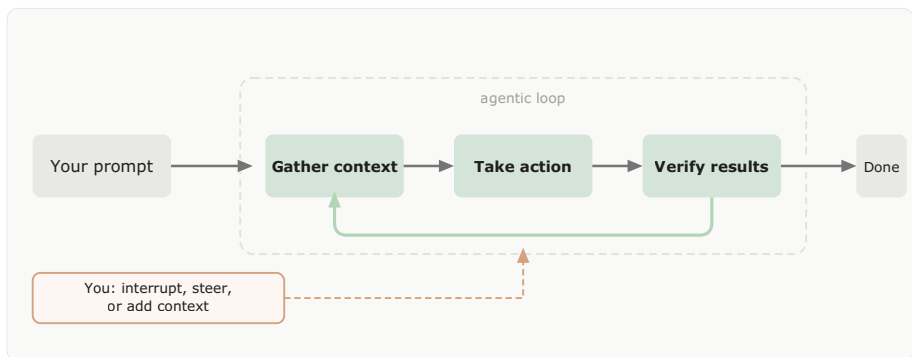
Comprenez la boucle agentic, les outils intégrés et comment Claude Code interagit avec votre projet.

Claude Code est un assistant agentif qui s'exécute dans votre terminal. Bien qu'il excelle dans la programmation, il peut vous aider pour tout ce que vous pouvez faire depuis la ligne de commande : rédiger de la documentation, exécuter des builds, rechercher des fichiers, faire des recherches sur des sujets, et bien plus.

Ce guide couvre l'architecture centrale, les capacités intégrées, et [des conseils pour travailler efficacement avec Claude Code](#). Pour des procédures pas à pas, consultez [Workflows courants](#). Pour les fonctionnalités d'extensibilité comme les skills, MCP et hooks, consultez [Étendre Claude Code](#).

La boucle agentic

Lorsque vous donnez une tâche à Claude, il travaille à travers trois phases : **rassembler le contexte**, **agir**, et **vérifier les résultats**. Ces phases se mélangent ensemble. Claude utilise des outils tout au long du processus, qu'il s'agisse de rechercher des fichiers pour comprendre votre code, d'éditer pour apporter des modifications, ou d'exécuter des tests pour vérifier son travail.



La boucle agentic : Votre prompt conduit Claude à rassembler le contexte, agir, vérifier les résultats, et répéter jusqu'à ce que la tâche soit terminée. Vous pouvez interrompre à tout moment.

La boucle s'adapte à ce que vous demandez. Une question sur votre base de code pourrait nécessiter uniquement la collecte de contexte. Une correction de bug parcourt les trois phases à plusieurs reprises. Une refactorisation pourrait impliquer une vérification extensive. Claude décide ce que chaque étape nécessite en fonction de ce qu'il a appris à l'étape précédente, en enchaînant des dizaines d'actions ensemble et en se corrigeant en cours de route.

Vous faites également partie de cette boucle. Vous pouvez interrompre à tout moment pour orienter Claude dans une direction différente, fournir un contexte supplémentaire, ou lui demander d'essayer une approche différente. Claude fonctionne de manière autonome mais reste réactif à votre contribution.

La boucle agentive est alimentée par deux composants : [les modèles](#) qui raisonnent et [les outils](#) qui agissent. Claude Code sert de **harnais agentif** autour de Claude : il fournit les outils, la gestion du contexte, et l'environnement d'exécution qui transforment un modèle de langage en un agent de programmation capable.

Modèles

Claude Code utilise les modèles Claude pour comprendre votre code et raisonner sur les tâches. Claude peut lire du code dans n'importe quel langage, comprendre comment les composants se connectent, et déterminer ce qui doit changer pour accomplir votre objectif. Pour les tâches complexes, il divise le travail en étapes, les exécute, et s'ajuste en fonction de ce qu'il apprend.

[Plusieurs modèles](#) sont disponibles avec des compromis différents. Sonnet gère bien la plupart des tâches de programmation. Opus fournit un raisonnement plus fort pour les décisions architecturales complexes. Basculez avec `/model` pendant une session ou commencez avec `claude --model <name>`.

Lorsque ce guide dit « Claude choisit » ou « Claude décide », c'est le modèle qui effectue le raisonnement.

Outils

Les outils sont ce qui rend Claude Code agentif. Sans outils, Claude ne peut que répondre avec du texte. Avec les outils, Claude peut agir : lire votre code, éditer des fichiers, exécuter des commandes, rechercher sur le web, et interagir avec des services externes. Chaque utilisation d'outil retourne des informations qui se réintègrent dans la boucle, informant la décision suivante de Claude.

Les outils intégrés se divisent généralement en cinq catégories, chacune représentant un type d'agentivité différent.

Catégorie	Ce que Claude peut faire
Opérations sur les fichiers	Lire des fichiers, éditer du code, créer de nouveaux fichiers, renommer et réorganiser
Recherche	Trouver des fichiers par motif, rechercher du contenu avec regex, explorer les bases de code
Exécution	Exécuter des commandes shell, démarrer des serveurs, exécuter des tests, utiliser git
Web	Rechercher sur le web, récupérer de la documentation, rechercher des messages d'erreur
Intelligence du code	Voir les erreurs de type et les avertissements après les éditions, accéder aux définitions, trouver les références (nécessite les plugins d'intelligence du code)

Ce sont les capacités principales. Claude dispose également d'outils pour générer des subagents, vous poser des questions, et d'autres tâches d'orchestration. Consultez [Outils disponibles pour Claude](#) pour la liste complète.

Claude choisit les outils à utiliser en fonction de votre prompt et de ce qu'il apprend en cours de route. Lorsque vous dites « corriger les tests qui échouent », Claude pourrait :

1. Exécuter la suite de tests pour voir ce qui échoue
2. Lire la sortie d'erreur
3. Rechercher les fichiers source pertinents
4. Lire ces fichiers pour comprendre le code
5. Éditer les fichiers pour corriger le problème
6. Exécuter les tests à nouveau pour vérifier

Chaque utilisation d'outil donne à Claude de nouvelles informations qui informent l'étape suivante. C'est la boucle agentic en action.

Étendre les capacités de base : Les outils intégrés sont la fondation. Vous pouvez étendre ce que Claude sait avec [les skills](#), vous connecter à des services externes avec [MCP](#), automatiser les workflows avec [hooks](#), et déléguer des tâches à [des subagents](#). Ces extensions forment une couche au-dessus de la boucle agentic principale. Consultez [Étendre Claude Code](#) pour des conseils sur le choix de la bonne extension pour vos besoins.

Ce que Claude peut accéder

Ce guide se concentre sur le terminal. Claude Code s'exécute également dans [VS Code](#), [les IDEs JetBrains](#), et d'autres environnements.

Lorsque vous exécutez `claude` dans un répertoire, Claude Code accède à :

- **Votre projet.** Les fichiers de votre répertoire et sous-répertoires, plus les fichiers ailleurs avec votre permission.
- **Votre terminal.** N'importe quelle commande que vous pourriez exécuter : outils de build, git, gestionnaires de paquets, utilitaires système, scripts. Si vous pouvez le faire depuis la ligne de commande, Claude aussi.
- **Votre état git.** La branche actuelle, les modifications non validées, et l'historique récent des commits.
- **Votre `CLAUDE.md`.** Un fichier markdown où vous stockez les instructions spécifiques au projet, les conventions, et le contexte que Claude devrait connaître à chaque session.
- **La mémoire automatique.** Les apprentissages que Claude sauvegarde automatiquement au fur et à mesure que vous travaillez, comme les motifs de projet et vos préférences. Les 200 premières lignes de `MEMORY.md` sont chargées au début de chaque session.
- **Les extensions que vous configurez.** [Les serveurs MCP](#) pour les services externes, [les skills](#) pour les workflows, [les subagents](#) pour le travail délégué, et [Claude dans Chrome](#) pour l'interaction avec le navigateur.

Parce que Claude voit votre projet entier, il peut travailler à travers celui-ci. Lorsque vous demandez à Claude de « corriger le bug d'authentification », il recherche les fichiers pertinents, lit plusieurs fichiers pour comprendre le contexte, effectue des éditions coordonnées à travers eux, exécute des tests pour vérifier la correction, et valide les modifications si vous le demandez. C'est différent des assistants de code en ligne qui ne voient que le fichier actuel.

Environnements et interfaces

La boucle agentive, les outils, et les capacités décrites ci-dessus sont les mêmes partout où vous utilisez Claude Code. Ce qui change, c'est où le code s'exécute et comment vous interagissez avec lui.

Environnements d'exécution

Claude Code s'exécute dans trois environnements, chacun avec des compromis différents pour l'endroit où votre code s'exécute.

Environnement	Où le code s'exécute	Cas d'usage
Local	Votre machine	Par défaut. Accès complet à vos fichiers, outils, et environnement
Cloud	VMs gérées par Anthropic	Déléguer des tâches, travailler sur des repos que vous n'avez pas localement
Contrôle à distance	Votre machine, contrôlée depuis un navigateur	Utiliser l'interface web tout en gardant tout local

Interfaces

Vous pouvez accéder à Claude Code via le terminal, [l'application de bureau](#), [les extensions IDE](#), [claude.ai/code](#), [Contrôle à distance](#), [Slack](#), et [les pipelines CI/CD](#). L'interface détermine comment vous voyez et interagissez avec Claude, mais la boucle agentive sous-jacente est identique. Consultez [Utiliser Claude Code partout](#) pour la liste complète.

Travailler avec les sessions

Claude Code sauvegarde votre conversation localement au fur et à mesure que vous travaillez. Chaque message, utilisation d'outil, et résultat est stocké, ce qui permet [de revenir en arrière](#), [de reprendre et de forker](#) les sessions. Avant que Claude ne fasse des modifications de code, il prend également un snapshot des fichiers affectés afin que vous puissiez revenir en arrière si nécessaire.

Les sessions sont indépendantes. Chaque nouvelle session commence avec une fenêtre de contexte fraîche, sans l'historique de conversation des sessions précédentes. Claude peut persister les apprentissages à travers les sessions en utilisant [la mémoire automatique](#), et vous pouvez ajouter vos propres instructions persistantes dans [CLAUDE.md](#).

Travailler à travers les branches

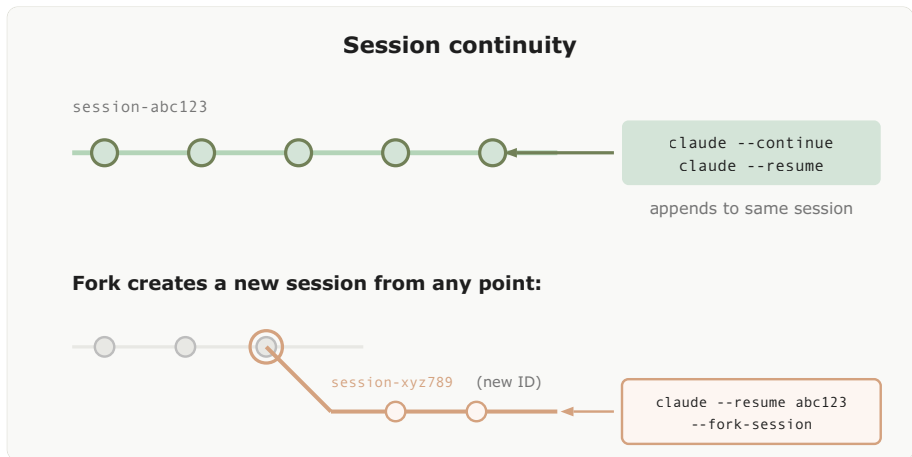
Chaque conversation Claude Code est une session liée à votre répertoire actuel. Lorsque vous reprenez, vous ne voyez que les sessions de ce répertoire.

Claude voit les fichiers de votre branche actuelle. Lorsque vous changez de branche, Claude voit les fichiers de la nouvelle branche, mais votre historique de conversation reste le même. Claude se souvient de ce que vous avez discuté même après avoir changé de branche.

Puisque les sessions sont liées aux répertoires, vous pouvez exécuter des sessions Claude parallèles en utilisant [git worktrees](#), qui créent des répertoires séparés pour les branches individuelles.

Reprendre ou forker les sessions

Lorsque vous reprenez une session avec `claude --continue` ou `claude --resume`, vous reprenez là où vous vous étiez arrêté en utilisant le même ID de session. Les nouveaux messages s'ajoutent à la conversation existante. Votre historique de conversation complet est restauré, mais les permissions scoped à la session ne le sont pas. Vous devrez les réapprouver.



Continuité de session : reprendre continue la même session, forker crée une nouvelle branche avec un nouvel ID.

Pour créer une branche et essayer une approche différente sans affecter la session d'origine, utilisez le flag `--fork-session` :

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

Cela crée un nouvel ID de session tout en préservant l'historique de conversation jusqu'à ce point. La session d'origine reste inchangée. Comme pour reprendre, les sessions forkées n'héritent pas des permissions scoped à la session.

Même session dans plusieurs terminaux : Si vous reprenez la même session dans plusieurs terminaux, les deux terminaux écrivent dans le même fichier de session. Les messages des deux sont entrelacés, comme deux personnes écrivant dans le même carnet. Rien ne se corrompt, mais la conversation devient confuse. Chaque terminal ne voit que ses propres messages pendant la session, mais si vous reprenez cette session plus tard, vous verrez tout entrelacé. Pour le travail parallèle à partir du même point de départ, utilisez `--fork-session` pour donner à chaque terminal sa propre session propre.

La fenêtre de contexte

La fenêtre de contexte de Claude contient votre historique de conversation, le contenu des fichiers, les sorties de commande, [CLAUDE.md](#), les skills chargés, et les instructions système. Au fur et à mesure que vous travaillez, le contexte se remplit. Claude compacte automatiquement, mais les instructions du début de la conversation peuvent être perdues. Mettez les règles persistantes dans [CLAUDE.md](#), et exécutez `/context` pour voir ce qui utilise l'espace.

Lorsque le contexte se remplit

Claude Code gère le contexte automatiquement à mesure que vous approchez de la limite. Il efface d'abord les sorties d'outils plus anciennes, puis résume la conversation si nécessaire. Vos demandes et les extraits de code clés sont préservés ; les instructions détaillées du début de la conversation peuvent être perdues. Mettez les règles persistantes dans [CLAUDE.md](#) plutôt que de compter sur l'historique de conversation.

Pour contrôler ce qui est préservé pendant la compaction, ajoutez une section « Compact Instructions » à [CLAUDE.md](#) ou exécutez `/compact` avec un focus (comme `/compact focus on the API changes`).

Exécutez `/context` pour voir ce qui utilise l'espace. Les serveurs MCP ajoutent des définitions d'outils à chaque demande, donc quelques serveurs peuvent consommer un contexte significatif avant que vous ne commenciez à travailler. Exécutez `/mcp` pour vérifier les coûts par serveur.

Gérer le contexte avec les skills et les subagents

Au-delà de la compaction, vous pouvez utiliser d'autres fonctionnalités pour contrôler ce qui se charge dans le contexte.

[Les skills](#) se chargent à la demande. Claude voit les descriptions des skills au démarrage de la session, mais le contenu complet ne se charge que lorsqu'un skill est utilisé. Pour les skills que vous invoquez manuellement, définissez `disable-model-invocation: true` pour garder les descriptions hors du contexte jusqu'à ce que vous en ayez besoin.

[Les subagents](#) obtiennent leur propre contexte frais, complètement séparé de votre conversation principale. Leur travail ne gonfle pas votre contexte. Une fois terminés, ils retournent un résumé. Cet isolement est pourquoi les subagents aident avec les sessions longues.

Consultez [les coûts de contexte](#) pour ce que chaque fonctionnalité coûte, et [réduire l'utilisation des tokens](#) pour des conseils sur la gestion du contexte.

Rester en sécurité avec les checkpoints et les permissions

Claude dispose de deux mécanismes de sécurité : les checkpoints vous permettent d'annuler les modifications de fichiers, et les permissions contrôlent ce que Claude peut faire sans demander.

Annuler les modifications avec les checkpoints

Chaque édition de fichier est réversible. Avant que Claude n'écrive un fichier, il prend un snapshot du contenu actuel. Si quelque chose se passe mal, appuyez deux fois sur **Esc** pour revenir à un état précédent, ou demandez à Claude d'annuler.

Les checkpoints sont locaux à votre session, séparés de git. Ils ne couvrent que les modifications de fichiers. Les actions qui affectent les systèmes distants (bases de données, APIs, déploiements) ne peuvent pas être checkpointées, c'est pourquoi Claude demande avant d'exécuter des commandes avec des effets secondaires externes.

Contrôler ce que Claude peut faire

Appuyez sur **Shift+Tab** pour parcourir les modes de permission :

- **Par défaut** : Claude demande avant les éditions de fichiers et les commandes shell
- **Auto-accepter les éditions** : Claude écrit les fichiers sans demander, demande toujours pour les commandes
- **Plan Mode** : Claude utilise uniquement les outils en lecture seule, créant un plan que vous pouvez approuver avant l'exécution

Vous pouvez également autoriser des commandes spécifiques dans `.claude/settings.json` afin que Claude ne demande pas à chaque fois. C'est utile pour les commandes de confiance comme `npm test` ou `git status`. Les paramètres peuvent être scoped des politiques à l'échelle de l'organisation jusqu'aux préférences personnelles. Consultez [Permissions](#) pour plus de détails.

Travailler efficacement avec Claude Code

Ces conseils vous aident à obtenir de meilleurs résultats avec Claude Code.

Demander de l'aide à Claude Code

Claude Code peut vous enseigner comment l'utiliser. Posez des questions comme « comment configurer les hooks ? » ou « quelle est la meilleure façon de structurer mon CLAUDE.md ? » et Claude expliquera.

Les commandes intégrées vous guident également à travers la configuration :

- `/init` vous guide à travers la création d'un CLAUDE.md pour votre projet
- `/agents` vous aide à configurer des subagents personnalisés
- `/doctor` diagnostique les problèmes courants avec votre installation

C'est une conversation

Claude Code est conversationnel. Vous n'avez pas besoin de prompts parfaits. Commencez par ce que vous voulez, puis affinez :

Corriger le bug de connexion

[Claude enquête, essaie quelque chose]

Ce n'est pas tout à fait correct. Le problème est dans la gestion de session.

[Claude ajuste son approche]

Lorsque la première tentative n'est pas correcte, vous ne recommencez pas. Vous itérez.

Interrompre et orienter

Vous pouvez interrompre Claude à tout moment. S'il va dans la mauvaise direction, tapez simplement votre correction et appuyez sur Entrée. Claude arrêtera ce qu'il fait et ajustera son approche en fonction de votre contribution. Vous n'avez pas à attendre qu'il finisse ou à recommencer.

Être spécifique dès le départ

Plus votre prompt initial est précis, moins de corrections vous aurez besoin. Référez-vous des fichiers spécifiques, mentionnez les contraintes, et pointez vers des motifs d'exemple.

```
Le flux de paiement est cassé pour les utilisateurs avec des cartes expirées.  
Vérifiez src/payments/ pour le problème, en particulier l'actualisation des  
tokens.  
Écrivez d'abord un test qui échoue, puis corrigez-le.
```

Les prompts vagues fonctionnent, mais vous passerez plus de temps à orienter. Les prompts spécifiques comme celui ci-dessus réussissent souvent à la première tentative.

Donner à Claude quelque chose à vérifier

Claude fonctionne mieux lorsqu'il peut vérifier son propre travail. Incluez des cas de test, collez des captures d'écran de l'interface utilisateur attendue, ou définissez la sortie que vous voulez.

```
Implémenter validateEmail. Cas de test : 'user@example.com' → true,  
'invalid' → false, 'user@.com' → false. Exécutez les tests après.
```

Pour le travail visuel, collez une capture d'écran de la conception et demandez à Claude de comparer son implémentation avec celle-ci.

Explorer avant d'implémenter

Pour les problèmes complexes, séparez la recherche de la programmation. Utilisez le plan mode (**Shift+Tab** deux fois) pour analyser d'abord la base de code :

```
Lire src/auth/ et comprendre comment nous gérons les sessions.  
Ensuite, créer un plan pour ajouter le support OAuth.
```

Examinez le plan, affinez-le à travers la conversation, puis laissez Claude implémenter. Cette approche en deux phases produit de meilleurs résultats que de sauter directement au code.

Déléguer, ne pas dicter

Pensez à déléguer à un collègue capable. Donnez le contexte et la direction, puis faites confiance à Claude pour déterminer les détails :

Le flux de paiement est cassé pour les utilisateurs avec des cartes expirées.
Le code pertinent est dans `src/payments/`. Pouvez-vous enquêter et corriger ?

Vous n'avez pas besoin de spécifier quels fichiers lire ou quelles commandes exécuter. Claude le détermine.

Prochaines étapes

- [Étendre avec des fonctionnalités](#) Ajouter des Skills, des connexions MCP, et des commandes personnalisées
- [Workflows courants](#) Guides pas à pas pour les tâches typiques

Part 2: Core Usage

Étendre Claude Code

Comprenez quand utiliser [CLAUDE.md](#), [Skills](#), [subagents](#), [hooks](#), [MCP](#) et [plugins](#).

Claude Code combine un modèle qui raisonne sur votre code avec des [outils intégrés](#) pour les opérations sur fichiers, la recherche, l'exécution et l'accès web. Les outils intégrés couvrent la plupart des tâches de codage. Ce guide couvre la couche d'extension : les fonctionnalités que vous ajoutez pour personnaliser ce que Claude connaît, le connecter à des services externes et automatiser les flux de travail.

Note:

Pour savoir comment fonctionne la boucle agentive principale, consultez [Comment fonctionne Claude Code](#).

Nouveau dans Claude Code ? Commencez par [CLAUDE.md](#) pour les conventions de projet. Ajoutez d'autres extensions au fur et à mesure de vos besoins.

Aperçu

Les extensions se connectent à différentes parties de la boucle agentive :

- [CLAUDE.md](#) ajoute un contexte persistant que Claude voit à chaque session
- [Skills](#) ajoutent des connaissances réutilisables et des flux de travail invocables
- [MCP](#) connecte Claude à des services et outils externes
- [Subagents](#) exécutent leurs propres boucles dans un contexte isolé, en retournant des résumés
- [Agent teams](#) coordonnent plusieurs sessions indépendantes avec des tâches partagées et une messagerie pair à pair
- [Hooks](#) s'exécutent en dehors de la boucle entièrement en tant que scripts déterministes
- [Plugins](#) et [marketplaces](#) empaquettent et distribuent ces fonctionnalités

[Skills](#) sont l'extension la plus flexible. Une skill est un fichier markdown contenant des connaissances, des flux de travail ou des instructions. Vous pouvez invoquer des skills avec une commande comme `/deploy`, ou Claude peut les charger automatiquement quand elles sont pertinentes. Les skills peuvent s'exécuter dans votre conversation actuelle ou dans un contexte isolé via des subagents.

Associer les fonctionnalités à votre objectif

Les fonctionnalités vont du contexte toujours actif que Claude voit à chaque session, aux capacités à la demande que vous ou Claude pouvez invoquer, à l'automatisation en arrière-plan qui s'exécute sur des événements spécifiques. Le tableau ci-dessous montre ce qui est disponible et quand chaque option a du sens.

Fonctionnalité	Ce qu'elle fait	Quand l'utiliser	Exemple
CLAUDE.md	Contexte persistant chargé à chaque conversation	Conventions de projet, règles « toujours faire X »	« Utilisez npm, pas pnpm. Exécutez les tests avant de valider. »
Skill	Instructions, connaissances et flux de travail que Claude peut utiliser	Contenu réutilisable, documents de référence, tâches répétables	<code>/deploy</code> exécute votre liste de contrôle de déploiement ; skill de documentation API avec modèles de points de terminaison
Subagent	Contexte d'exécution isolé qui retourne des résultats résumés	Isolation du contexte, tâches parallèles, travailleurs spécialisés	Tâche de recherche qui lit de nombreux fichiers mais retourne uniquement les conclusions clés
Agent teams	Coordonnez plusieurs sessions Claude Code indépendantes	Recherche parallèle, développement de nouvelles fonctionnalités, débogage avec hypothèses concurrentes	Générez des relecteurs pour vérifier la sécurité, les performances et les tests simultanément

Fonction- nalité	Ce qu'elle fait	Quand l'utiliser	Exemple
MCP	Connectez-vous à des services externes	Données ou actions externes	Interrogez votre base de données, publiez sur Slack, contrôlez un navigateur
Hook	Script déterministe qui s'exécute sur des événements	Automatisation prévisible, aucun LLM impliqué	Exécutez ESLint après chaque modification de fichier

[Plugins](#) sont la couche d'empaquetage. Un plugin regroupe des skills, des hooks, des subagents et des serveurs MCP dans une seule unité installable. Les skills de plugin sont espacés de noms (comme `/my-plugin:review`) afin que plusieurs plugins puissent coexister. Utilisez les plugins quand vous voulez réutiliser la même configuration sur plusieurs référentiels ou distribuer à d'autres via une [marketplace](#).

Comparer les fonctionnalités similaires

Certaines fonctionnalités peuvent sembler similaires. Voici comment les distinguer.

Skill vs Subagent

Les skills et les subagents résolvent des problèmes différents :

- **Skills** sont du contenu réutilisable que vous pouvez charger dans n'importe quel contexte
- **Subagents** sont des travailleurs isolés qui s'exécutent séparément de votre conversation principale

Aspect	Skill	Subagent
Ce que c'est	Instructions, connaissances ou flux de travail réutilisables	Travailleur isolé avec son propre contexte
Avantage clé	Partagez le contenu entre les contextes	Isolation du contexte. Le travail se fait séparément, seul le résumé revient
Meilleur pour	Matériel de référence, flux de travail invocables	Tâches qui lisent de nombreux fichiers, travail parallèle, travailleurs spécialisés

Les skills peuvent être de référence ou d'action. Les skills de référence fournissent des connaissances que Claude utilise tout au long de votre session (comme votre guide de style API). Les skills d'action disent à Claude de faire quelque chose de spécifique (comme `/deploy` qui exécute votre flux de travail de déploiement).

Utilisez un subagent quand vous avez besoin d'isolation du contexte ou quand votre fenêtre de contexte se remplit. Le subagent pourrait lire des dizaines de fichiers ou exécuter des recherches étendues, mais votre conversation principale ne reçoit qu'un résumé. Puisque le travail du subagent ne consomme pas votre contexte principal, c'est aussi utile quand vous n'avez pas besoin que le travail intermédiaire reste visible. Les subagents personnalisés peuvent avoir leurs propres instructions et peuvent précharger des skills.

Ils peuvent se combiner. Un subagent peut précharger des skills spécifiques (champ `skills:`). Une skill peut s'exécuter dans un contexte isolé en utilisant `context: fork` . Consultez [Skills](#) pour plus de détails.

CLAUDE.md vs Skill

Les deux stockent des instructions, mais elles se chargent différemment et servent des objectifs différents.

Aspect	CLAUDE.md	Skill
Se charge	À chaque session, automatiquement	À la demande
Peut inclure des fichiers	Oui, avec les importations <code>@path</code>	Oui, avec les importations <code>@path</code>
Peut déclencher des flux de travail	Non	Oui, avec <code>/<code><name></code></code>
Meilleur pour	Règles « toujours faire X »	Matériel de référence, flux de travail invocables

Mettez-le dans CLAUDE.md si Claude devrait toujours le savoir : conventions de codage, commandes de construction, structure du projet, règles « ne jamais faire X ».

Mettez-le dans une skill si c'est du matériel de référence dont Claude a besoin parfois (documentation API, guides de style) ou un flux de travail que vous déclenchez avec `/<name>` (déployer, examiner, publier).

Règle générale : Gardez CLAUDE.md sous 200 lignes. S'il grandit, déplacez le contenu de référence vers des skills ou divisez-le en fichiers `.claude/rules/` .

CLAUDE.md vs Rules vs Skills

Les trois stockent des instructions, mais elles se chargent différemment :

Aspect	CLAUDE.md	.claude/rules/	Skill
Se charge	À chaque session	À chaque session, ou quand les fichiers correspondants sont ouverts	À la demande, quand invoqué ou pertinent
Portée	Projet entier	Peut être limité aux chemins de fichiers	Spécifique à la tâche
Meilleur pour	Conventions principales et commandes de construction	Directives spécifiques au langage ou au répertoire	Matériel de référence, flux de travail répertables

Utilisez **CLAUDE.md** pour les instructions que chaque session a besoin : commandes de construction, conventions de test, architecture du projet.

Utilisez les **règles** pour garder CLAUDE.md concentré. Les règles avec [frontmatter paths](#) ne se chargent que quand Claude travaille avec des fichiers correspondants, économisant du contexte.

Utilisez les **skills** pour le contenu dont Claude n'a besoin que parfois, comme la documentation API ou une liste de contrôle de déploiement que vous déclenchez avec `/ <name>`.

Subagent vs Agent team

Les deux parallélisent le travail, mais ils sont architecturalement différents :

- **Subagents** s'exécutent dans votre session et rapportent les résultats à votre contexte principal
- **Agent teams** sont des sessions Claude Code indépendantes qui communiquent les unes avec les autres

Aspect	Subagent	Agent team
Contexte	Fenêtre de contexte propre ; les résultats reviennent à l'appelant	Fenêtre de contexte propre ; complètement indépendant
Communication	Rapporte les résultats à l'agent principal uniquement	Les coéquipiers se mesagent directement

Aspect	Subagent	Agent team
Coordination	L'agent principal gère tout le travail	Liste de tâches partagée avec auto-coordination
Meilleur pour	Tâches ciblées où seul le résultat compte	Travail complexe nécessitant discussion et collaboration
Coût en tokens	Inférieur : les résultats sont résumés au contexte principal	Supérieur : chaque coéquipier est une instance Claude distincte

Utilisez un subagent quand vous avez besoin d'un travailleur rapide et ciblé : rechercher une question, vérifier une affirmation, examiner un fichier. Le subagent fait le travail et retourne un résumé. Votre conversation principale reste propre.

Utilisez une agent team quand les coéquipiers ont besoin de partager les conclusions, de se remettre en question et de se coordonner indépendamment. Les agent teams sont meilleures pour la recherche avec des hypothèses concurrentes, la relecture de code parallèle et le développement de nouvelles fonctionnalités où chaque coéquipier possède une pièce distincte.

Point de transition : Si vous exécutez des subagents parallèles mais atteignez les limites du contexte, ou si vos subagents ont besoin de communiquer les uns avec les autres, les agent teams sont l'étape naturelle suivante.

Note:

Les agent teams sont expérimentales et désactivées par défaut. Consultez [agent teams](#) pour la configuration et les limitations actuelles.

MCP vs Skill

MCP connecte Claude à des services externes. Les skills étendent ce que Claude connaît, y compris comment utiliser efficacement ces services.

Aspect	MCP	Skill
Ce que c'est	Protocole de connexion à des services externes	Connaissances, flux de travail et matériel de référence

Aspect	MCP	Skill
Fournit	Accès aux outils et aux données	Connaissances, flux de travail, matériel de référence
Exemples	Intégration Slack, requêtes de base de données, contrôle de navigateur	Liste de contrôle de relecture de code, flux de travail de déploiement, guide de style API

Ces solutions résolvent des problèmes différents et fonctionnent bien ensemble :

MCP donne à Claude la capacité d'interagir avec les systèmes externes. Sans MCP, Claude ne peut pas interroger votre base de données ou publier sur Slack.

Skills donnent à Claude des connaissances sur la façon d'utiliser efficacement ces outils, plus des flux de travail que vous pouvez déclencher avec `/<name>`. Une skill pourrait inclure le schéma de votre base de données et les modèles de requête, ou un flux de travail `/post-to-slack` avec les règles de formatage des messages de votre équipe.

Exemple : Un serveur MCP connecte Claude à votre base de données. Une skill enseigne à Claude votre modèle de données, les modèles de requête courants et les tables à utiliser pour différentes tâches.

Comprendre comment les fonctionnalités se superposent

Les fonctionnalités peuvent être définies à plusieurs niveaux : à l'échelle de l'utilisateur, par projet, via des plugins ou via des politiques gérées. Vous pouvez également imbriquer des fichiers CLAUDE.md dans des sous-répertoires ou placer des skills dans des packages spécifiques d'un monorepo. Quand la même fonctionnalité existe à plusieurs niveaux, voici comment elles se superposent :

- **Les fichiers CLAUDE.md** sont additifs : tous les niveaux contribuent du contenu au contexte de Claude simultanément. Les fichiers de votre répertoire de travail et au-dessus se chargent au lancement ; les sous-répertoires se chargent au fur et à mesure que vous y travaillez. Quand les instructions entrent en conflit, Claude utilise son jugement pour les réconcilier, les instructions plus spécifiques ayant généralement la priorité. Consultez [comment les fichiers CLAUDE.md se chargent](#).
- **Les skills et subagents** se remplacent par nom : quand le même nom existe à plusieurs niveaux, une définition gagne en fonction de la priorité (géré > utilisateur > projet pour les skills ; géré > drapeau CLI > projet > utilisateur > plugin pour les subagents). Les skills de plugin sont [espacées de noms](#) pour éviter les conflits. Consultez [découverte de skills](#) et [portée du subagent](#).

- **Les serveurs MCP** se remplacent par nom : local > projet > utilisateur. Consultez [portée MCP](#).
- **Les hooks** fusionnent : tous les hooks enregistrés se déclenchent pour leurs événements correspondants indépendamment de la source. Consultez [hooks](#).

Combiner les fonctionnalités

Chaque extension résout un problème différent : CLAUDE.md gère le contexte toujours actif, les skills gèrent les connaissances et les flux de travail à la demande, MCP gère les connexions externes, les subagents gèrent l'isolation et les hooks gèrent l'automatisation. Les configurations réelles les combinent en fonction de votre flux de travail.

Par exemple, vous pourriez utiliser CLAUDE.md pour les conventions de projet, une skill pour votre flux de travail de déploiement, MCP pour vous connecter à votre base de données et un hook pour exécuter le linting après chaque modification. Chaque fonctionnalité gère ce pour quoi elle est la meilleure.

Modèle	Comment ça fonctionne	Exemple
Skill + MCP	MCP fournit la connexion ; une skill enseigne à Claude comment l'utiliser bien	MCP se connecte à votre base de données, une skill documente votre schéma et les modèles de requête
Skill + Subagent	Une skill génère des subagents pour le travail parallèle	La skill <code>/audit</code> lance des subagents de sécurité, de performance et de style qui travaillent dans un contexte isolé
CLAUDE.md + Skills	CLAUDE.md contient les règles toujours actives ; les skills contiennent le matériel de référence chargé à la demande	CLAUDE.md dit ' suivez nos conventions API ', une skill contient le guide de style API complet
Hook + MCP	Un hook déclenche des actions externes via MCP	Le hook post-édition envoie une notification Slack quand Claude modifie des fichiers critiques

Comprendre les coûts du contexte

Chaque fonctionnalité que vous ajoutez consomme une partie du contexte de Claude. Trop peut remplir votre fenêtre de contexte, mais cela peut aussi ajouter du bruit qui rend Claude moins efficace ; les skills peuvent ne pas se déclencher correctement, ou Claude peut perdre de vue vos conventions. Comprendre ces compromis vous aide à construire une configuration efficace.

Coût du contexte par fonctionnalité

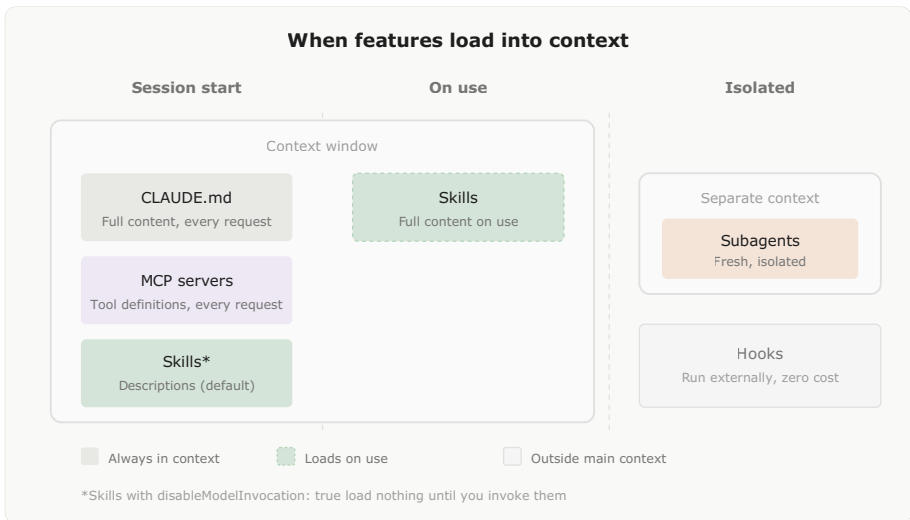
Chaque fonctionnalité a une stratégie de chargement et un coût de contexte différents :

Fonctionnalité	Quand elle se charge	Ce qui se charge	Coût du contexte
CLAUDE.md	Début de session	Contenu complet	À chaque requête
Skills	Début de session + quand utilisé	Descriptions au démarrage, contenu complet quand utilisé	Faible (descriptions à chaque requête)*
Serveurs MCP	Début de session	Toutes les définitions d'outils et schémas	À chaque requête
Sub-agents	Quand généré	Contexte frais avec les skills spécifiées	Isolé de la session principale
Hooks	Au déclenchement	Rien (s'exécute en externe)	Zéro, sauf si le hook retourne du contexte supplémentaire

*Par défaut, les descriptions de skills se chargent au début de la session afin que Claude puisse décider quand les utiliser. Définissez `disable-model-invocation: true` dans le frontmatter d'une skill pour la masquer complètement à Claude jusqu'à ce que vous l'invoquiez manuellement. Cela réduit le coût du contexte à zéro pour les skills que vous ne déclenchez que vous-même.

Comprendre comment les fonctionnalités se chargent

Chaque fonctionnalité se charge à différents points de votre session. Les onglets ci-dessous expliquent quand chacune se charge et ce qui entre dans le contexte.



Chargement du contexte : CLAUDE.md et MCP se chargent au début de la session et restent dans chaque requête. Les skills chargent les descriptions au démarrage, le contenu complet à l'invocation. Les subagents obtiennent un contexte isolé. Les hooks s'exécutent en externe.

CLAUDE.md

Quand : Début de session

Ce qui se charge : Contenu complet de tous les fichiers CLAUDE.md (niveaux géré, utilisateur et projet).

Héritage : Claude lit les fichiers CLAUDE.md de votre répertoire de travail jusqu'à la racine et découvre les fichiers imbriqués dans les sous-répertoires au fur et à mesure qu'il accède à ces fichiers. Consultez [Comment les fichiers CLAUDE.md se chargent](#) pour plus de détails.

Tip:

Gardez CLAUDE.md sous environ 500 lignes. Déplacez le matériel de référence vers les skills, qui se chargent à la demande.

Skills

Les skills sont des capacités supplémentaires dans la boîte à outils de Claude. Elles peuvent être du matériel de référence (comme un guide de style API) ou des flux de travail invocables que vous déclenchez avec `/<name>` (comme `/deploy`). Claude Code est livré

avec des [skills groupées](#) comme `/simplify`, `/batch` et `/debug` qui fonctionnent directement. Vous pouvez également créer les vôtres. Claude utilise les skills quand approprié, ou vous pouvez en invoquer une directement.

Quand : Dépend de la configuration de la skill. Par défaut, les descriptions se chargent au début de la session et le contenu complet se charge quand utilisé. Pour les skills utilisateur uniquement (`disable-model-invocation: true`), rien ne se charge jusqu'à ce que vous les invoquiez.

Ce qui se charge : Pour les skills invocables par modèle, Claude voit les noms et descriptions dans chaque requête. Quand vous invoquez une skill avec `/<name>` ou que Claude la charge automatiquement, le contenu complet se charge dans votre conversation.

Comment Claude choisit les skills : Claude associe votre tâche aux descriptions de skills pour décider lesquelles sont pertinentes. Si les descriptions sont vagues ou se chevauchent, Claude peut charger la mauvaise skill ou en manquer une qui aiderait. Pour dire à Claude d'utiliser une skill spécifique, invoquez-la avec `/<name>`. Les skills avec `disable-model-invocation: true` sont invisibles à Claude jusqu'à ce que vous les invoquiez.

Coût du contexte : Faible jusqu'à utilisation. Les skills utilisateur uniquement ont un coût zéro jusqu'à invocation.

Dans les subagents : Les skills fonctionnent différemment dans les subagents. Au lieu du chargement à la demande, les skills passées à un subagent sont entièrement préchargées dans son contexte au lancement. Les subagents n'héritent pas des skills de la session principale ; vous devez les spécifier explicitement.

Tip:

Utilisez `disable-model-invocation: true` pour les skills avec des effets secondaires. Cela économise du contexte et garantit que seul vous les déclenchez.

Serveurs MCP

Quand : Début de session.

Ce qui se charge : Toutes les définitions d'outils et schémas JSON des serveurs connectés.

Coût du contexte : [Recherche d'outils](#) (activée par défaut) charge les outils MCP jusqu'à 10 % du contexte et reporte le reste jusqu'à ce qu'il soit nécessaire.

Note de fiabilité : Les connexions MCP peuvent échouer silencieusement en milieu de session. Si un serveur se déconnecte, ses outils disparaissent sans avertissement. Claude peut essayer d'utiliser un outil qui n'existe plus. Si vous remarquez que Claude ne peut pas utiliser un outil MCP auquel il pouvait accéder précédemment, vérifiez la connexion avec `/mcp`.

Tip:

Exécutez `/mcp` pour voir les coûts en tokens par serveur. Déconnectez les serveurs que vous n'utilisez pas activement.

Subagents

Quand : À la demande, quand vous ou Claude en générez un pour une tâche.

Ce qui se charge : Contexte frais et isolé contenant :

- L'invite système (partagée avec le parent pour l'efficacité du cache)
- Contenu complet des skills listées dans le champ `skills:` de l'agent
- CLAUDE.md et statut git (hérité du parent)
- Quel que soit le contexte que l'agent principal transmet dans l'invite

Coût du contexte : Isolé de la session principale. Les subagents n'héritent pas de votre historique de conversation ou des skills invoquées.

Tip:

Utilisez les subagents pour le travail qui n'a pas besoin de votre contexte de conversation complet. Leur isolation empêche de gonfler votre session principale.

Hooks

Quand : Au déclenchement. Les hooks se déclenchent à des événements de cycle de vie spécifiques comme l'exécution d'outils, les limites de session, la soumission d'invite, les demandes de permission et la compaction. Consultez [Hooks](#) pour la liste complète.

Ce qui se charge : Rien par défaut. Les hooks s'exécutent en tant que scripts externes.

Coût du contexte : Zéro, sauf si le hook retourne une sortie qui est ajoutée en tant que messages à votre conversation.

Tip:

Les hooks sont idéaux pour les effets secondaires (linting, journalisation) qui n'ont pas besoin d'affecter le contexte de Claude.

En savoir plus

Chaque fonctionnalité a son propre guide avec des instructions de configuration, des exemples et des options de configuration.

- [CLAUDE.md](#) Stockez le contexte du projet, les conventions et les instructions
- [Skills](#) Donnez à Claude une expertise de domaine et des flux de travail réutilisables
- [Subagents](#) Déléguez le travail à un contexte isolé
- [Agent teams](#) Coordonnez plusieurs sessions travaillant en parallèle
- [MCP](#) Connectez Claude à des services externes
- [Hooks](#) Automatisez les flux de travail avec des hooks
- [Plugins](#) Empaquetez et partagez des ensembles de fonctionnalités
- [Marketplaces](#) Hébergez et distribuez des collections de plugins

Mode interactif

Référence complète des raccourcis clavier, modes d'entrée et fonctionnalités interactives dans les sessions Claude Code.

Raccourcis clavier

Note:

Les raccourcis clavier peuvent varier selon la plateforme et le terminal. Appuyez sur **?** pour voir les raccourcis disponibles dans votre environnement.

Utilisateurs macOS : les raccourcis avec la touche Option/Alt (**Alt+B** , **Alt+F** , **Alt+Y** , **Alt+M** , **Alt+P**) nécessitent de configurer Option comme Meta dans votre terminal :

- **iTerm2** : paramètres → Profils → Touches → définir la touche Option gauche/droite sur « Esc+ »
- **Terminal.app** : paramètres → Profils → Clavier → cocher « Utiliser Option comme touche Meta »
- **VS Code** : paramètres → Profils → Touches → définir la touche Option gauche/droite sur « Esc+ »

Consultez [Configuration du terminal](#) pour plus de détails.

Contrôles généraux

Raccourci	Description	Contexte
Ctrl+C	Annuler l'entrée ou la génération actuelle	Interruption standard
Ctrl+F	Arrêter tous les agents en arrière-plan. Appuyez deux fois dans les 3 secondes pour confirmer	Contrôle des agents en arrière-plan
Ctrl+D	Quitter la session Claude Code	Signal EOF
Ctrl+G	Ouvrir dans l'éditeur de texte par défaut	Modifiez votre invite ou réponse personnalisée dans votre éditeur de texte par défaut

Raccourci	Description	Contexte
Ctrl+L	Effacer l'écran du terminal	Conserve l'historique de conversation
Ctrl+O	Basculer la sortie détaillée	Affiche l'utilisation détaillée des outils et l'exécution
Ctrl+R	Recherche inversée dans l'historique des commandes	Rechercher dans les commandes précédentes de manière interactive
Ctrl+V ou Cmd+V (iTerm2) ou Alt+V (Windows)	Coller une image du presse-papiers	Colle une image ou un chemin vers un fichier image
Ctrl+B	Tâches en arrière-plan	Met en arrière-plan les commandes bash et les agents. Les utilisateurs Tmux appuiez deux fois
Ctrl+T	Basculer la liste des tâches	Afficher ou masquer la liste des tâches dans la zone d'état du terminal
Flèches gauche/droite	Parcourir les onglets de dialogue	Naviguer entre les onglets dans les dialogues de permission et les menus
Flèches haut/bas	Naviguer dans l'historique des commandes	Rappeler les entrées précédentes
Esc + Esc	Rembobiner ou résumer	Restaurer le code et/ou la conversation à un point antérieur, ou résumer à partir d'un message sélectionné
Shift+Tab ou Alt+M (certaines configurations)	Basculer les modes de permission	Basculer entre le mode Auto-Accept, Plan Mode et le mode normal.
Option+P (macOS) ou Alt+P (Windows/Linux)	Changer de modèle	Changer de modèles sans effacer votre invite

Raccourci	Description	Contexte
<code>Option+T</code> (macOS) ou <code>Alt+T</code> (Windows/Linux)	Basculer la réflexion étendue	Activer ou désactiver le mode de réflexion étendue. Exécutez d’abord <code>/terminal-setup</code> pour activer ce raccourci

Édition de texte

Raccourci	Description	Contexte
<code>Ctrl+K</code>	Supprimer jusqu’à la fin de la ligne	Stocke le texte supprimé pour le collage
<code>Ctrl+U</code>	Supprimer la ligne entière	Stocke le texte supprimé pour le collage
<code>Ctrl+Y</code>	Coller le texte supprimé	Coller le texte supprimé avec <code>Ctrl+K</code> ou <code>Ctrl+U</code>
<code>Alt+Y</code> (après <code>Ctrl+Y</code>)	Parcourir l’historique du collage	Après le collage, parcourir le texte précédemment supprimé. Nécessite Option comme Meta sur macOS
<code>Alt+B</code>	Déplacer le curseur d’un mot en arrière	Navigation par mot. Nécessite Option comme Meta sur macOS
<code>Alt+F</code>	Déplacer le curseur d’un mot en avant	Navigation par mot. Nécessite Option comme Meta sur macOS

Thème et affichage

Raccourci	Description	Contexte
<code>Ctrl+T</code>	Basculer la coloration syntaxique pour les blocs de code	Fonctionne uniquement dans le menu du sélecteur <code>/theme</code> . Contrôle si le code dans les réponses de Claude utilise la coloration syntaxique

Note:

La coloration syntaxique n'est disponible que dans la version native de Claude Code.

Entrée multiligne

Méthode	Raccourci	Contexte
Échappement rapide	<code>\ + Entrée</code>	Fonctionne dans tous les terminaux
Par défaut macOS	<code>Option+Entrée</code>	Par défaut sur macOS
Shift+Entrée	<code>Shift+Entrée</code>	Fonctionne directement dans iTerm2, WezTerm, Ghostty, Kitty
Séquence de contrôle	<code>Ctrl+J</code>	Caractère de saut de ligne pour multiligne
Mode collage	Coller directement	Pour les blocs de code, les journaux

Tip:

Shift+Entrée fonctionne sans configuration dans iTerm2, WezTerm, Ghostty et Kitty. Pour les autres terminaux (VS Code, Alacritty, Zed, Warp), exécutez `/terminal-setup` pour installer la liaison.

Commandes rapides

Raccourci	Description	Notes
<code>/</code> au début	Commande ou skill	Consultez commandes intégrées et skills
<code>!</code> au début	Mode Bash	Exécuter les commandes directement et ajouter la sortie d'exécution à la session
<code>@</code>	Mention de chemin de fichier	Déclencher l'autocomplétion du chemin de fichier

Commandes intégrées

Tapez `/` dans Claude Code pour voir toutes les commandes disponibles, ou tapez `/` suivi de n'importe quelles lettres pour filtrer. Toutes les commandes ne sont pas visibles pour tous les utilisateurs. Certaines dépendent de votre plateforme, plan ou environnement. Par exemple, `/desktop` n'apparaît que sur macOS et Windows, `/upgrade` et `/privacy-settings` ne sont disponibles que pour les plans Pro et Max, et `/terminal-setup` est masqué lorsque votre terminal supporte nativement ses liaisons de touches.

Claude Code est également livré avec des [skills groupés](#) comme `/simplify`, `/batch` et `/debug` qui apparaissent aux côtés des commandes intégrées lorsque vous tapez `/`. Pour créer vos propres commandes, consultez [skills](#).

Dans le tableau ci-dessous, `<arg>` indique un argument obligatoire et `[arg]` indique un argument facultatif.

Commande	Objectif
<code>/add-dir</code> <code><path></code>	Ajouter un nouveau répertoire de travail à la session actuelle
<code>/agents</code>	Gérer les configurations d' agent
<code>/btw</code> <code><question></code>	Poser une question rapide sans l'ajouter à la conversation
<code>/chrome</code>	Configurer les paramètres de Claude dans Chrome
<code>/clear</code>	Effacer l'historique de conversation et libérer du contexte. Alias : <code>/reset</code> , <code>/new</code>
<code>/compact</code> <code>[instructions]</code>	Compacter la conversation avec des instructions de focus optionnelles
<code>/config</code>	Ouvrir l'interface Paramètres pour ajuster le thème, le modèle, style de sortie et autres préférences. Alias : <code>/settings</code>
<code>/context</code>	Visualiser l'utilisation actuelle du contexte sous forme de grille colorée
<code>/copy</code>	Copier la dernière réponse de l'assistant dans le presse-papiers. Lorsque des blocs de code sont présents, affiche un sélecteur interactif pour sélectionner des blocs individuels ou la réponse complète
<code>/cost</code>	Afficher les statistiques d'utilisation des tokens. Consultez le guide de suivi des coûts pour les détails spécifiques à l'abonnement

Commande	Objectif
<code>/desktop</code>	Continuer la session actuelle dans l'application Claude Code Desktop. macOS et Windows uniquement. Alias : <code>/app</code>
<code>/diff</code>	Ouvrir une visionneuse de diff interactive affichant les modifications non validées et les diffs par tour. Utilisez les flèches gauche/droite pour basculer entre le diff git actuel et les tours Claude individuels, et haut/bas pour parcourir les fichiers
<code>/doctor</code>	Diagnostiquer et vérifier votre installation et vos paramètres Claude Code
<code>/exit</code>	Quitter le CLI. Alias : <code>/quit</code>
<code>/export [filename]</code>	Exporter la conversation actuelle en texte brut. Avec un nom de fichier, écrit directement dans ce fichier. Sans, ouvre un dialogue pour copier dans le presse-papiers ou enregistrer dans un fichier
<code>/extra-usage</code>	Configurer l'utilisation supplémentaire pour continuer à travailler lorsque les limites de débit sont atteintes
<code>/fast [on off]</code>	Basculer le mode rapide activé ou désactivé
<code>/feedback [report]</code>	Soumettre des commentaires sur Claude Code. Alias : <code>/bug</code>
<code>/fork [name]</code>	Créer une fourche de la conversation actuelle à ce stade
<code>/help</code>	Afficher l'aide et les commandes disponibles
<code>/hooks</code>	Gérer les configurations de hook pour les événements d'outils
<code>/ide</code>	Gérer les intégrations IDE et afficher l'état
<code>/init</code>	Initialiser le projet avec le guide CLAUDE.md
<code>/insights</code>	Générer un rapport analysant vos sessions Claude Code, y compris les domaines de projet, les modèles d'interaction et les points de friction
<code>/install-github-app</code>	Configurer l'application Claude GitHub Actions pour un référentiel. Vous guide dans la sélection d'un référentiel et la configuration de l'intégration
<code>/install-slack-app</code>	Installer l'application Claude Slack. Ouvre un navigateur pour terminer le flux OAuth
<code>/keybindings</code>	Ouvrir ou créer votre fichier de configuration des liaisons de touches

Commande	Objectif
<code>/login</code>	Se connecter à votre compte Anthropic
<code>/logout</code>	Se déconnecter de votre compte Anthropic
<code>/mcp</code>	Gérer les connexions de serveur MCP et l'authentification OAuth
<code>/memory</code>	Modifier les fichiers de mémoire <code>CLAUDE.md</code> , activer ou désactiver la mémoire automatique et afficher les entrées de mémoire automatique
<code>/mobile</code>	Afficher le code QR pour télécharger l'application mobile Claude. Alias : <code>/ios</code> , <code>/android</code>
<code>/model [model]</code>	Sélectionner ou changer le modèle IA. Pour les modèles qui le supportent, utilisez les flèches gauche/droite pour ajuster le niveau d'effort . Le changement prend effet immédiatement sans attendre la fin de la réponse actuelle
<code>/passes</code>	Partager une semaine gratuite de Claude Code avec des amis. Visible uniquement si votre compte est éligible
<code>/permissions</code>	Afficher ou mettre à jour les permissions . Alias : <code>/allowed-tools</code>
<code>/plan</code>	Entrer directement en mode plan à partir de l'invite
<code>/plugin</code>	Gérer les plugins Claude Code
<code>/pr-comments [PR]</code>	Récupérer et afficher les commentaires d'une demande de tirage GitHub. Détecte automatiquement la PR pour la branche actuelle, ou passez une URL ou un numéro de PR. Nécessite le CLI <code>gh</code>
<code>/privacy-settings</code>	Afficher et mettre à jour vos paramètres de confidentialité. Disponible uniquement pour les abonnés aux plans Pro et Max
<code>/release-notes</code>	Afficher le journal des modifications complet, avec la version la plus récente la plus proche de votre invite
<code>/reload-plugins</code>	Recharger tous les plugins actifs pour appliquer les modifications en attente sans redémarrer. Signale ce qui a été chargé et note les modifications qui nécessitent un redémarrage
<code>/remote-control</code>	Rendre cette session disponible pour le contrôle à distance depuis claude.ai. Alias : <code>/rc</code>
<code>/remote-env</code>	Configurer l'environnement distant par défaut pour les sessions de téléportation

Commande	Objectif
<code>/rename [name]</code>	Renommer la session actuelle. Sans nom, en génère automatiquement un à partir de l'historique de conversation
<code>/resume [session]</code>	Reprendre une conversation par ID ou nom, ou ouvrir le sélecteur de session. Alias : <code>/continue</code>
<code>/review</code>	Déprécié. Installez plutôt le plugin code-review : <code>claude plugin install code-review@claude-code-marketplace</code>
<code>/rewind</code>	Rembobiner la conversation et/ou le code à un point antérieur, ou résumer à partir d'un message sélectionné. Consultez checkpointing . Alias : <code>/checkpoint</code>
<code>/sandbox</code>	Basculer le mode sandbox . Disponible uniquement sur les plateformes supportées
<code>/security-review</code>	Analyser les modifications en attente sur la branche actuelle pour les vulnérabilités de sécurité. Examine le diff git et identifie les risques comme l'injection, les problèmes d'authentification et l'exposition de données
<code>/skills</code>	Lister les skills disponibles
<code>/stats</code>	Visualiser l'utilisation quotidienne, l'historique des sessions, les séries et les préférences de modèle
<code>/status</code>	Ouvrir l'interface Paramètres (onglet Statut) affichant la version, le modèle, le compte et la connectivité
<code>/statusline</code>	Configurer la ligne d'état de Claude Code. Décrivez ce que vous voulez, ou exécutez sans arguments pour auto-configurer à partir de votre invite shell
<code>/stickers</code>	Commander des autocollants Claude Code
<code>/tasks</code>	Lister et gérer les tâches en arrière-plan
<code>/terminal-setup</code>	Configurer les liaisons de touches du terminal pour Shift+Entrée et d'autres raccourcis. Visible uniquement dans les terminaux qui en ont besoin, comme VS Code, Alacritty ou Warp
<code>/theme</code>	Changer le thème de couleur. Inclut les variantes claires et sombres, les thèmes accessibles aux daltoniens (daltonisés) et les thèmes ANSI qui utilisent la palette de couleurs de votre terminal
<code>/upgrade</code>	Ouvrir la page de mise à niveau pour passer à un niveau de plan supérieur

Commande	Objectif
<code>/usage</code>	Afficher les limites d'utilisation du plan et l'état de la limite de débit
<code>/vim</code>	Basculer entre les modes d'édition Vim et Normal

Invites MCP

Les serveurs MCP peuvent exposer des invites qui apparaissent comme des commandes. Celles-ci utilisent le format `/mcp__<server>__<prompt>` et sont découvertes dynamiquement à partir des serveurs connectés. Consultez [Invites MCP](#) pour plus de détails.

Mode éditeur Vim

Activer l'édition de style vim avec la commande `/vim` ou configurer de manière permanente via `/config`.

Basculement de mode

Commande	Action	Du mode
<code>Esc</code>	Entrer en mode NORMAL	INSERT
<code>i</code>	Insérer avant le curseur	NORMAL
<code>I</code>	Insérer au début de la ligne	NORMAL
<code>a</code>	Insérer après le curseur	NORMAL
<code>A</code>	Insérer à la fin de la ligne	NORMAL
<code>o</code>	Ouvrir une ligne en dessous	NORMAL
<code>O</code>	Ouvrir une ligne au-dessus	NORMAL

Navigation (mode NORMAL)

Commande	Action
<code>h / j / k / l</code>	Déplacer gauche/bas/haut/droite
<code>w</code>	Mot suivant
<code>e</code>	Fin du mot
<code>b</code>	Mot précédent

Commande	Action
<code>0</code>	Début de la ligne
<code>\$</code>	Fin de la ligne
<code>^</code>	Premier caractère non vide
<code>gg</code>	Début de l'entrée
<code>G</code>	Fin de l'entrée
<code>f{char}</code>	Sauter à la prochaine occurrence du caractère
<code>F{char}</code>	Sauter à l'occurrence précédente du caractère
<code>t{char}</code>	Sauter juste avant la prochaine occurrence du caractère
<code>T{char}</code>	Sauter juste après l'occurrence précédente du caractère
<code>;</code>	Répéter le dernier mouvement f/F/t/T
<code>,</code>	Répéter le dernier mouvement f/F/t/T en sens inverse

Note:

En mode normal vim, si le curseur est au début ou à la fin de l'entrée et ne peut pas se déplacer davantage, les touches fléchées naviguent dans l'historique des commandes à la place.

Édition (mode NORMAL)

Commande	Action
<code>x</code>	Supprimer le caractère
<code>dd</code>	Supprimer la ligne
<code>D</code>	Supprimer jusqu'à la fin de la ligne
<code>dw</code> / <code>de</code> / <code>db</code>	Supprimer mot/jusqu'à la fin/en arrière
<code>cc</code>	Changer la ligne
<code>C</code>	Changer jusqu'à la fin de la ligne
<code>cw</code> / <code>ce</code> / <code>cb</code>	Changer mot/jusqu'à la fin/en arrière

Commande	Action
<code>yy / Y</code>	Copier la ligne
<code>yw / ye / yb</code>	Copier mot/jusqu'à la fin/en arrière
<code>p</code>	Coller après le curseur
<code>P</code>	Coller avant le curseur
<code>>></code>	Indenter la ligne
<code><<</code>	Dédenter la ligne
<code>J</code>	Joindre les lignes
<code>.</code>	Répéter la dernière modification

Objets texte (mode NORMAL)

Les objets texte fonctionnent avec les opérateurs comme `d`, `c` et `y` :

Commande	Action
<code>iw / aw</code>	Mot intérieur/autour
<code>iW / aW</code>	MOT intérieur/autour (délimité par espace)
<code>i" / a"</code>	Guillemets doubles intérieurs/autour
<code>i' / a'</code>	Guillemets simples intérieurs/autour
<code>i(/ a(</code>	Parenthèses intérieures/autour
<code>i[/ a[</code>	Crochets intérieurs/autour
<code>i{ / a{</code>	Accolades intérieures/autour

Historique des commandes

Claude Code maintient l'historique des commandes pour la session actuelle :

- L'historique d'entrée est stocké par répertoire de travail
- L'historique d'entrée se réinitialise lorsque vous exécutez `/clear` pour démarrer une nouvelle session. La conversation de la session précédente est conservée et peut être reprise.

- Utilisez les flèches haut/bas pour naviguer (voir les raccourcis clavier ci-dessus)
- **Remarque** : l'expansion d'historique (!) est désactivée par défaut

Recherche inversée avec Ctrl+R

Appuyez sur **Ctrl+R** pour rechercher de manière interactive dans votre historique de commandes :

1. **Démarrer la recherche** : appuyez sur **Ctrl+R** pour activer la recherche d'historique inversée
2. **Tapez la requête** : entrez le texte à rechercher dans les commandes précédentes. Le terme de recherche est mis en évidence dans les résultats correspondants
3. **Naviguer dans les correspondances** : appuyez à nouveau sur **Ctrl+R** pour parcourir les correspondances plus anciennes
4. **Accepter la correspondance** :
 - Appuyez sur **Tab** ou **Esc** pour accepter la correspondance actuelle et continuer l'édition
 - Appuyez sur **Entrée** pour accepter et exécuter la commande immédiatement
5. **Annuler la recherche** :
 - Appuyez sur **Ctrl+C** pour annuler et restaurer votre entrée d'origine
 - Appuyez sur **Retour arrière** sur une recherche vide pour annuler

La recherche affiche les commandes correspondantes avec le terme de recherche mis en évidence, afin que vous puissiez trouver et réutiliser les entrées précédentes.

Commandes bash en arrière-plan

Claude Code supporte l'exécution de commandes bash en arrière-plan, vous permettant de continuer à travailler pendant que les processus de longue durée s'exécutent.

Comment fonctionne la mise en arrière-plan

Lorsque Claude Code exécute une commande en arrière-plan, il exécute la commande de manière asynchrone et retourne immédiatement un ID de tâche en arrière-plan. Claude Code peut répondre à de nouvelles invites pendant que la commande continue à s'exécuter en arrière-plan.

Pour exécuter les commandes en arrière-plan, vous pouvez soit :

- Inviter Claude Code à exécuter une commande en arrière-plan

- Appuyez sur Ctrl+B pour déplacer une invocation d'outil Bash régulière en arrière-plan. (Les utilisateurs Tmux doivent appuyer sur Ctrl+B deux fois en raison de la touche de préfixe de tmux.)

Caractéristiques clés :

- La sortie est mise en mémoire tampon et Claude peut la récupérer à l'aide de l'outil TaskOutput
- Les tâches en arrière-plan ont des ID uniques pour le suivi et la récupération de la sortie
- Les tâches en arrière-plan sont automatiquement nettoyées lorsque Claude Code se ferme

Pour désactiver toute la fonctionnalité de tâche en arrière-plan, définissez la variable d'environnement `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` sur `1`. Consultez [Variables d'environnement](#) pour plus de détails.

Commandes couramment mises en arrière-plan :

- Outils de construction (webpack, vite, make)
- Gestionnaires de paquets (npm, yarn, pnpm)
- Exécuteurs de tests (jest, pytest)
- Serveurs de développement
- Processus de longue durée (docker, terraform)

Mode Bash avec le préfixe `!`

Exécutez les commandes bash directement sans passer par Claude en préfixant votre entrée avec `!` :

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

Mode Bash :

- Ajoute la commande et sa sortie au contexte de conversation
- Affiche la progression et la sortie en temps réel
- Supporte la même mise en arrière-plan `Ctrl+B` pour les commandes de longue durée
- Ne nécessite pas que Claude interprète ou approuve la commande

- Supporte l'autocomplétion basée sur l'historique : tapez une commande partielle et appuyez sur **Tab** pour compléter à partir des commandes  précédentes du projet actuel
- Quitter avec **Échap**, **Retour arrière** ou **Ctrl+U** sur une invite vide

Ceci est utile pour les opérations shell rapides tout en maintenant le contexte de conversation.

Suggestions d'invite

Lorsque vous ouvrez d'abord une session, une commande d'exemple grisée apparaît dans l'entrée d'invite pour vous aider à démarrer. Claude Code choisit ceci à partir de l'historique git de votre projet, il reflète donc les fichiers sur lesquels vous avez travaillé récemment.

Après que Claude réponde, les suggestions continuent à apparaître en fonction de votre historique de conversation, comme une étape de suivi d'une demande en plusieurs parties ou une continuation naturelle de votre flux de travail.

- Appuyez sur **Tab** pour accepter la suggestion, ou appuyez sur **Entrée** pour accepter et soumettre
- Commencez à taper pour la rejeter

La suggestion s'exécute comme une demande en arrière-plan qui réutilise le cache d'invite de la conversation parent, le coût supplémentaire est donc minimal. Claude Code ignore la génération de suggestions lorsque le cache est froid pour éviter les coûts inutiles.

Les suggestions sont automatiquement ignorées après le premier tour d'une conversation, en mode non interactif et en mode plan.

Pour désactiver complètement les suggestions d'invite, définissez la variable d'environnement ou basculez le paramètre dans **/config** :

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

Questions rapides avec /btw

Utilisez **/btw** pour poser une question rapide sur votre travail actuel sans l'ajouter à l'historique de conversation. Ceci est utile lorsque vous voulez une réponse rapide mais ne voulez pas encombrer le contexte principal ou détourner Claude d'une tâche de longue durée.


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

Les questions rapides ont une visibilité complète sur la conversation actuelle, vous pouvez donc poser des questions sur le code que Claude a déjà lu, les décisions qu'il a prises plus tôt, ou n'importe quoi d'autre de la session. La question et la réponse sont éphémères : elles apparaissent dans une superposition rejetable et n'entrent jamais dans l'historique de conversation.

- **Disponible pendant que Claude travaille** : vous pouvez exécuter `/btw` même pendant que Claude traite une réponse. La question rapide s'exécute indépendamment et n'interrompt pas le tour principal.
- **Pas d'accès aux outils** : les questions rapides répondent uniquement à partir de ce qui est déjà en contexte. Claude ne peut pas lire les fichiers, exécuter les commandes ou rechercher lors de la réponse à une question rapide.
- **Réponse unique** : il n'y a pas de tours de suivi. Si vous avez besoin d'un aller-retour, utilisez une invite normale à la place.
- **Coût faible** : la question rapide réutilise le cache d'invite de la conversation parent, le coût supplémentaire est donc minimal.

Appuyez sur **Espace**, **Entrée** ou **Échap** pour rejeter la réponse et revenir à l'invite.

`/btw` est l'inverse d'un `subagent` : il voit votre conversation complète mais n'a pas d'outils, tandis qu'un `subagent` a tous les outils mais commence avec un contexte vide. Utilisez `/btw` pour poser des questions sur ce que Claude sait déjà de cette session ; utilisez un `subagent` pour aller découvrir quelque chose de nouveau.

Liste des tâches

Lorsque vous travaillez sur un travail complexe en plusieurs étapes, Claude crée une liste de tâches pour suivre la progression. Les tâches apparaissent dans la zone d'état de votre terminal avec des indicateurs montrant ce qui est en attente, en cours ou terminé.

- Appuyez sur `Ctrl+T` pour basculer l'affichage de la liste des tâches. L'affichage montre jusqu'à 10 tâches à la fois
- Pour voir toutes les tâches ou les effacer, demandez directement à Claude : « show me all tasks » ou « clear all tasks »
- Les tâches persistent à travers les compactions de contexte, aidant Claude à rester organisé sur les projets plus importants

- Pour partager une liste de tâches entre les sessions, définissez `CLAUDE_CODE_TASK_LIST_ID` pour utiliser un répertoire nommé dans `~/.claude/tasks/` : `CLAUDE_CODE_TASK_LIST_ID=my-project claude`
- Pour revenir à la liste TODO précédente, définissez `CLAUDE_CODE_ENABLE_TASKS=false`.

Statut de révision PR

Lorsque vous travaillez sur une branche avec une demande de tirage ouverte, Claude Code affiche un lien PR cliquable dans le pied de page (par exemple, « PR #446 »). Le lien a un soulignement coloré indiquant l'état de révision :

- Vert : approuvé
- Jaune : révision en attente
- Rouge : modifications demandées
- Gris : brouillon
- Violet : fusionné

`Cmd+clic` (Mac) ou `Ctrl+clic` (Windows/Linux) sur le lien pour ouvrir la demande de tirage dans votre navigateur. L'état se met à jour automatiquement toutes les 60 secondes.

Note:

Le statut PR nécessite que le CLI `gh` soit installé et authentifié (`gh auth login`).

Voir aussi

- [Skills](#) - Invites personnalisées et flux de travail
- [Checkpointing](#) - Rembobiner les modifications de Claude et restaurer les états précédents
- [Référence CLI](#) - Drapeaux et options de ligne de commande
- [Paramètres](#) - Options de configuration
- [Gestion de la mémoire](#) - Gestion des fichiers CLAUDE.md

Meilleures pratiques pour Claude Code

Conseils et modèles pour tirer le meilleur parti de Claude Code, de la configuration de votre environnement à la mise à l'échelle sur plusieurs sessions parallèles.

Claude Code est un environnement de codage agentique. Contrairement à un chatbot qui répond aux questions et attend, Claude Code peut lire vos fichiers, exécuter des commandes, apporter des modifications et travailler de manière autonome sur les problèmes pendant que vous regardez, redirigez ou vous éloignez complètement.

Cela change votre façon de travailler. Au lieu d'écrire du code vous-même et de demander à Claude de le réviser, vous décrivez ce que vous voulez et Claude détermine comment le construire. Claude explore, planifie et implémente.

Mais cette autonomie s'accompagne toujours d'une courbe d'apprentissage. Claude fonctionne dans certaines contraintes que vous devez comprendre.

Ce guide couvre les modèles qui se sont avérés efficaces dans les équipes internes d'Anthropic et pour les ingénieurs utilisant Claude Code sur diverses bases de code, langages et environnements. Pour comprendre comment la boucle agentique fonctionne sous le capot, consultez [Comment fonctionne Claude Code](#).

La plupart des meilleures pratiques sont basées sur une contrainte : la fenêtre de contexte de Claude se remplit rapidement et les performances se dégradent à mesure qu'elle se remplit.

La fenêtre de contexte de Claude contient l'intégralité de votre conversation, y compris chaque message, chaque fichier que Claude lit et chaque sortie de commande. Cependant, cela peut se remplir rapidement. Une seule session de débogage ou exploration de base de code peut générer et consommer des dizaines de milliers de tokens.

Cela importe car les performances des LLM se dégradent à mesure que le contexte se remplit. Lorsque la fenêtre de contexte est presque pleine, Claude peut commencer à « oublier » les instructions antérieures ou faire plus d'erreurs. La fenêtre de contexte est la ressource la plus importante à gérer. Suivez l'utilisation du contexte en continu avec une [ligne d'état personnalisée](#), et consultez [Réduire l'utilisation des tokens](#) pour des stratégies de réduction de l'utilisation des tokens.

Donnez à Claude un moyen de vérifier son travail

Tip:

Incluez des tests, des captures d'écran ou des résultats attendus pour que Claude puisse se vérifier lui-même. C'est la chose la plus importante que vous puissiez faire.

Claude fonctionne beaucoup mieux lorsqu'il peut vérifier son propre travail, comme exécuter des tests, comparer des captures d'écran et valider les résultats.

Sans critères de succès clairs, il pourrait produire quelque chose qui semble correct mais qui ne fonctionne pas réellement. Vous devenez la seule boucle de rétroaction, et chaque erreur nécessite votre attention.

Stratégie	Avant	Après
Fournir des critères de vérification	« implémenter une fonction qui valide les adresses e-mail »	« écrire une fonction validateEmail. exemples de cas de test : user@example.com est vrai, invalid est faux , user@.com est faux. exécuter les tests après l'implémentation »
Vérifier les modifications de l'interface utilisateur visuellement	« rendre le tableau de bord plus beau »	« [coller la capture d'écran] implémenter cette conception. prendre une capture d'écran du résultat et la comparer à l'original. lister les différences et les corriger »
Traiter les causes profondes, pas les symptômes	« la compilation échoue »	« la compilation échoue avec cette erreur : [coller l'erreur]. la corriger et vérifier que la compilation réussit. traiter la cause profonde, ne pas supprimer l'erreur »

Les modifications de l'interface utilisateur peuvent être vérifiées à l'aide de l'[extension Claude dans Chrome](#). Elle ouvre de nouveaux onglets dans votre navigateur, teste l'interface utilisateur et itère jusqu'à ce que le code fonctionne.

Votre vérification peut également être une suite de tests, un linter ou une commande Bash qui vérifie la sortie. Investissez pour rendre votre vérification solide.

Explorez d’abord, puis planifiez, puis codez

Tip:

Séparez la recherche et la planification de l’implémentation pour éviter de résoudre le mauvais problème.

Laisser Claude sauter directement au codage peut produire du code qui résout le mauvais problème. Utilisez [Plan Mode](#) pour séparer l’exploration de l’exécution.

Le flux de travail recommandé comporte quatre phases :

Step 1: Explorez

Entrez en Plan Mode. Claude lit les fichiers et répond aux questions sans apporter de modifications.

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

Step 2: Planifiez

Demandez à Claude de créer un plan d’implémentation détaillé.

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

Appuyez sur **Ctrl+G** pour ouvrir le plan dans votre éditeur de texte pour une édition directe avant que Claude ne procède.

Step 3: Implémentez

Revenez au Mode Normal et laissez Claude coder, en vérifiant par rapport à son plan.

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

Step 4: Validez

Demandez à Claude de valider avec un message descriptif et de créer une PR.

commit with a descriptive message and open a PR

Plan Mode est utile, mais ajoute également des frais généraux.

Pour les tâches où la portée est claire et la correction est petite (comme corriger une faute de frappe, ajouter une ligne de journal ou renommer une variable), demandez à Claude de le faire directement.

La planification est plus utile lorsque vous êtes incertain de l'approche, lorsque la modification modifie plusieurs fichiers ou lorsque vous n'êtes pas familier avec le code en cours de modification. Si vous pouviez décrire le diff en une phrase, ignorez le plan.

Fournissez un contexte spécifique dans vos invites

Tip:

Plus vos instructions sont précises, moins vous aurez besoin de corrections.

Claude peut déduire l'intention, mais il ne peut pas lire dans vos pensées. Référez des fichiers spécifiques, mentionnez les contraintes et pointez vers des modèles d'exemple.

Stratégie	Avant	Après
Délimitez la tâche. Spécifiez quel fichier, quel scénario et les préférences de test.	« ajouter des tests pour foo.py »	« écrire un test pour foo.py couvrant le cas limite où l'utilisateur est déconnecté. éviter les mocks. »
Pointez vers les sources. Dirigez Claude vers la source qui peut répondre à une question.	« pourquoi ExecutionFactory a-t-il une API aussi bizarre ? »	« parcourir l'historique git d'ExecutionFactory et résumer comment son API en est venue à être »

Stratégie	Avant	Après
Référenciez les modèles existants. Pointez Claude vers les modèles de votre base de code.	<i>« ajouter un widget calendrier »</i>	<i>« regarder comment les widgets existants sont implémentés sur la page d'accueil pour comprendre les modèles. HotDogWidget.php est un bon exemple. suivre le modèle pour implémenter un nouveau widget calendrier qui permet à l'utilisateur de sélectionner un mois et de paginer vers l'avant/l'arrière pour choisir une année. construire à partir de zéro sans bibliothèques autres que celles déjà utilisées dans la base de code. »</i>
Décrivez le symptôme. Fournissez le symptôme, l'emplacement probable et ce que « corrigé » signifie.	<i>« corriger le bug de connexion »</i>	<i>« les utilisateurs signalent que la connexion échoue après l'expiration de la session. vérifier le flux d'authentification dans src/auth/, en particulier l'actualisation des tokens. écrire un test défaillant qui reproduit le problème, puis le corriger »</i>

Les invites vagues peuvent être utiles lorsque vous explorez et que vous pouvez vous permettre de corriger la trajectoire. Une invite comme *« qu'amélioreriez-vous dans ce fichier ? »* peut révéler des choses auxquelles vous n'auriez pas pensé à demander.

Fournissez du contenu riche

Tip:

Utilisez @ pour référencer des fichiers, coller des captures d'écran/images ou canaliser les données directement.

Vous pouvez fournir des données riches à Claude de plusieurs façons :

- **Référenciez les fichiers avec @** au lieu de décrire où le code se trouve. Claude lit le fichier avant de répondre.
- **Collez les images directement.** Copiez/collez ou glissez-déposez les images dans l'invite.
- **Donnez des URL** pour la documentation et les références API. Utilisez `/permissions` pour autoriser les domaines fréquemment utilisés.
- **Canalisez les données** en exécutant `cat error.log | claude` pour envoyer le contenu du fichier directement.
- **Laissez Claude récupérer ce dont il a besoin.** Dites à Claude de tirer le contexte lui-même en utilisant des commandes Bash, des outils MCP ou en lisant des fichiers.

Configurez votre environnement

Quelques étapes de configuration rendent Claude Code beaucoup plus efficace dans toutes vos sessions. Pour un aperçu complet des fonctionnalités d'extension et du moment d'utiliser chacune, consultez [Étendre Claude Code](#).

Écrivez un CLAUDE.md efficace

Tip:

Exécutez `/init` pour générer un fichier CLAUDE.md de démarrage basé sur la structure actuelle de votre projet, puis affinez au fil du temps.

CLAUDE.md est un fichier spécial que Claude lit au début de chaque conversation. Incluez des commandes Bash, le style de code et les règles de flux de travail. Cela donne à Claude un contexte persistant qu'il ne peut pas déduire du code seul.

La commande `/init` analyse votre base de code pour détecter les systèmes de construction, les frameworks de test et les modèles de code, vous donnant une base solide à affiner.

Il n'y a pas de format requis pour les fichiers CLAUDE.md, mais gardez-le court et lisible par l'homme. Par exemple :

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 est chargé à chaque session, donc incluez uniquement les choses qui s'appliquent largement. Pour les connaissances de domaine ou les flux de travail qui ne sont pertinents que parfois, utilisez [skills](#) à la place. Claude les charge à la demande sans surcharger chaque conversation.

Gardez-le concis. Pour chaque ligne, demandez-vous : « *Supprimer cela causerait-il à Claude de faire des erreurs ?* » Si non, supprimez-le. Les fichiers CLAUDE.md gonflés font que Claude ignore vos instructions réelles !

 Inclure	 Exclure
Commandes Bash que Claude ne peut pas deviner	Tout ce que Claude peut déduire en lisant le code
Règles de style de code qui diffèrent des valeurs par défaut	Conventions de langage standard que Claude connaît déjà
Instructions de test et exécuteurs de test préférés	Documentation API détaillée (lien vers les docs à la place)
Étiquette du référentiel (nommage des branches, conventions PR)	Informations qui changent fréquemment
Décisions architecturales spécifiques à votre projet	Explications longues ou tutoriels
Particularités de l'environnement de développement (variables d'env requises)	Pratiques évidentes comme « écrire du code propre »
Pièges courants ou comportements non évidents	Descriptions du code fichier par fichier

Si Claude continue à faire quelque chose que vous ne voulez pas malgré une règle contre cela, le fichier est probablement trop long et la règle se perd. Si Claude vous pose des questions qui sont répondues dans CLAUDE.md, la formulation pourrait être ambiguë.

Traitez CLAUDE.md comme du code : révisez-le lorsque les choses vont mal, élaguez-le régulièrement et testez les modifications en observant si le comportement de Claude change réellement.

Vous pouvez affiner les instructions en ajoutant de l'emphase (par exemple, « IMPORTANT » ou « VOUS DEVEZ ») pour améliorer l'adhérence. Vérifiez CLAUDE.md dans git pour que votre équipe puisse contribuer. Le fichier augmente en valeur au fil du temps.

Les fichiers CLAUDE.md peuvent importer des fichiers supplémentaires en utilisant la syntaxe `@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

Vous pouvez placer les fichiers CLAUDE.md dans plusieurs emplacements :

- **Dossier personnel** (`~/ .claude/CLAUDE.md`) : s'applique à toutes les sessions Claude
- **Racine du projet** (`./CLAUDE.md`) : vérifier dans git pour partager avec votre équipe
- **Répertoires parents** : utile pour les monorepos où `root/CLAUDE.md` et `root/foo/CLAUDE.md` sont extraits automatiquement
- **Répertoires enfants** : Claude extrait les fichiers CLAUDE.md enfants à la demande lorsqu'il travaille avec des fichiers dans ces répertoires

Configurez les permissions

Tip:

Utilisez `/permissions` pour autoriser les commandes sûres ou `/sandbox` pour l'isolation au niveau du système d'exploitation. Cela réduit les interruptions tout en vous gardant en contrôle.

Par défaut, Claude Code demande une permission pour les actions qui pourraient modifier votre système : écritures de fichiers, commandes Bash, outils MCP, etc. C'est sûr mais fastidieux. Après la dixième approbation, vous ne révisez vraiment plus, vous cliquez simplement. Il y a deux façons de réduire ces interruptions :

- **Listes blanches de permissions** : permettre des outils spécifiques que vous savez être sûrs (comme `npm run lint` ou `git commit`)

- **Sandboxing** : activer l'isolation au niveau du système d'exploitation qui restreint l'accès au système de fichiers et au réseau, permettant à Claude de travailler plus librement dans des limites définies

Alternativement, utilisez `--dangerously-skip-permissions` pour contourner tous les contrôles de permission pour les flux de travail contenus comme corriger les erreurs de lint ou générer du code passe-partout.

Warning:

Laisser Claude exécuter des commandes arbitraires peut entraîner une perte de données, une corruption du système ou une exfiltration de données via injection de prompt. Utilisez uniquement `--dangerously-skip-permissions` dans un sandbox sans accès à Internet.

En savoir plus sur [la configuration des permissions](#) et [l'activation du sandboxing](#).

Utilisez les outils CLI

Tip:

Dites à Claude Code d'utiliser les outils CLI comme `gh`, `aws`, `gcloud` et `sentry-cli` lors de l'interaction avec les services externes.

Les outils CLI sont le moyen le plus efficace en contexte d'interagir avec les services externes. Si vous utilisez GitHub, installez le CLI `gh`. Claude sait comment l'utiliser pour créer des problèmes, ouvrir des demandes de tirage et lire les commentaires. Sans `gh`, Claude peut toujours utiliser l'API GitHub, mais les demandes non authentifiées atteignent souvent les limites de débit.

Claude est également efficace pour apprendre les outils CLI qu'il ne connaît pas déjà. Essayez des invites comme `Utilisez 'foo-cli-tool --help' pour en savoir plus sur l'outil foo, puis utilisez-le pour résoudre A, B, C.`

Connectez les serveurs MCP

Tip:

Exécutez `claude mcp add` pour connecter les outils externes comme Notion, Figma ou votre base de données.

Avec les [serveurs MCP](#), vous pouvez demander à Claude d'implémenter des fonctionnalités à partir de suivi de problèmes, interroger des bases de données, analyser les données de surveillance, intégrer les conceptions de Figma et automatiser les flux de travail.

Configurez les hooks

Tip:

Utilisez les hooks pour les actions qui doivent se produire à chaque fois sans exception.

Les [hooks](#) exécutent automatiquement les scripts à des points spécifiques du flux de travail de Claude. Contrairement aux instructions CLAUDE.md qui sont consultatives, les hooks sont déterministes et garantissent que l'action se produit.

Claude peut écrire des hooks pour vous. Essayez des invites comme « *Écrire un hook qui exécute eslint après chaque édition de fichier* » ou « *Écrire un hook qui bloque les écritures dans le dossier migrations.* » Exécutez `/hooks` pour la configuration interactive, ou modifiez `.claude/settings.json` directement.

Créez des skills

Tip:

Créez des fichiers `SKILL.md` dans `.claude/skills/` pour donner à Claude des connaissances de domaine et des flux de travail réutilisables.

Les [skills](#) étendent les connaissances de Claude avec des informations spécifiques à votre projet, équipe ou domaine. Claude les applique automatiquement lorsqu'elles sont pertinentes, ou vous pouvez les invoquer directement avec `/skill-name`.

Créez une skill en ajoutant un répertoire avec un `SKILL.md` à `.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/)
```

Les skills peuvent également définir des flux de travail réutilisables que vous invoquez directement :

```
---
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
```

Exécutez `/fix-issue 1234` pour l'invoquer. Utilisez `disable-model-invocation: true` pour les flux de travail avec des effets secondaires que vous souhaitez déclencher manuellement.

Créez des subagents personnalisés

Tip:

Définissez des assistants spécialisés dans `.claude/agents/` que Claude peut déléguer pour les tâches isolées.

Les [subagents](#) s'exécutent dans leur propre contexte avec leur propre ensemble d'outils autorisés. Ils sont utiles pour les tâches qui lisent de nombreux fichiers ou qui ont besoin d'une attention spécialisée sans encombrer votre conversation 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.
```

Dites à Claude d'utiliser les subagents explicitement : « *Utilisez un subagent pour réviser ce code pour les problèmes de sécurité.* »

Installez les plugins

Tip:

Exécutez `/plugin` pour parcourir la marketplace. Les plugins ajoutent des skills, des outils et des intégrations sans configuration.

Les [plugins](#) regroupent les skills, les hooks, les subagents et les serveurs MCP dans une seule unité installable de la communauté et d'Anthropic. Si vous travaillez avec un langage typé, installez un [plugin d'intelligence de code](#) pour donner à Claude une navigation de symboles précise et une détection d'erreur automatique après les éditions.

Pour des conseils sur le choix entre les skills, les subagents, les hooks et MCP, consultez [Étendre Claude Code](#).

Communiquez efficacement

La façon dont vous communiquez avec Claude Code a un impact significatif sur la qualité des résultats.

Posez des questions sur la base de code

Tip:

Posez à Claude les questions que vous poseriez à un ingénieur senior.

Lors de l'intégration à une nouvelle base de code, utilisez Claude Code pour l'apprentissage et l'exploration. Vous pouvez poser à Claude les mêmes types de questions que vous poseriez à un autre ingénieur :

- Comment fonctionne la journalisation ?
- Comment créer un nouveau point de terminaison API ?
- Que fait `async move { ... }` à la ligne 134 de `foo.rs` ?
- Quels cas limites `CustomerOnboardingFlowImpl` gère-t-il ?
- Pourquoi ce code appelle-t-il `foo()` au lieu de `bar()` à la ligne 333 ?

Utiliser Claude Code de cette façon est un flux de travail d'intégration efficace, améliorant le temps de montée en charge et réduisant la charge sur les autres ingénieurs. Aucune invite spéciale requise : posez les questions directement.

Laissez Claude vous interviewer

Tip:

Pour les fonctionnalités plus grandes, laissez Claude vous interviewer d'abord. Commencez par une invite minimale et demandez à Claude de vous interviewer en utilisant l'outil

`AskUserQuestion` .

Claude pose des questions sur les choses que vous n'auriez peut-être pas considérées, y compris l'implémentation technique, l'interface utilisateur/UX, les cas limites et les compromis.

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.

Une fois la spécification complète, démarrez une nouvelle session pour l'exécuter. La nouvelle session a un contexte propre entièrement axé sur l'implémentation, et vous avez une spécification écrite à référencer.

Gérez votre session

Les conversations sont persistantes et réversibles. Utilisez cela à votre avantage !

Corrigez la trajectoire tôt et souvent

Tip:

Corrigez Claude dès que vous remarquez qu'il s'écarte de la bonne voie.

Les meilleurs résultats proviennent de boucles de rétroaction serrées. Bien que Claude résolve occasionnellement les problèmes parfaitement à la première tentative, le corriger rapidement produit généralement de meilleures solutions plus rapidement.

- **Esc** : arrêtez Claude en pleine action avec la touche **Esc**. Le contexte est préservé, vous pouvez donc rediriger.
- **Esc + Esc** ou **/rewind** : appuyez sur **Esc** deux fois ou exécutez **/rewind** pour ouvrir le menu de rembobinage et restaurer la conversation et l'état du code précédents, ou résumer à partir d'un message sélectionné.
- **« Annuler cela »** : demandez à Claude d'annuler ses modifications.
- **/clear** : réinitialiser le contexte entre les tâches non liées. Les sessions longues avec un contexte non pertinent peuvent réduire les performances.

Si vous avez corrigé Claude plus de deux fois sur le même problème dans une session, le contexte est encombré d'approches échouées. Exécutez `/clear` et recommencez avec une invite plus spécifique qui incorpore ce que vous avez appris. Une session propre avec une meilleure invite surpasse presque toujours une session longue avec des corrections accumulées.

Gérez le contexte agressivement

Tip:

Exécutez `/clear` entre les tâches non liées pour réinitialiser le contexte.

Claude Code compacte automatiquement l'historique de conversation lorsque vous approchez des limites de contexte, ce qui préserve le code et les décisions importants tout en libérant de l'espace.

Pendant les sessions longues, la fenêtre de contexte de Claude peut se remplir de conversations non pertinentes, de contenu de fichiers et de commandes. Cela peut réduire les performances et parfois distraire Claude.

- Utilisez `/clear` fréquemment entre les tâches pour réinitialiser complètement la fenêtre de contexte
- Lorsque le compactage automatique se déclenche, Claude résume ce qui importe le plus, y compris les modèles de code, les états de fichiers et les décisions clés
- Pour plus de contrôle, exécutez `/compact <instructions>`, comme `/compact Focus on the API changes`
- Pour compacter uniquement une partie de la conversation, utilisez `Esc + Esc` ou `/rewind`, sélectionnez un point de contrôle de message et choisissez **Summarize from here**. Cela condense les messages à partir de ce point tout en gardant le contexte antérieur intact.
- Personnalisez le comportement de compactage dans CLAUDE.md avec des instructions comme « `Lors du compactage, toujours préserver la liste complète des fichiers modifiés et toutes les commandes de test` » pour assurer que le contexte critique survit à la résumé
- Pour les questions rapides qui n'ont pas besoin de rester en contexte, utilisez `/btw`. La réponse apparaît dans une superposition rejetable et n'entre jamais dans l'historique de conversation, vous pouvez donc vérifier un détail sans augmenter le contexte.

Utilisez les subagents pour l'investigation

Tip:

Déléguez la recherche avec « **utiliser les subagents pour enquêter sur X** ». Ils explorent dans un contexte séparé, gardant votre conversation principale propre pour l'implémentation.

Puisque le contexte est votre contrainte fondamentale, les subagents sont l'un des outils les plus puissants disponibles. Lorsque Claude enquête sur une base de code, il lit de nombreux fichiers, qui consomment tous votre contexte. Les subagents s'exécutent dans des fenêtres de contexte séparées et rapportent les résumés :

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

Le subagent explore la base de code, lit les fichiers pertinents et rapporte les résultats, tout sans encombrer votre conversation principale.

Vous pouvez également utiliser les subagents pour la vérification après que Claude implémente quelque chose :

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

Rembobinez avec des points de contrôle

Tip:

Chaque action que Claude fait crée un point de contrôle. Vous pouvez restaurer la conversation, le code ou les deux à n'importe quel point de contrôle précédent.

Claude crée automatiquement des points de contrôle avant les modifications. Appuyez deux fois sur **Escape** ou exécutez **/rewind** pour ouvrir le menu de rembobinage. Vous pouvez restaurer la conversation uniquement, restaurer le code uniquement, restaurer les deux ou résumer à partir d'un message sélectionné. Consultez [Checkpointing](#) pour plus de détails.

Au lieu de planifier soigneusement chaque mouvement, vous pouvez dire à Claude d'essayer quelque chose de risqué. Si cela ne fonctionne pas, rembobinez et essayez une approche différente. Les points de contrôle persistent entre les sessions, vous pouvez donc fermer votre terminal et toujours rembobiner plus tard.

Warning:

Les points de contrôle ne suivent que les modifications apportées *par Claude*, pas les processus externes. Ce n'est pas un remplacement pour git.

Reprenez les conversations

Tip:

Exécutez `claude --continue` pour reprendre là où vous vous êtes arrêté, ou `--resume` pour choisir parmi les sessions récentes.

Claude Code enregistre les conversations localement. Lorsqu'une tâche s'étend sur plusieurs sessions, vous n'avez pas à réexpliquer le contexte :

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

Utilisez `/rename` pour donner aux sessions des noms descriptifs comme « `oauth-migration` » ou « `debugging-memory-leak` » pour pouvoir les trouver plus tard. Traitez les sessions comme des branches : différents flux de travail peuvent avoir des contextes séparés et persistants.

Automatisez et mettez à l'échelle

Une fois que vous êtes efficace avec un Claude, multipliez votre production avec des sessions parallèles, le mode non interactif et les modèles de fan-out.

Tout ce qui précède suppose un humain, un Claude et une conversation. Mais Claude Code se met à l'échelle horizontalement. Les techniques de cette section montrent comment vous pouvez en faire plus.

Exécutez le mode non interactif

Tip:

Utilisez `claude -p "prompt"` dans CI, les hooks de pré-commit ou les scripts. Ajoutez `--output-format stream-json` pour la sortie JSON en streaming.

Avec `claude -p "your prompt"`, vous pouvez exécuter Claude de manière non interactive, sans session. Le mode non interactif est la façon dont vous intégrez Claude dans les pipelines CI, les hooks de pré-commit ou tout flux de travail automatisé. Les formats de sortie vous permettent d'analyser les résultats par programmation : texte brut, JSON ou JSON en 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
```

Exécutez plusieurs sessions Claude

Tip:

Exécutez plusieurs sessions Claude en parallèle pour accélérer le développement, exécuter des expériences isolées ou démarrer des flux de travail complexes.

Il y a trois façons principales d'exécuter des sessions parallèles :

- [Application de bureau Claude Code](#) : Gérez visuellement plusieurs sessions locales. Chaque session obtient son propre worktree isolé.
- [Claude Code sur le web](#) : Exécutez sur l'infrastructure cloud sécurisée d'Anthropic dans des VM isolées.
- [Équipes d'agents](#) : Coordination automatisée de plusieurs sessions avec des tâches partagées, la messagerie et un chef d'équipe.

Au-delà de la parallélisation du travail, plusieurs sessions permettent des flux de travail axés sur la qualité. Un contexte frais améliore la révision de code puisque Claude ne sera pas biaisé vers le code qu’il vient d’écrire.

Par exemple, utilisez un modèle Writer/Reviewer :

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

Vous pouvez faire quelque chose de similaire avec les tests : avoir un Claude écrire des tests, puis un autre écrire du code pour les réussir.

Fan out sur les fichiers

Tip:

Bouclez à travers les tâches en appelant `claude -p` pour chacune. Utilisez `--allowedTools` pour délimiter les permissions pour les opérations par lot.

Pour les migrations ou analyses à grande échelle, vous pouvez distribuer le travail sur de nombreuses invocations Claude parallèles :

Step 1: Générez une liste de tâches

Demandez à Claude de lister tous les fichiers qui doivent être migrés (par exemple, `list all 2,000 Python files that need migrating`)

Step 2: Écrivez un script pour boucler à travers la liste

```
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: Testez sur quelques fichiers, puis exécutez à l'échelle

Affinez votre invite en fonction de ce qui se passe mal avec les 2-3 premiers fichiers, puis exécutez sur l'ensemble complet. L'indicateur `--allowedTools` restreint ce que Claude peut faire, ce qui importe lorsque vous exécutez sans surveillance.

Vous pouvez également intégrer Claude dans les pipelines de données/traitement existants :

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

Utilisez `--verbose` pour le débogage pendant le développement, et désactivez-le en production.

Évitez les modèles d'échec courants

Ce sont des erreurs courantes. Les reconnaître tôt économise du temps :

- **La session fourre-tout.** Vous commencez par une tâche, puis demandez à Claude quelque chose d'autre, puis revenez à la première tâche. Le contexte est plein d'informations non pertinentes. > **Correction** : `/clear` entre les tâches non liées.
- **Corriger encore et encore.** Claude fait quelque chose de mal, vous le corrigez, c'est toujours mal, vous corrigez à nouveau. Le contexte est pollué par des approches échouées. > **Correction** : Après deux corrections échouées, `/clear` et écrivez une meilleure invite initiale incorporant ce que vous avez appris.
- **Le CLAUDE.md sur-spécifié.** Si votre CLAUDE.md est trop long, Claude ignore la moitié car les règles importantes se perdent dans le bruit. > **Correction** : Élaguez impitoyablement. Si Claude fait déjà quelque chose correctement sans l'instruction, supprimez-le ou convertissez-le en hook.
- **L'écart de confiance-puis-vérification.** Claude produit une implémentation plausible qui ne gère pas les cas limites. > **Correction** : Fournissez toujours une vérification (tests, scripts, captures d'écran). Si vous ne pouvez pas la vérifier, ne la déployez pas.

- **L'exploration infinie.** Vous demandez à Claude d'« enquêter » sur quelque chose sans le délimiter. Claude lit des centaines de fichiers, remplissant le contexte. > **Correction :** Délimitez les enquêtes étroitement ou utilisez les subagents pour que l'exploration ne consomme pas votre contexte principal.
-

Développez votre intuition

Les modèles de ce guide ne sont pas gravés dans le marbre. Ce sont des points de départ qui fonctionnent bien en général, mais pourraient ne pas être optimaux pour chaque situation.

Parfois, vous *devriez* laisser le contexte s'accumuler parce que vous êtes profondément dans un problème complexe et l'historique est précieux. Parfois, vous devriez ignorer la planification et laisser Claude le découvrir parce que la tâche est exploratoire. Parfois, une invite vague est exactement ce qu'il faut parce que vous voulez voir comment Claude interprète le problème avant de le contraindre.

Faites attention à ce qui fonctionne. Lorsque Claude produit une excellente sortie, remarquez ce que vous avez fait : la structure de l'invite, le contexte que vous avez fourni, le mode dans lequel vous étiez. Lorsque Claude lutte, demandez-vous pourquoi. Le contexte était-il trop bruyant ? L'invite trop vague ? La tâche trop grande pour une seule passe ?

Au fil du temps, vous développerez une intuition qu'aucun guide ne peut capturer. Vous saurez quand être spécifique et quand être ouvert, quand planifier et quand explorer, quand effacer le contexte et quand le laisser s'accumuler.

Ressources connexes

- [Comment fonctionne Claude Code](#) : la boucle agentique, les outils et la gestion du contexte
- [Étendre Claude Code](#) : skills, hooks, MCP, subagents et plugins
- [Flux de travail courants](#) : recettes étape par étape pour le débogage, les tests, les PR et plus
- [CLAUDE.md](#) : stocker les conventions de projet et le contexte persistant

Flux de travail courants

Guides étape par étape pour explorer les bases de code, corriger les bogues, refactoriser, tester et autres tâches quotidiennes avec Claude Code.

Cette page couvre les flux de travail pratiques pour le développement quotidien : explorer du code inconnu, déboguer, refactoriser, écrire des tests, créer des PR et gérer les sessions. Chaque section inclut des exemples de prompts que vous pouvez adapter à vos propres projets. Pour des modèles et des conseils de plus haut niveau, consultez [Bonnes pratiques](#).

Comprendre les nouvelles bases de code

Obtenir un aperçu rapide de la base de code

Supposons que vous venez de rejoindre un nouveau projet et que vous devez comprendre rapidement sa structure.

Step 1: Accédez au répertoire racine du projet

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

Step 2: Démarrez Claude Code

```
claude
```

Step 3: Demandez un aperçu de haut niveau

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

Step 4: Approfondissez les composants spécifiques

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

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



```
how is authentication handled?
```

Tip:

Conseils :

- Commencez par des questions larges, puis réduisez à des domaines spécifiques
- Posez des questions sur les conventions de codage et les modèles utilisés dans le projet
- Demandez un glossaire des termes spécifiques au projet

Trouver du code pertinent

Supposons que vous ayez besoin de localiser du code lié à une fonctionnalité ou une capacité spécifique.

Step 1: Demandez à Claude de trouver les fichiers pertinents

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

Step 2: Obtenez du contexte sur la façon dont les composants interagissent

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

Step 3: Comprenez le flux d'exécution

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

Tip:

Conseils :

- Soyez spécifique sur ce que vous recherchez
- Utilisez le langage du domaine du projet
- Installez un [plugin d'intelligence de code](#) pour votre langage afin de donner à Claude une navigation précise ' aller à la définition ' et ' trouver les références '

Corriger les bogues efficacement

Supposons que vous ayez rencontré un message d'erreur et que vous ayez besoin de trouver et de corriger sa source.

Step 1: Partagez l'erreur avec Claude

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

Step 2: Demandez des recommandations de correction

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

Step 3: Appliquez la correction

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

Tip:

Conseils :

- Dites à Claude la commande pour reproduire le problème et obtenir une trace de pile
- Mentionnez les étapes pour reproduire l'erreur
- Faites savoir à Claude si l'erreur est intermittente ou cohérente

Refactoriser le code

Supposons que vous ayez besoin de mettre à jour du code ancien pour utiliser des modèles et des pratiques modernes.

Step 1: Identifiez le code hérité pour la refactorisation

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

Step 2: Obtenez des recommandations de refactorisation

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

Step 3: Appliquez les modifications en toute sécurité

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

Step 4: Vérifiez la refactorisation

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

Tip:

Conseils :

- Demandez à Claude d'expliquer les avantages de l'approche moderne
- Demandez que les modifications maintiennent la compatibilité rétroactive si nécessaire
- Effectuez la refactorisation par petits incréments testables

Utiliser des subagents spécialisés

Supposons que vous souhaitiez utiliser des subagents IA spécialisés pour gérer des tâches spécifiques plus efficacement.

Step 1: Afficher les subagents disponibles

```
/agents
```

Cela affiche tous les subagents disponibles et vous permet d'en créer de nouveaux.

Step 2: Utiliser les subagents automatiquement

Claude Code délègue automatiquement les tâches appropriées aux subagents spécialisés :

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

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

Step 3: Demander explicitement des subagents spécifiques

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

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

Step 4: Créer des subagents personnalisés pour votre flux de travail

```
/agents
```

Sélectionnez ensuite « Créer un nouveau subagent » et suivez les invites pour définir :

- Un identifiant unique qui décrit l'objectif du subagent (par exemple, `code-reviewer`, `api-designer`).
- Quand Claude doit utiliser cet agent
- Quels outils il peut accéder
- Une invite système décrivant le rôle et le comportement de l'agent

Tip:

Conseils :

- Créez des subagents spécifiques au projet dans `.claude/agents/` pour le partage en équipe
- Utilisez des champs `description` descriptifs pour activer la délégation automatique
- Limitez l'accès aux outils à ce que chaque subagent a réellement besoin
- Consultez la [documentation des subagents](#) pour des exemples détaillés

Utiliser le Plan Mode pour une analyse de code sûre

Plan Mode demande à Claude de créer un plan en analysant la base de code avec des opérations en lecture seule, parfait pour explorer les bases de code, planifier des modifications complexes ou examiner le code en toute sécurité. En Plan Mode, Claude utilise

`AskUserQuestion` pour recueillir les exigences et clarifier vos objectifs avant de proposer un plan.

Quand utiliser Plan Mode

- **Implémentation multi-étapes** : Quand votre fonctionnalité nécessite de faire des modifications à de nombreux fichiers
- **Exploration de code** : Quand vous souhaitez rechercher la base de code en profondeur avant de modifier quoi que ce soit
- **Développement interactif** : Quand vous souhaitez itérer sur la direction avec Claude

Comment utiliser Plan Mode

Activez Plan Mode pendant une session

Vous pouvez basculer en Plan Mode pendant une session en utilisant **Maj+Tab** pour parcourir les modes de permission.

Si vous êtes en Mode Normal, **Maj+Tab** bascule d'abord en Mode Auto-Accept, indiqué par `▶▶ accept edits on` en bas du terminal. Un **Maj+Tab** ultérieur basculera en Plan Mode, indiqué par `⏏ plan mode on`.

Démarrez une nouvelle session en Plan Mode

Pour démarrer une nouvelle session en Plan Mode, utilisez le drapeau `--permission-mode plan` :

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

Exécutez des requêtes « headless » en Plan Mode

Vous pouvez également exécuter une requête en Plan Mode directement avec `-p` (c'est-à-dire en « [mode headless](#) ») :

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

Exemple : Planifier une refactorisation complexe

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

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

Claude analyse l'implémentation actuelle et crée un plan complet. Affinez avec des suites :

What about backward compatibility?

How should we handle database migration?

Tip:

Appuyez sur **Ctrl+G** pour ouvrir le plan dans votre éditeur de texte par défaut, où vous pouvez le modifier directement avant que Claude ne procède.

Configurer Plan Mode par défaut

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

Consultez la [documentation des paramètres](#) pour plus d'options de configuration.

Travailler avec les tests

Supposons que vous ayez besoin d'ajouter des tests pour du code non couvert.

Step 1: Identifiez le code non testé

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

Step 2: Générez l'échafaudage des tests

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

Step 3: Ajoutez des cas de test significatifs

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

Step 4: Exécutez et vérifiez les tests

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

Claude peut générer des tests qui suivent les modèles et conventions existants de votre projet. Lorsque vous demandez des tests, soyez spécifique sur le comportement que vous souhaitez vérifier. Claude examine vos fichiers de test existants pour correspondre au style, aux frameworks et aux modèles d'assertion déjà en usage.

Pour une couverture complète, demandez à Claude d'identifier les cas limites que vous auriez pu manquer. Claude peut analyser vos chemins de code et suggérer des tests pour les conditions d'erreur, les valeurs limites et les entrées inattendues qui sont faciles à oublier.

Créer des pull requests

Vous pouvez créer des pull requests en demandant directement à Claude (« créer une pr pour mes modifications »), ou guider Claude à travers cela étape par étape :

Step 1: Résumez vos modifications

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

Step 2: Générez une pull request

```
create a pr
```

Step 3: Examinez et affinez

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

Lorsque vous créez une PR en utilisant `gh pr create`, la session est automatiquement liée à cette PR. Vous pouvez la reprendre plus tard avec `claude --from-pr <number>`.

Tip:

Examinez la PR générée par Claude avant de la soumettre et demandez à Claude de mettre en évidence les risques ou considérations potentiels.

Gérer la documentation

Supposons que vous ayez besoin d'ajouter ou de mettre à jour la documentation de votre code.

Step 1: Identifiez le code non documenté

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

Step 2: Générez la documentation

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

Step 3: Examinez et améliorez

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

Step 4: Vérifiez la documentation

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


Tip:

Conseils :

- Spécifiez le style de documentation que vous souhaitez (JSDoc, docstrings, etc.)
- Demandez des exemples dans la documentation
- Demandez la documentation pour les API publiques, les interfaces et la logique complexe

Travailler avec les images

Supposons que vous ayez besoin de travailler avec des images dans votre base de code et que vous souhaitiez l'aide de Claude pour analyser le contenu des images.

Step 1: Ajoutez une image à la conversation

Vous pouvez utiliser l'une de ces méthodes :

1. Glissez-déposez une image dans la fenêtre Claude Code
2. Copiez une image et collez-la dans l'interface CLI avec `ctrl+v` (N'utilisez pas `cmd+v`)
3. Fournissez un chemin d'image à Claude. Par exemple, « Analyser cette image : `/path/to/your/image.png` »

Step 2: Demandez à Claude d'analyser l'image

What does this image show?

Describe the UI elements in this screenshot

Are there any problematic elements in this diagram?

Step 3: Utilisez les images pour le contexte

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: Obtenez des suggestions de code à partir du contenu visuel

Generate CSS to match this design mockup

What HTML structure would recreate this component?

Tip:

Conseils :

- Utilisez les images quand les descriptions textuelles seraient peu claires ou fastidieuses
- Incluez des captures d'écran d'erreurs, de conceptions d'interface utilisateur ou de diagrammes pour un meilleur contexte
- Vous pouvez travailler avec plusieurs images dans une conversation
- L'analyse d'images fonctionne avec les diagrammes, les captures d'écran, les maquettes et bien d'autres
- Quand Claude référence des images (par exemple, `[Image #1]`), `Cmd+Click` (Mac) ou `Ctrl+Click` (Windows/Linux) le lien pour ouvrir l'image dans votre visionneuse par défaut

Référencer les fichiers et répertoires

Utilisez `@` pour inclure rapidement des fichiers ou des répertoires sans attendre que Claude les lise.

Step 1: Référez un seul fichier

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

Cela inclut le contenu complet du fichier dans la conversation.

Step 2: Référez un répertoire

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

Cela fournit une liste de répertoires avec les informations de fichier.

Step 3: Référez les ressources MCP

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

Cela récupère les données des serveurs MCP connectés en utilisant le format `@server:resource`. Consultez [Ressources MCP](#) pour plus de détails.

Tip:

Conseils :

- Les chemins de fichiers peuvent être relatifs ou absolus
- Les références de fichiers `@` ajoutent `CLAUDE.md` dans le répertoire du fichier et les répertoires parents au contexte
- Les références de répertoires affichent les listes de fichiers, pas les contenus
- Vous pouvez référencer plusieurs fichiers dans un seul message (par exemple, « `@file1.js` et `@file2.js` »)

Utiliser la réflexion étendue (mode de réflexion)

La [réflexion étendue](#) est activée par défaut, donnant à Claude l'espace pour raisonner sur les problèmes complexes étape par étape avant de répondre. Ce raisonnement est visible en mode verbeux, que vous pouvez activer avec `Ctrl+0`.

De plus, Opus 4.6 introduit le raisonnement adaptatif : au lieu d'un budget de jetons de réflexion fixe, le modèle alloue dynamiquement la réflexion en fonction de votre paramètre [niveau d'effort](#). La réflexion étendue et le raisonnement adaptatif fonctionnent ensemble pour vous donner le contrôle sur la profondeur de la réflexion de Claude avant de répondre.

La réflexion étendue est particulièrement précieuse pour les décisions architecturales complexes, les bogues difficiles, la planification de l'implémentation multi-étapes et l'évaluation des compromis entre différentes approches.

Note:

Les phrases comme « think », « think hard » et « think more » sont interprétées comme des instructions d'invite régulières et n'allouent pas de jetons de réflexion.

Configurer le mode de réflexion

La réflexion est activée par défaut, mais vous pouvez l'ajuster ou la désactiver.

Portée	Comment configurer	Détails
Niveau d'effort	Ajustez dans <code>/model</code> ou définissez <code>CLAUDE_CODE_EFFORT_LEVEL</code>	Contrôlez la profondeur de la réflexion pour Opus 4.6 et Sonnet 4.6 : low, medium, high. Consultez Ajuster le niveau d'effort
Mot-clé <code>ultrathink</code>	Incluez « ultrathink » n'importe où dans votre prompt	Définit l'effort à high pour ce tour sur Opus 4.6 et Sonnet 4.6. Utile pour les tâches ponctuelles nécessitant un raisonnement profond sans modifier définitivement votre paramètre d'effort
Raccourci de basculement	Appuyez sur <code>Option+T</code> (macOS) ou <code>Alt+T</code> (Windows/Linux)	Basculez la réflexion activée/désactivée pour la session actuelle (tous les modèles). Peut nécessiter une configuration du terminal pour activer les raccourcis de la touche Option
Défaut global	Utilisez <code>/config</code> pour basculer le mode de réflexion	Définit votre défaut sur tous les projets (tous les modèles). Enregistré en tant que <code>alwaysThinkingEnabled</code> dans <code>~/.claude/settings.json</code>
Limiter le budget de jetons	Définissez la variable d'environnement <code>MAX_THINKING_TOKENS</code>	Limitez le budget de réflexion à un nombre spécifique de jetons (ignoré sur Opus 4.6 sauf s'il est défini à 0). Exemple : <code>export MAX_THINKING_TOKENS=10000</code>

Pour afficher le processus de réflexion de Claude, appuyez sur `Ctrl+O` pour basculer le mode verbeux et voir le raisonnement interne affiché en texte gris italique.

Comment fonctionne la réflexion étendue

La réflexion étendue contrôle la quantité de raisonnement interne que Claude effectue avant de répondre. Plus de réflexion fournit plus d'espace pour explorer les solutions, analyser les cas limites et corriger les erreurs.

Avec Opus 4.6, la réflexion utilise le raisonnement adaptatif : le modèle alloue dynamiquement les jetons de réflexion en fonction du [niveau d'effort](#) que vous sélectionnez (low, medium, high). C'est la façon recommandée d'ajuster le compromis entre la vitesse et la profondeur du raisonnement.

Avec d'autres modèles, la réflexion utilise un budget fixe de jusqu'à 31 999 jetons de votre budget de sortie. Vous pouvez limiter cela avec la variable d'environnement `MAX_THINKING_TOKENS`, ou désactiver complètement la réflexion via `/config` ou le basculement `Option+T` / `Alt+T`.

`MAX_THINKING_TOKENS` est ignoré sur Opus 4.6 et Sonnet 4.6, car le raisonnement adaptatif contrôle la profondeur de la réflexion à la place. L'une des exceptions : définir `MAX_THINKING_TOKENS=0` désactive toujours complètement la réflexion sur n'importe quel modèle. Pour désactiver la réflexion adaptative et revenir au budget de réflexion fixe, définissez `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Consultez [variables d'environnement](#).

Warning:

Vous êtes facturé pour tous les jetons de réflexion utilisés, même si les modèles Claude 4 affichent une réflexion résumée

Reprendre les conversations précédentes

Lors du démarrage de Claude Code, vous pouvez reprendre une session précédente :

- `claude --continue` continue la conversation la plus récente dans le répertoire actuel
- `claude --resume` ouvre un sélecteur de conversation ou reprend par nom
- `claude --from-pr 123` reprend les sessions liées à une pull request spécifique

À partir d'une session active, utilisez `/resume` pour basculer vers une conversation différente.

Les sessions sont stockées par répertoire de projet. Le sélecteur `/resume` affiche les sessions du même référentiel git, y compris les worktrees.

Nommez vos sessions

Donnez aux sessions des noms descriptifs pour les trouver plus tard. C'est une bonne pratique lorsque vous travaillez sur plusieurs tâches ou fonctionnalités.

Step 1: Nommez la session actuelle

Utilisez `/rename` pendant une session pour lui donner un nom mémorable :

```
/rename auth-refactor
```

Vous pouvez également renommer n'importe quelle session à partir du sélecteur : exécutez `/resume` , accédez à une session et appuyez sur `R` .

Step 2: Reprenez par nom plus tard

À partir de la ligne de commande :

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

Ou à partir d'une session active :

```
/resume auth-refactor
```

Utiliser le sélecteur de session

La commande `/resume` (ou `claude --resume` sans arguments) ouvre un sélecteur de session interactif avec ces fonctionnalités :

Raccourcis clavier dans le sélecteur :

Raccourci	Action
↑ / ↓	Naviguer entre les sessions
→ / ←	Développer ou réduire les sessions groupées
Entrée	Sélectionner et reprendre la session en surbrillance
P	Aperçu du contenu de la session
R	Renommer la session en surbrillance
/	Rechercher pour filtrer les sessions

Raccourci	Action
A	Basculer entre le répertoire actuel et tous les projets
B	Filtrer les sessions de votre branche git actuelle
Échap	Quitter le sélecteur ou le mode de recherche

Organisation des sessions :

Le sélecteur affiche les sessions avec des métadonnées utiles :

- Nom de la session ou invite initiale
- Temps écoulé depuis la dernière activité
- Nombre de messages
- Branche Git (le cas échéant)

Les sessions bifurquées (créées avec `/rewind` ou `--fork-session`) sont groupées ensemble sous leur session racine, ce qui facilite la recherche de conversations connexes.

Tip:

Conseils :

- **Nommez les sessions tôt** : Utilisez `/rename` au démarrage du travail sur une tâche distincte — il est beaucoup plus facile de trouver « payment-integration » que « explain this function » plus tard
- Utilisez `--continue` pour un accès rapide à votre conversation la plus récente dans le répertoire actuel
- Utilisez `--resume session-name` quand vous savez quelle session vous avez besoin
- Utilisez `--resume` (sans nom) quand vous avez besoin de parcourir et de sélectionner
- Pour les scripts, utilisez `claude --continue --print "prompt"` pour reprendre en mode non interactif
- Appuyez sur **P** dans le sélecteur pour prévisualiser une session avant de la reprendre
- La conversation reprise démarre avec le même modèle et la même configuration que l'original

Comment cela fonctionne :

1. **Stockage des conversations** : Toutes les conversations sont automatiquement enregistrées localement avec leur historique complet des messages
2. **Désérialisation des messages** : Lors de la reprise, l'historique complet des messages est restauré pour maintenir le contexte
3. **État des outils** : L'utilisation des outils et les résultats de la conversation précédente sont préservés

4. Restauration du contexte : La conversation reprend avec tout le contexte précédent intact

Exécuter des sessions Claude Code parallèles avec Git worktrees

Lorsque vous travaillez sur plusieurs tâches à la fois, vous avez besoin que chaque session Claude ait sa propre copie de la base de code afin que les modifications ne se heurtent pas. Les worktrees Git résolvent ce problème en créant des répertoires de travail séparés qui ont chacun leurs propres fichiers et branche, tout en partageant le même historique de référentiel et les mêmes connexions distantes. Cela signifie que vous pouvez avoir Claude travaillant sur une fonctionnalité dans un worktree tout en corrigeant un bogue dans un autre, sans que l'une ou l'autre session n'interfère avec l'autre.

Utilisez le drapeau `--worktree` (`-w`) pour créer un worktree isolé et démarrer Claude dedans. La valeur que vous transmettez devient le nom du répertoire worktree et le nom de la branche :

```
## Démarrez Claude dans un worktree nommé « feature-auth »
## Crée .claude/worktrees/feature-auth/ avec une nouvelle branche
claude --worktree feature-auth

## Démarrez une autre session dans un worktree séparé
claude --worktree bugfix-123
```

Si vous omettez le nom, Claude en génère un automatiquement :

```
## Génère automatiquement un nom comme « bright-running-fox »
claude --worktree
```

Les worktrees sont créés à `<repo>/.claude/worktrees/<name>` et se ramifient à partir de la branche distante par défaut. La branche worktree est nommée `worktree-<name>`.

Vous pouvez également demander à Claude de « travailler dans un worktree » ou « démarrer un worktree » pendant une session, et il en créera un automatiquement.

Worktrees des subagents

Les subagents peuvent également utiliser l'isolation worktree pour travailler en parallèle sans conflits. Demandez à Claude d'« utiliser les worktrees pour vos agents » ou configurez-le dans un [subagent personnalisé](#) en ajoutant `isolation: worktree` au front-matter de l'agent. Chaque subagent obtient son propre worktree qui est automatiquement nettoyé quand le subagent se termine sans modifications.

Nettoyage des worktrees

Lorsque vous quittez une session worktree, Claude gère le nettoyage en fonction de si vous avez apporté des modifications :

- **Pas de modifications** : le worktree et sa branche sont supprimés automatiquement
- **Les modifications ou les commits existent** : Claude vous demande de conserver ou de supprimer le worktree. La conservation préserve le répertoire et la branche afin que vous puissiez revenir plus tard. La suppression supprime le répertoire worktree et sa branche, en supprimant toutes les modifications non validées et les commits

Pour nettoyer les worktrees en dehors d'une session Claude, utilisez la [gestion manuelle des worktrees](#).

Tip:

Ajoutez `.claude/worktrees/` à votre `.gitignore` pour empêcher le contenu des worktrees d'apparaître comme des fichiers non suivis dans votre référentiel principal.

Gérer les worktrees manuellement

Pour plus de contrôle sur l'emplacement du worktree et la configuration de la branche, créez des worktrees directement avec Git. C'est utile quand vous avez besoin de vérifier une branche existante spécifique ou de placer le worktree en dehors du référentiel.

```
## Créez un worktree avec une nouvelle branche
git worktree add ../project-feature-a -b feature-a

## Créez un worktree avec une branche existante
git worktree add ../project-bugfix bugfix-123

## Démarrez Claude dans le worktree
cd ../project-feature-a && claude

## Nettoyez quand vous avez terminé
git worktree list
git worktree remove ../project-feature-a
```

En savoir plus dans la [documentation officielle de Git worktree](#).

Tip:

N'oubliez pas d'initialiser votre environnement de développement dans chaque nouveau worktree selon la configuration de votre projet. Selon votre pile, cela peut inclure l'exécution de l'installation des dépendances (`npm install` , `yarn`), la configuration des environnements virtuels ou le suivi du processus de configuration standard de votre projet.

Contrôle de version non-git

L'isolation worktree fonctionne avec git par défaut. Pour d'autres systèmes de contrôle de version comme SVN, Perforce ou Mercurial, configurez les hooks [WorktreeCreate et WorktreeRemove](#) pour fournir une logique de création et de nettoyage de worktree personnalisée. Lorsqu'ils sont configurés, ces hooks remplacent le comportement git par défaut lorsque vous utilisez `--worktree`.

Pour la coordination automatisée des sessions parallèles avec des tâches partagées et la messagerie, consultez [équipes d'agents](#).

Recevez une notification quand Claude a besoin de votre attention

Lorsque vous lancez une tâche longue et que vous basculez vers une autre fenêtre, vous pouvez configurer des notifications de bureau afin de savoir quand Claude se termine ou a besoin de votre entrée. Cela utilise l'événement de hook **Notification** [hook event](#), qui se déclenche chaque fois que Claude attend une permission, est inactif et prêt pour une nouvelle invite, ou complète l'authentification.

Step 1: Ouvrez le menu des hooks

Tapez `/hooks` et sélectionnez **Notification** dans la liste des événements.

Step 2: Configurez le matcher

Sélectionnez **+ Match all (no filter)** pour se déclencher sur tous les types de notifications. Pour notifier uniquement les événements spécifiques, sélectionnez **+ Add new matcher...** et entrez l'une de ces valeurs :

Matcher	Se déclenche quand
<code>permission_prompt</code>	Claude a besoin que vous approuviez une utilisation d'outil
<code>idle_prompt</code>	Claude a terminé et attend votre prochaine invite
<code>auth_success</code>	L'authentification se termine
<code>elicitation_dialog</code>	Claude vous pose une question

Step 3: Ajoutez votre commande de notification

Sélectionnez **+ Add new hook...** et entrez la commande pour votre système d'exploitation :

macOS

Utilise `osascript` pour déclencher une notification macOS native via AppleScript :

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

Linux

Utilise `notify-send`, qui est préinstallé sur la plupart des bureaux Linux avec un démon de notification :

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

Windows (PowerShell)

Utilise PowerShell pour afficher une boîte de message native via .NET Windows Forms :

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

Step 4: Enregistrez dans les paramètres utilisateur

Sélectionnez **User settings** pour appliquer la notification sur tous vos projets.

Pour la procédure complète avec des exemples de configuration JSON, consultez [Automatiser les flux de travail avec les hooks](#). Pour le schéma d'événement complet et les types de notifications, consultez la [référence Notification](#).

Utiliser Claude comme un utilitaire de style unix

Ajouter Claude à votre processus de vérification

Supposons que vous souhaitiez utiliser Claude Code comme linter ou examinateur de code.

Ajoutez Claude à votre script de construction :

```
// 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:

Conseils :

- Utilisez Claude pour l'examen automatisé du code dans votre pipeline CI/CD
- Personnalisez l'invite pour vérifier les problèmes spécifiques pertinents pour votre projet
- Envisagez de créer plusieurs scripts pour différents types de vérification

Tuyau entrant, tuyau sortant

Supposons que vous souhaitiez canaliser les données dans Claude et récupérer les données dans un format structuré.

Canalisez les données via Claude :

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

Tip:

Conseils :

- Utilisez les tuyaux pour intégrer Claude dans les scripts shell existants
- Combinez avec d'autres outils Unix pour des flux de travail puissants
- Envisagez d'utiliser `–output-format` pour une sortie structurée

Contrôler le format de sortie

Supposons que vous ayez besoin de la sortie de Claude dans un format spécifique, en particulier lors de l'intégration de Claude Code dans des scripts ou d'autres outils.

Step 1: Utilisez le format texte (par défaut)

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

Cela génère uniquement la réponse en texte brut de Claude (comportement par défaut).

Step 2: Utilisez le format JSON

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

Cela génère un tableau JSON de messages avec des métadonnées incluant le coût et la durée.

Step 3: Utilisez le format JSON en streaming

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

Cela génère une série d'objets JSON en temps réel au fur et à mesure que Claude traite la demande. Chaque message est un objet JSON valide, mais la sortie entière n'est pas un JSON valide s'il est concaténé.

Tip:

Conseils :

- Utilisez `--output-format text` pour les intégrations simples où vous avez juste besoin de la réponse de Claude
- Utilisez `--output-format json` quand vous avez besoin du journal de conversation complet
- Utilisez `--output-format stream-json` pour la sortie en temps réel de chaque tour de conversation

Demandez à Claude ses capacités

Claude a un accès intégré à sa documentation et peut répondre à des questions sur ses propres fonctionnalités et limitations.

Exemples de questions

```
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 fournit des réponses basées sur la documentation à ces questions. Pour des exemples exécutables et des démonstrations pratiques, consultez les sections de flux de travail spécifiques ci-dessus.

Tip:

Conseils :

- Claude a toujours accès à la dernière documentation de Claude Code, quelle que soit la version que vous utilisez
- Posez des questions spécifiques pour obtenir des réponses détaillées
- Claude peut expliquer les fonctionnalités complexes comme l'intégration MCP, les configurations d'entreprise et les flux de travail avancés

Étapes suivantes

- [Bonnes pratiques](#) Modèles pour tirer le meilleur parti de Claude Code
- [Comment fonctionne Claude Code](#) Comprendre la boucle agentique et la gestion du contexte
- [Étendre Claude Code](#) Ajouter des skills, des hooks, MCP, des subagents et des plugins
- [Implémentation de référence](#) Clonez notre implémentation de référence du conteneur de développement

Part 3: Commands & Reference

Référence CLI

Référence complète pour l'interface de ligne de commande Claude Code, incluant les commandes et les drapeaux.

Commandes CLI

Vous pouvez démarrer des sessions, traiter du contenu, reprendre des conversations et gérer les mises à jour avec ces commandes :

Commande	Description	Exemple
<code>claude</code>	Démarrer une session interactive	<code>claude</code>
<code>claude "query"</code>	Démarrer une session interactive avec une invite initiale	<code>claude "explain this project"</code>
<code>claude -p "query"</code>	Interroger via SDK, puis quitter	<code>claude -p "explain this function"</code>
<code>cat file claude -p "query"</code>	Traiter le contenu canalisé	<code>cat logs.txt claude -p "explain"</code>
<code>claude -c</code>	Continuer la conversation la plus récente dans le répertoire courant	<code>claude -c</code>
<code>claude -c -p "query"</code>	Continuer via SDK	<code>claude -c -p "Check for type errors"</code>
<code>claude -r "<session>" "query"</code>	Reprendre une session par ID ou nom	<code>claude -r "auth-refactor" "Finish this PR"</code>
<code>claude update</code>	Mettre à jour vers la dernière version	<code>claude update</code>
<code>claude auth login</code>	Se connecter à votre compte Anthropic. Utilisez <code>--email</code> pour pré-remplir votre adresse e-mail et <code>--sso</code> pour forcer l'authentification SSO	<code>claude auth login --email user@example.com --sso</code>

Commande	Description	Exemple
<code>claude auth logout</code>	Se déconnecter de votre compte Anthropic	<code>claude auth logout</code>
<code>claude auth status</code>	Afficher l'état de l'authentification en JSON. Utilisez <code>--text</code> pour une sortie lisible par l'homme. Quitte avec le code 0 si connecté, 1 sinon	<code>claude auth status</code>
<code>claude agents</code>	Lister tous les subagents configurés, groupés par source	<code>claude agents</code>
<code>claude mcp</code>	Configurer les serveurs Model Context Protocol (MCP)	Voir la documentation Claude Code MCP .
<code>claude remote-control</code>	Démarrer une session Remote Control pour contrôler Claude Code depuis Claude.ai ou l'application Claude tout en exécutant localement. Voir Remote Control pour les drapeaux	<code>claude remote-control</code>

Drapeaux CLI

Personnalisez le comportement de Claude Code avec ces drapeaux de ligne de commande :

Drapeau	Description	Exemple
<code>--add-dir</code>	Ajouter des répertoires de travail supplémentaires pour que Claude y accède (valide que chaque chemin existe en tant que répertoire)	<code>claude --add-dir ../apps ../lib</code>
<code>--agent</code>	Spécifier un agent pour la session actuelle (remplace le paramètre <code>agent</code>)	<code>claude --agent my-custom-agent</code>
<code>--agents</code>	Définir des subagents personnalisés dynamiquement via JSON (voir ci-dessous pour le format)	<code>claude --agents '{"reviewer":{"description":"Review code","prompt":"You are a code reviewer"}}'</code>

Drapeau	Description	Exemple
<code>--allow-dangerously-skip-permissions</code>	Activer le contournement des permissions en tant qu'option sans l'activer immédiatement. Permet de composer avec <code>--permission-mode</code> (à utiliser avec prudence)	<code>claude --permission-mode plan --allow-dangerously-skip-permissions</code>
<code>--allowedTools</code>	Outils qui s'exécutent sans demander la permission. Voir syntaxe des règles de permission pour la correspondance de motifs. Pour restreindre les outils disponibles, utilisez <code>--tools</code> à la place	<code>"Bash(git log *)"</code> <code>"Bash(git diff *)"</code> <code>"Read"</code>
<code>--append-system-prompt</code>	Ajouter du texte personnalisé à la fin de l'invite système par défaut	<code>claude --append-system-prompt "Always use TypeScript"</code>
<code>--append-system-prompt-file</code>	Charger du texte d'invite système supplémentaire à partir d'un fichier et l'ajouter à l'invite par défaut	<code>claude --append-system-prompt-file ./extra-rules.txt</code>
<code>--betas</code>	En-têtes bêta à inclure dans les requêtes API (utilisateurs de clé API uniquement)	<code>claude --betas interleaved-thinking</code>
<code>--chrome</code>	Activer l' intégration du navigateur Chrome pour l'automatisation web et les tests	<code>claude --chrome</code>
<code>--continue</code> , <code>-c</code>	Charger la conversation la plus récente dans le répertoire courant	<code>claude --continue</code>
<code>--dangerously-skip-permissions</code>	Ignorer toutes les invites de permission (à utiliser avec prudence)	<code>claude --dangerously-skip-permissions</code>
<code>--debug</code>	Activer le mode débogage avec filtrage de catégorie optionnel (par exemple, <code>"api,hooks"</code> ou <code>"!statsig,!file"</code>)	<code>claude --debug "api,mcp"</code>
<code>--disable-slash-commands</code>	Désactiver tous les skills et commandes pour cette session	<code>claude --disable-slash-commands</code>

Drapeau	Description	Exemple
<code>--disallowedTools</code>	Outils qui sont supprimés du contexte du modèle et ne peuvent pas être utilisés	<code>"Bash(git log *)"</code> <code>"Bash(git diff *)"</code> <code>"Edit"</code>
<code>--fallback-model</code>	Activer le basculement automatique vers le modèle spécifié lorsque le modèle par défaut est surchargé (mode impression uniquement)	<code>claude -p --fallback-model sonnet "query"</code>
<code>--fork-session</code>	Lors de la reprise, créer un nouvel ID de session au lieu de réutiliser l'original (à utiliser avec <code>--resume</code> ou <code>--continue</code>)	<code>claude --resume abc123 --fork-session</code>
<code>--from-pr</code>	Reprendre les sessions liées à une PR GitHub spécifique. Accepte un numéro de PR ou une URL. Les sessions sont automatiquement liées lors de la création via <code>gh pr create</code>	<code>claude --from-pr 123</code>
<code>--ide</code>	Se connecter automatiquement à l'IDE au démarrage s'il y a exactement un IDE valide disponible	<code>claude --ide</code>
<code>--init</code>	Exécuter les hooks d'initialisation et démarrer le mode interactif	<code>claude --init</code>
<code>--init-only</code>	Exécuter les hooks d'initialisation et quitter (pas de session interactive)	<code>claude --init-only</code>
<code>--include-partial-messages</code>	Inclure les événements de streaming partiels dans la sortie (nécessite <code>--print</code> et <code>--output-format=stream-json</code>)	<code>claude -p --output-format stream-json --include-partial-messages "query"</code>
<code>--input-format</code>	Spécifier le format d'entrée pour le mode impression (options : <code>text</code> , <code>stream-json</code>)	<code>claude -p --output-format json --input-format stream-json</code>

Drapeau	Description	Exemple
<code>--json-schema</code>	Obtenir une sortie JSON validée correspondant à un JSON Schema après que l'agent ait terminé son flux de travail (mode impression unique-ment, voir structured outputs)	<code>claude -p --json-schema '{"type":"object","properties":{"...}}" "query"</code>
<code>--maintenance</code>	Exécuter les hooks de maintenance et quitter	<code>claude --maintenance</code>
<code>--max-budget-usd</code>	Montant en dollars maximum à dépenser pour les appels API avant d'arrêter (mode impression unique-ment)	<code>claude -p --max-budget-usd 5.00 "query"</code>
<code>--max-turns</code>	Limiter le nombre de tours agentiques (mode impression unique-ment). Quitte avec une erreur lorsque la limite est atteinte. Pas de limite par défaut	<code>claude -p --max-turns 3 "query"</code>
<code>--mcp-config</code>	Charger les serveurs MCP à partir de fichiers ou de chaînes JSON (séparés par des espaces)	<code>claude --mcp-config ./mcp.json</code>
<code>--model</code>	Définit le modèle pour la session actuelle avec un alias pour le dernier modèle (<code>sonnet</code> ou <code>opus</code>) ou le nom complet d'un modèle	<code>claude --model claude-sonnet-4-6</code>
<code>--no-chrome</code>	Désactiver l' intégration du navigateur Chrome pour cette session	<code>claude --no-chrome</code>
<code>--no-session-persistence</code>	Désactiver la persistance de session afin que les sessions ne soient pas enregistrées sur le disque et ne puissent pas être reprises (mode impression uniquement)	<code>claude -p --no-session-persistence "query"</code>
<code>--output-format</code>	Spécifier le format de sortie pour le mode impression (options : <code>text</code> , <code>json</code> , <code>stream-json</code>)	<code>claude -p "query" --output-format json</code>

Drapeau	Description	Exemple
<code>--permission-mode</code>	Commencer dans un mode de permission spécifié	<code>claude --permission-mode plan</code>
<code>--permission-prompt-tool</code>	Spécifier un outil MCP pour gérer les invites de permission en mode non interactif	<code>claude -p --permission-prompt-tool mcp_auth_tool "query"</code>
<code>--plugin-dir</code>	Charger les plugins à partir de répertoires pour cette session uniquement (répérable)	<code>claude --plugin-dir ./my-plugins</code>
<code>--print</code> , <code>-p</code>	Imprimer la réponse sans mode interactif (voir la documentation Agent SDK pour les détails d'utilisation programmatique)	<code>claude -p "query"</code>
<code>--remote</code>	Créer une nouvelle session web sur claude.ai avec la description de tâche fournie	<code>claude --remote "Fix the login bug"</code>
<code>--resume</code> , <code>-r</code>	Reprendre une session spécifique par ID ou nom, ou afficher un sélecteur interactif pour choisir une session	<code>claude --resume auth-refactor</code>
<code>--session-id</code>	Utiliser un ID de session spécifique pour la conversation (doit être un UUID valide)	<code>claude --session-id "550e8400-e29b-41d4-a716-446655440000"</code>
<code>--setting-sources</code>	Liste séparée par des virgules des sources de paramètres à charger (<code>user</code> , <code>project</code> , <code>local</code>)	<code>claude --setting-sources user,project</code>
<code>--settings</code>	Chemin vers un fichier JSON de paramètres ou une chaîne JSON pour charger des paramètres supplémentaires	<code>claude --settings ./settings.json</code>
<code>--strict-mcp-config</code>	Utiliser uniquement les serveurs MCP de <code>--mcp-config</code> , en ignorant toutes les autres configurations MCP	<code>claude --strict-mcp-config --mcp-config ./mcp.json</code>

Drapeau	Description	Exemple
<code>--system-prompt</code>	Remplacer l'invite système entière par du texte personnalisé	<code>claude --system-prompt "You are a Python expert"</code>
<code>--system-prompt-file</code>	Charger l'invite système à partir d'un fichier, remplaçant l'invite par défaut	<code>claude --system-prompt-file ./custom-prompt.txt</code>
<code>--teleport</code>	Reprendre une session web dans votre terminal local	<code>claude --teleport</code>
<code>--teammate-mode</code>	Définir comment les coéquipiers de l' équipe d'agents s'affichent : <code>auto</code> (par défaut), <code>in-process</code> , ou <code>tmux</code> . Voir configurer les équipes d'agents	<code>claude --teammate-mode in-process</code>
<code>--tools</code>	Restreindre les outils intégrés que Claude peut utiliser. Utilisez <code>""</code> pour désactiver tous, <code>"default"</code> pour tous, ou des noms d'outils comme <code>"Bash,Edit,Read"</code>	<code>claude --tools "Bash,Edit,Read"</code>
<code>--verbose</code>	Activer la journalisation détaillée, affiche la sortie complète tour par tour	<code>claude --verbose</code>
<code>--version</code> , <code>-v</code>	Afficher le numéro de version	<code>claude -v</code>
<code>--worktree</code> , <code>-w</code>	Démarrer Claude dans un git work-tree isolé à <code><repo>/claude/worktrees/<name></code> . Si aucun nom n'est donné, un est généré automatiquement	<code>claude -w feature-auth</code>

Tip:

Le drapeau `--output-format json` est particulièrement utile pour les scripts et l'automatisation, vous permettant d'analyser les réponses de Claude par programmation.

Format du drapeau agents

Le drapeau `--agents` accepte un objet JSON qui définit un ou plusieurs subagents personnalisés. Chaque subagent nécessite un nom unique (comme clé) et un objet de définition avec les champs suivants :

Champ	Requis	Description
<code>description</code>	Oui	Description en langage naturel du moment où le subagent doit être invoqué
<code>prompt</code>	Oui	L'invite système qui guide le comportement du subagent
<code>tools</code>	Non	Tableau des outils spécifiques que le subagent peut utiliser, par exemple <code>["Read", "Edit", "Bash"]</code> . S'il est omis, hérite de tous les outils. Supporte la syntaxe <code>Agent(agent_type)</code>
<code>disallowedTools</code>	Non	Tableau des noms d'outils à explicitement refuser pour ce subagent
<code>model</code>	Non	Alias de modèle à utiliser : <code>sonnet</code> , <code>opus</code> , <code>haiku</code> , ou <code>inherit</code> . S'il est omis, par défaut <code>inherit</code>
<code>skills</code>	Non	Tableau des noms de skill à précharger dans le contexte du subagent
<code>mcpServers</code>	Non	Tableau des serveurs MCP pour ce subagent. Chaque entrée est une chaîne de nom de serveur ou un objet <code>{name: config}</code>

Champ	Requis	Description
<code>maxTurns</code>	Non	Nombre maximum de tours agentiques avant que le subagent s'arrête

Exemple :

```
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."
  }
}'
```

Pour plus de détails sur la création et l'utilisation de subagents, voir la [documentation des subagents](#).

Drapeaux d’invite système

Claude Code fournit quatre drapeaux pour personnaliser l’invite système. Les quatre fonctionnent à la fois en mode interactif et non interactif.

Drapeau	Comportement	Cas d’utilisation
<code>--system-prompt</code>	Remplace l’invite par défaut entière	Contrôle complet sur le comportement et les ins-tructions de Claude
<code>--system-prompt-file</code>	Remplace par le contenu du fichier	Charger les invites à partir de fichiers pour la repro-ductibilité et le contrôle de version

Drapeau	Comportement	Cas d'utilisation
<code>--append-system-prompt</code>	Ajoute à l'invite par défaut	Ajouter des instructions spécifiques tout en conservant le comportement par défaut de Claude Code
<code>--append-system-prompt-file</code>	Ajoute le contenu du fichier à l'invite par défaut	Charger des instructions supplémentaires à partir de fichiers tout en conservant les valeurs par défaut

Quand utiliser chacun :

- `--system-prompt` : utilisez lorsque vous avez besoin d'un contrôle complet sur l'invite système de Claude. Cela supprime toutes les instructions Claude Code par défaut, vous donnant une ardoise vierge.

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

- `--system-prompt-file` : utilisez lorsque vous souhaitez charger une invite personnalisée à partir d'un fichier, utile pour la cohérence d'équipe ou les modèles d'invite contrôlés par version.

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

- `--append-system-prompt` : utilisez lorsque vous souhaitez ajouter des instructions spécifiques tout en conservant les capacités par défaut de Claude Code. C'est l'option la plus sûre pour la plupart des cas d'utilisation.

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

- `--append-system-prompt-file` : utilisez lorsque vous souhaitez ajouter des instructions à partir d'un fichier tout en conservant les valeurs par défaut de Claude Code. Utile pour les ajouts contrôlés par version.

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

`--system-prompt` et `--system-prompt-file` s'excluent mutuellement. Les drapeaux d'ajout peuvent être utilisés ensemble avec l'un ou l'autre drapeau de remplacement.

Pour la plupart des cas d'utilisation, `--append-system-prompt` ou `--append-system-prompt-file` est recommandé car ils préservent les capacités intégrées de Claude Code tout en ajoutant vos exigences personnalisées. Utilisez `--system-prompt` ou `--system-prompt-file` uniquement lorsque vous avez besoin d'un contrôle complet sur l'invite système.

Voir aussi

- [Extension Chrome](#) - Automatisation du navigateur et tests web
- [Mode interactif](#) - Raccourcis, modes d'entrée et fonctionnalités interactives
- [Guide de démarrage rapide](#) - Prise en main de Claude Code
- [Flux de travail courants](#) - Flux de travail et modèles avancés
- [Paramètres](#) - Options de configuration
- [Documentation Agent SDK](#) - Utilisation programmatique et intégrations

Personnaliser les raccourcis clavier

Personnalisez les raccourcis clavier dans Claude Code avec un fichier de configuration des liaisons de touches.

Note:

Les raccourcis clavier personnalisables nécessitent Claude Code v2.1.18 ou version ultérieure. Vérifiez votre version avec `claude --version`.

Claude Code prend en charge les raccourcis clavier personnalisables. Exécutez `/keybindings` pour créer ou ouvrir votre fichier de configuration à `~/.claude/keybindings.json`.

Fichier de configuration

Le fichier de configuration des liaisons de touches est un objet avec un tableau `bindings`. Chaque bloc spécifie un contexte et une carte des séquences de touches aux actions.

Note:

Les modifications du fichier `keybindings` sont automatiquement détectées et appliquées sans redémarrer Claude Code.

Champ	Description
<code>\$schema</code>	URL du schéma JSON optionnel pour l'autocomplétion de l'éditeur
<code>\$docs</code>	URL de documentation optionnelle
<code>bindings</code>	Tableau de blocs de liaison par contexte

Cet exemple lie `Ctrl+E` pour ouvrir un éditeur externe dans le contexte de chat, et délié `Ctrl+U` :

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

Contextes

Chaque bloc de liaison spécifie un **contexte** où les liaisons s'appliquent :

Contexte	Description
Global	S'applique partout dans l'application
Chat	Zone de saisie de chat principale
Autocomplete	Le menu d'autocomplétion est ouvert
Settings	Menu des paramètres (fermeture par échappement uniquement)
Confirmation	Dialogues de permission et de confirmation
Tabs	Composants de navigation par onglets
Help	Le menu d'aide est visible
Transcript	Visionneuse de transcription
HistorySearch	Mode de recherche d'historique (Ctrl+R)
Task	Une tâche de fond est en cours d'exécution
ThemePicker	Dialogue du sélecteur de thème
Attachments	Navigation de la barre d'image/pièce jointe

Contexte	Description
Footer	Navigation de l'indicateur de pied de page (tâches, équipes, diff)
MessageSelector	Sélection de message du dialogue de rembobinage et de résumé
DiffDialog	Navigation de la visionneuse de diff
ModelPicker	Niveau d'effort du sélecteur de modèle
Select	Composants génériques de sélection/liste
Plugin	Dialogue du plugin (parcourir, découvrir, gérer)

Actions disponibles

Les actions suivent un format `namespace:action`, tel que `chat:submit` pour envoyer un message ou `app:toggleTodos` pour afficher la liste des tâches. Chaque contexte a des actions spécifiques disponibles.

Actions d'application

Actions disponibles dans le contexte `Global` :

Action	Par défaut	Description
<code>app:interrupt</code>	Ctrl+C	Annuler l'opération en cours
<code>app:exit</code>	Ctrl+D	Quitter Claude Code
<code>app:toggleTodos</code>	Ctrl+T	Basculer la visibilité de la liste des tâches
<code>app:toggleTranscript</code>	Ctrl+O	Basculer la transcription détaillée

Actions d'historique

Actions pour naviguer dans l'historique des commandes :

Action	Par défaut	Description
<code>history:search</code>	Ctrl+R	Ouvrir la recherche d'historique
<code>history:previous</code>	Haut	Élément d'historique précédent

Action	Par défaut	Description
<code>history:next</code>	Bas	Élément d'historique suivant

Actions de chat

Actions disponibles dans le contexte `Chat` :

Action	Par défaut	Description
<code>chat:cancel</code>	Échappement	Annuler l'entrée actuelle
<code>chat:cycleMode</code>	Maj+Tab*	Cycler les modes de permission
<code>chat:modelPicker</code>	Cmd+P / Meta+P	Ouvrir le sélecteur de modèle
<code>chat:thinkingToggle</code>	Cmd+T / Meta+T	Basculer la réflexion étendue
<code>chat:submit</code>	Entrée	Soumettre le message
<code>chat:undo</code>	Ctrl+_	Annuler la dernière action
<code>chat:externalEditor</code>	Ctrl+G	Ouvrir dans un éditeur externe
<code>chat:stash</code>	Ctrl+S	Mettre en cache l'invite actuelle
<code>chat:imagePaste</code>	Ctrl+V (Alt+V sous Windows)	Coller une image

*Sous Windows sans mode VT (Node <24.2.0/<22.17.0, Bun <1.2.23), la valeur par défaut est Meta+M.

Actions d'autocomplétion

Actions disponibles dans le contexte `Autocomplete` :

Action	Par défaut	Description
<code>autocomplete:accept</code>	Tab	Accepter la suggestion
<code>autocomplete:dismiss</code>	Échappement	Fermer le menu
<code>autocomplete:previous</code>	Haut	Suggestion précédente

Action	Par défaut	Description
<code>autocomplete:next</code>	Bas	Suggestion suivante

Actions de confirmation

Actions disponibles dans le contexte `Confirmation` :

Action	Par défaut	Description
<code>confirm:yes</code>	Y, Entrée	Confirmer l'action
<code>confirm:no</code>	N, Échappement	Refuser l'action
<code>confirm:previous</code>	Haut	Option précédente
<code>confirm:next</code>	Bas	Option suivante
<code>confirm:nextField</code>	Tab	Champ suivant
<code>confirm:previousField</code>	(non lié)	Champ précédent
<code>confirm:cycleMode</code>	Maj+Tab	Cycler les modes de permission
<code>confirm:toggleExplanation</code>	Ctrl+E	Basculer l'explication de permission

Actions de permission

Actions disponibles dans le contexte `Confirmation` pour les dialogues de permission :

Action	Par défaut	Description
<code>permission:toggleDebug</code>	Ctrl+D	Basculer les informations de débogage de permission

Actions de transcription

Actions disponibles dans le contexte `Transcript` :

Action	Par défaut	Description
<code>transcript:toggleShowAll</code>	Ctrl+E	Basculer l'affichage de tout le contenu

Action	Par défaut	Description
<code>transcript:exit</code>	Ctrl+C, Échappement	Quitter la vue de transcription

Actions de recherche d'historique

Actions disponibles dans le contexte `HistorySearch` :

Action	Par défaut	Description
<code>historySearch:next</code>	Ctrl+R	Correspondance suivante
<code>historySearch:accept</code>	Échappement, Tab	Accepter la sélection
<code>historySearch:cancel</code>	Ctrl+C	Annuler la recherche
<code>historySearch:execute</code>	Entrée	Exécuter la commande sélectionnée

Actions de tâche

Actions disponibles dans le contexte `Task` :

Action	Par défaut	Description
<code>task:background</code>	Ctrl+B	Mettre la tâche actuelle en arrière-plan

Actions de thème

Actions disponibles dans le contexte `ThemePicker` :

Action	Par défaut	Description
<code>theme:toggleSyntaxHighlighting</code>	Ctrl+T	Basculer la coloration syntaxique

Actions d'aide

Actions disponibles dans le contexte `Help` :

Action	Par défaut	Description
<code>help:dismiss</code>	Échappement	Fermer le menu d'aide

Actions d’onglets

Actions disponibles dans le contexte `Tabs` :

Action	Par défaut	Description
<code>tabs:next</code>	Tab, Droite	Onglet suivant
<code>tabs:previous</code>	Maj+Tab, Gauche	Onglet précédent

Actions de pièces jointes

Actions disponibles dans le contexte `Attachments` :

Action	Par défaut	Description
<code>attachments:next</code>	Droite	Pièce jointe suivante
<code>attachments:previous</code>	Gauche	Pièce jointe précédente
<code>attachments:remove</code>	Retour arrière, Supprimer	Supprimer la pièce jointe sélectionnée
<code>attachments:exit</code>	Bas, Échappement	Quitter la barre de pièces jointes

Actions de pied de page

Actions disponibles dans le contexte `Footer` :

Action	Par défaut	Description
<code>footer:next</code>	Droite	Élément de pied de page suivant
<code>footer:previous</code>	Gauche	Élément de pied de page précédent
<code>footer:openSelected</code>	Entrée	Ouvrir l’élément de pied de page sélectionné
<code>footer:clearSelection</code>	Échappement	Effacer la sélection du pied de page

Actions du sélecteur de message

Actions disponibles dans le contexte `MessageSelector` :

Action	Par défaut	Description
<code>messageSelector:up</code>	Haut, K, Ctrl+P	Déplacer vers le haut dans la liste
<code>messageSelector:down</code>	Bas, J, Ctrl+N	Déplacer vers le bas dans la liste
<code>messageSelector:top</code>	Ctrl+Haut, Maj+Haut, Meta+Haut, Maj+K	Sauter au début
<code>messageSelector:bottom</code>	Ctrl+Bas, Maj+Bas, Meta+Bas, Maj+J	Sauter à la fin
<code>messageSelector:select</code>	Entrée	Sélectionner le message

Actions de diff

Actions disponibles dans le contexte `DiffDialog` :

Action	Par défaut	Description
<code>diff:dismiss</code>	Échappement	Fermer la visionneuse de diff
<code>diff:previousSource</code>	Gauche	Source de diff précédente
<code>diff:nextSource</code>	Droite	Source de diff suivante
<code>diff:previousFile</code>	Haut	Fichier précédent dans le diff
<code>diff:nextFile</code>	Bas	Fichier suivant dans le diff
<code>diff:viewDetails</code>	Entrée	Afficher les détails du diff
<code>diff:back</code>	(spécifique au contexte)	Revenir en arrière dans la visionneuse de diff

Actions du sélecteur de modèle

Actions disponibles dans le contexte `ModelPicker` :

Action	Par défaut	Description
<code>modelPicker:decreaseEffort</code>	Gauche	Diminuer le niveau d'effort
<code>modelPicker:increaseEffort</code>	Droite	Augmenter le niveau d'effort

Actions de sélection

Actions disponibles dans le contexte `Select` :

Action	Par défaut	Description
<code>select:next</code>	Bas, J, Ctrl+N	Option suivante
<code>select:previous</code>	Haut, K, Ctrl+P	Option précédente
<code>select:accept</code>	Entrée	Accepter la sélection
<code>select:cancel</code>	Échappement	Annuler la sélection

Actions de plugin

Actions disponibles dans le contexte `Plugin` :

Action	Par défaut	Description
<code>plugin:toggle</code>	Espace	Basculer la sélection du plugin
<code>plugin:install</code>	I	Installer les plugins sélectionnés

Actions des paramètres

Actions disponibles dans le contexte `Settings` :

Action	Par défaut	Description
<code>settings:search</code>	/	Entrer en mode de recherche
<code>settings:retry</code>	R	Réessayer de charger les données d'utilisation (en cas d'erreur)

Syntaxe des séquences de touches

Modificateurs

Utilisez les touches de modification avec le séparateur `+` :

- `ctrl` ou `control` - Touche Contrôle
- `alt`, `opt`, ou `option` - Touche Alt/Option
- `shift` - Touche Maj
- `meta`, `cmd`, ou `command` - Touche Meta/Commande

Par exemple :

<code>ctrl+k</code>	Touche unique avec modificateur
<code>shift+tab</code>	Maj + Tab
<code>meta+p</code>	Commande/Meta + P
<code>ctrl+shift+c</code>	Plusieurs modificateurs

Lettres majuscules

Une lettre majuscule autonome implique Maj. Par exemple, `K` est équivalent à `shift+k`. Ceci est utile pour les liaisons de style vim où les touches majuscules et minuscules ont des significations différentes.

Les lettres majuscules avec des modificateurs (par exemple, `ctrl+K`) sont traitées comme stylistiques et n'impliquent **pas** Maj — `ctrl+K` est identique à `ctrl+k`.

Accords

Les accords sont des séquences de touches séparées par des espaces :

`ctrl+k ctrl+s` Appuyez sur Ctrl+K, relâchez, puis Ctrl+S

Touches spéciales

- `escape` ou `esc` - Touche Échappement
- `enter` ou `return` - Touche Entrée
- `tab` - Touche Tab
- `space` - Barre d'espace
- `up`, `down`, `left`, `right` - Touches fléchées
- `backspace`, `delete` - Touches de suppression

Délier les raccourcis par défaut

Définissez une action sur `null` pour délier un raccourci par défaut :

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

Raccourcis réservés

Ces raccourcis ne peuvent pas être reliés :

Raccourci	Raison
Ctrl+C	Interruption/annulation codée en dur
Ctrl+D	Sortie codée en dur

Conflits de terminal

Certains raccourcis peuvent entrer en conflit avec les multiplexeurs de terminal :

Raccourci	Conflit
Ctrl+B	Préfixe tmux (appuyez deux fois pour envoyer)
Ctrl+A	Préfixe GNU screen
Ctrl+Z	Suspension de processus Unix (SIGTSTP)

Interaction du mode Vim

Lorsque le mode vim est activé (`/vim`), les liaisons de touches et le mode vim fonctionnent indépendamment :

- **Mode Vim** gère l'entrée au niveau de la saisie de texte (mouvement du curseur, modes, motions)
- **Liaisons de touches** gèrent les actions au niveau du composant (basculer les tâches, soumettre, etc.)
- La touche Échappement en mode vim bascule INSERT en mode NORMAL ; elle ne déclenche pas `chat:cancel`
- La plupart des raccourcis Ctrl+touche passent par le mode vim au système de liaison de touches
- En mode NORMAL vim, `?` affiche le menu d'aide (comportement vim)

Validation

Claude Code valide vos liaisons de touches et affiche des avertissements pour :

- Erreurs d'analyse (JSON invalide ou structure)
- Noms de contexte invalides
- Conflits de raccourcis réservés
- Conflits de multiplexeur de terminal
- Liaisons en double dans le même contexte

Exécutez `/doctor` pour voir les avertissements de liaison de touches.

Part 4: Configuration

Configurer les autorisations

Contrôlez ce que Claude Code peut accéder et faire avec des règles d'autorisation granulaires, des modes et des politiques gérées.

Claude Code prend en charge les autorisations granulaires afin que vous puissiez spécifier exactement ce que l'agent est autorisé à faire et ce qu'il ne peut pas faire. Les paramètres d'autorisation peuvent être archivés dans le contrôle de version et distribués à tous les développeurs de votre organisation, ainsi que personnalisés par les développeurs individuels.

Système d'autorisation

Claude Code utilise un système d'autorisation à plusieurs niveaux pour équilibrer la puissance et la sécurité :

Type d'outil	Exemple	Approbation requise	Comportement « Oui, ne pas demander à nouveau »
Lecture seule	Lectures de fichiers, Grep	Non	S/O
Commandes Bash	Exécution shell	Oui	Permanent par répertoire de projet et commande
Modification de fichier	Édition/écriture de fichiers	Oui	Jusqu'à la fin de la session

Gérer les autorisations

Vous pouvez afficher et gérer les autorisations d'outils de Claude Code avec `/permissions`. Cette interface utilisateur répertorie toutes les règles d'autorisation et le fichier `settings.json` dont elles proviennent.

- Les règles **Allow** permettent à Claude Code d'utiliser l'outil spécifié sans approbation manuelle.
- Les règles **Ask** demandent une confirmation chaque fois que Claude Code essaie d'utiliser l'outil spécifié.
- Les règles **Deny** empêchent Claude Code d'utiliser l'outil spécifié.

Les règles sont évaluées dans l'ordre : **deny** -> **ask** -> **allow**. La première règle correspondante gagne, donc les règles de refus ont toujours la priorité.

Modes d'autorisation

Claude Code prend en charge plusieurs modes d'autorisation qui contrôlent la façon dont les outils sont approuvés. Définissez le `defaultMode` dans vos [fichiers de paramètres](#) :

Mode	Description
<code>default</code>	Comportement standard : demande une autorisation à la première utilisation de chaque outil
<code>acceptEdits</code>	Accepte automatiquement les autorisations d'édition de fichiers pour la session
<code>plan</code>	Plan Mode : Claude peut analyser mais pas modifier les fichiers ou exécuter les commandes
<code>dontAsk</code>	Refuse automatiquement les outils sauf s'ils sont pré-approuvés via <code>/permissions</code> ou les règles <code>permissions.allow</code>
<code>bypassPermissions</code>	Ignore tous les invites d'autorisation (nécessite un environnement sûr, voir l'avertissement ci-dessous)

Warning:

Le mode `bypassPermissions` désactive tous les contrôles d'autorisation. Utilisez-le uniquement dans des environnements isolés comme les conteneurs ou les machines virtuelles où Claude Code ne peut pas causer de dommages. Les administrateurs peuvent empêcher ce mode en définissant `disableBypassPermissionsMode` sur `"disable"` dans les [paramètres gérés](#).

Syntaxe des règles d'autorisation

Les règles d'autorisation suivent le format `Tool` ou `Tool(specifier)`.

Correspondre à tous les usages d'un outil

Pour correspondre à tous les usages d'un outil, utilisez simplement le nom de l'outil sans parenthèses :

Règle	Effet
<code>Bash</code>	Correspond à toutes les commandes Bash
<code>WebFetch</code>	Correspond à toutes les demandes de récupération web
<code>Read</code>	Correspond à toutes les lectures de fichiers

`Bash(*)` est équivalent à `Bash` et correspond à toutes les commandes Bash.

Utiliser des spécificateurs pour un contrôle granulaire

Ajoutez un spécificateur entre parenthèses pour correspondre à des usages d'outils spécifiques :

Règle	Effet
<code>Bash(npm run build)</code>	Correspond à la commande exacte <code>npm run build</code>
<code>Read(./env)</code>	Correspond à la lecture du fichier <code>.env</code> dans le répertoire courant
<code>WebFetch(domain:example.com)</code>	Correspond aux demandes de récupération vers <code>example.com</code>

Modèles de caractères génériques

Les règles Bash prennent en charge les modèles glob avec `*`. Les caractères génériques peuvent apparaître à n'importe quelle position dans la commande. Cette configuration permet les commandes npm et git commit tout en bloquant git push :

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

L'espace avant `*` est important : `Bash(ls *)` correspond à `ls -la` mais pas à `ls of`, tandis que `Bash(ls*)` correspond aux deux. La syntaxe du suffixe `:*` hérité est équivalente à `*` mais est dépréciée.

Règles d'autorisation spécifiques aux outils

Bash

Les règles d'autorisation Bash prennent en charge la correspondance de caractères génériques avec `*`. Les caractères génériques peuvent apparaître à n'importe quelle position dans la commande, y compris au début, au milieu ou à la fin :

- `Bash(npm run build)` correspond à la commande Bash exacte `npm run build`
- `Bash(npm run test *)` correspond aux commandes Bash commençant par `npm run test`
- `Bash(npm *)` correspond à toute commande commençant par `npm`
- `Bash(* install)` correspond à toute commande se terminant par `install`
- `Bash(git * main)` correspond à des commandes comme `git checkout main`, `git merge main`

Lorsque `*` apparaît à la fin avec un espace avant (comme `Bash(ls *)`), il applique une limite de mot, exigeant que le préfixe soit suivi d'un espace ou de la fin de la chaîne. Par exemple, `Bash(ls *)` correspond à `ls -la` mais pas à `ls of`. En contraste, `Bash(ls*)` sans espace correspond aux deux `ls -la` et `ls of` car il n'y a pas de contrainte de limite de mot.

Tip:

Claude Code est conscient des opérateurs shell (comme `&&`) donc une règle de correspondance de préfixe comme `Bash(safe-cmd *)` ne lui donnera pas la permission d'exécuter la commande `safe-cmd && other-cmd`.

Warning:

Les modèles d'autorisation Bash qui tentent de contraindre les arguments de commande sont fragiles. Par exemple, `Bash(curl http://github.com/ *)` a l'intention de restreindre curl aux URL GitHub, mais ne correspondra pas aux variations comme :

- Options avant l'URL : `curl -X GET http://github.com/...`
- Protocole différent : `curl https://github.com/...`
- Redirections : `curl -L http://bit.ly/xyz` (redirige vers github)
- Variables : `URL=http://github.com && curl $URL`
- Espaces supplémentaires : `curl http://github.com`

Pour un filtrage d'URL plus fiable, envisagez :

- **Restreindre les outils réseau Bash** : utilisez les règles de refus pour bloquer `curl`, `wget` et les commandes similaires, puis utilisez l'outil WebFetch avec l'autorisation `WebFetch(domain:github.com)` pour les domaines autorisés
- **Utiliser les hooks PreToolUse** : implémentez un hook qui valide les URL dans les commandes Bash et bloque les domaines non autorisés
- Instruire Claude Code sur vos modèles curl autorisés via CLAUDE.md

Notez que l'utilisation de WebFetch seul n'empêche pas l'accès au réseau. Si Bash est autorisé, Claude peut toujours utiliser `curl`, `wget` ou d'autres outils pour atteindre n'importe quelle URL.

Read et Edit

Les règles `Edit` s'appliquent à tous les outils intégrés qui éditent les fichiers. Claude fait un effort raisonnable pour appliquer les règles `Read` à tous les outils intégrés qui lisent les fichiers comme Grep et Glob.

Les règles Read et Edit suivent toutes deux la spécification [gitignore](#) avec quatre types de modèles distincts :

Modèle	Signification	Exemple	Correspond
<code>//path</code>	Chemin absolu à partir de la racine du système de fichiers	<code>Read(//Users/alice/secrets/**)</code>	<code>/Users/alice/secrets/**</code>
<code>~/path</code>	Chemin à partir du répertoire home	<code>Read(~/Documents/*.pdf)</code>	<code>/Users/alice/Documents/*.pdf</code>
<code>/path</code>	Chemin relatif à la racine du projet	<code>Edit(/src/**/*.ts)</code>	<code><racine du projet>/src/**/*.ts</code>
<code>path</code> ou <code>./path</code>	Chemin relatif au répertoire courant	<code>Read(*.env)</code>	<code><cwd>/*.env</code>

Warning:

Un modèle comme `/Users/alice/file` n'est PAS un chemin absolu. Il est relatif à la racine du projet. Utilisez `//Users/alice/file` pour les chemins absolus.

Exemples :

- `Edit(/docs/**)` : édite dans `<projet>/docs/` (PAS `/docs/` et PAS `<projet>/.claude/docs/`)
- `Read(~/.zshrc)` : lit le `.zshrc` de votre répertoire home
- `Edit(//tmp/scratch.txt)` : édite le chemin absolu `/tmp/scratch.txt`
- `Read(src/**)` : lit à partir de `<répertoire courant>/src/`

Note:

Dans les modèles gitignore, `*` correspond aux fichiers dans un seul répertoire tandis que `**` correspond récursivement dans les répertoires. Pour autoriser tous les accès aux fichiers, utilisez simplement le nom de l'outil sans parenthèses : `Read`, `Edit` ou `Write`.

WebFetch

- `WebFetch(domain:example.com)` correspond aux demandes de récupération vers `example.com`

MCP

- `mcp__puppeteer` correspond à tout outil fourni par le serveur `puppeteer` (nom configuré dans Claude Code)
- `mcp__puppeteer__*` syntaxe de caractère générique qui correspond également à tous les outils du serveur `puppeteer`
- `mcp__puppeteer__puppeteer_navigate` correspond à l'outil `puppeteer_navigate` fourni par le serveur `puppeteer`

Agent (subagents)

Utilisez les règles `Agent(AgentName)` pour contrôler quels [subagents](#) Claude peut utiliser :

- `Agent(Explore)` correspond au subagent Explore
- `Agent(Plan)` correspond au subagent Plan
- `Agent(my-custom-agent)` correspond à un subagent personnalisé nommé `my-custom-agent`

Ajoutez ces règles au tableau `deny` dans vos paramètres ou utilisez l'indicateur CLI `--disallowedTools` pour désactiver des agents spécifiques. Pour désactiver l'agent Explore :

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

Étendre les autorisations avec des hooks

Les [hooks Claude Code](#) fournissent un moyen d'enregistrer des commandes shell personnalisées pour effectuer l'évaluation des autorisations à l'exécution. Lorsque Claude Code effectue un appel d'outil, les hooks `PreToolUse` s'exécutent avant le système d'autorisation, et la sortie du hook peut déterminer s'il faut approuver ou refuser l'appel d'outil à la place du système d'autorisation.

Répertoires de travail

Par défaut, Claude a accès aux fichiers du répertoire où il a été lancé. Vous pouvez étendre cet accès :

- **Au démarrage** : utilisez l'argument CLI `--add-dir <path>`

- **Pendant la session** : utilisez la commande `/add-dir`
- **Configuration persistante** : ajoutez à `additionalDirectories` dans les [fichiers de paramètres](#)

Les fichiers dans les répertoires supplémentaires suivent les mêmes règles d'autorisation que le répertoire de travail d'origine : ils deviennent lisibles sans invites, et les autorisations d'édition de fichiers suivent le mode d'autorisation actuel.

Comment les autorisations interagissent avec le sandboxing

Les autorisations et le [sandboxing](#) sont des couches de sécurité complémentaires :

- **Les autorisations** contrôlent quels outils Claude Code peut utiliser et quels fichiers ou domaines il peut accéder. Elles s'appliquent à tous les outils (Bash, Read, Edit, WebFetch, MCP et autres).
- **Le sandboxing** fournit une application au niveau du système d'exploitation qui restreint l'accès du système de fichiers et du réseau de l'outil Bash. Il s'applique uniquement aux commandes Bash et à leurs processus enfants.

Utilisez les deux pour une défense en profondeur :

- Les règles de refus d'autorisation empêchent Claude d'essayer même d'accéder aux ressources restreintes
- Les restrictions de sandbox empêchent les commandes Bash d'atteindre les ressources en dehors des limites définies, même si une injection de prompt contourne la prise de décision de Claude
- Les restrictions du système de fichiers dans le sandbox utilisent les règles de refus Read et Edit, pas une configuration de sandbox séparée
- Les restrictions réseau combinent les règles d'autorisation WebFetch avec la liste `allowedDomains` du sandbox

Paramètres gérés

Pour les organisations qui ont besoin d'un contrôle centralisé sur la configuration de Claude Code, les administrateurs peuvent déployer des paramètres gérés qui ne peuvent pas être remplacés par les paramètres utilisateur ou projet. Ces paramètres de politique suivent le même format que les fichiers de paramètres réguliers et peuvent être livrés via des politiques MDM/au niveau du système d'exploitation, des fichiers de paramètres gérés ou des [paramètres gérés par le serveur](#). Voir [fichiers de paramètres](#) pour les mécanismes de livraison et les emplacements de fichiers.

Paramètres gérés uniquement

Certains paramètres ne sont efficaces que dans les paramètres gérés :

Paramètre	Description
<code>disableBypassPermissionsMode</code>	Définissez sur <code>"disable"</code> pour empêcher le mode <code>bypassPermissions</code> et l'indicateur <code>--dangerously-skip-permissions</code>
<code>allowManagedPermissionRulesOnly</code>	Lorsque <code>true</code> , empêche les paramètres utilisateur et projet de définir les règles d'autorisation <code>allow</code> , <code>ask</code> ou <code>deny</code> . Seules les règles dans les paramètres gérés s'appliquent
<code>allowManagedHooksOnly</code>	Lorsque <code>true</code> , empêche le chargement des hooks utilisateur, projet et plugin. Seuls les hooks gérés et les hooks SDK sont autorisés
<code>allowManagedMcpServersOnly</code>	Lorsque <code>true</code> , seuls les <code>allowedMcpServers</code> des paramètres gérés sont respectés. <code>deniedMcpServers</code> fusionne toujours à partir de toutes les sources. Voir Configuration MCP gérée
<code>blockedMarketplaces</code>	Liste de blocage des sources de marketplace. Les sources bloquées sont vérifiées avant le téléchargement, elles ne touchent donc jamais le système de fichiers. Voir restrictions de marketplace gérées
<code>sandbox.network.allowManagedDomainsOnly</code>	Lorsque <code>true</code> , seuls les <code>allowedDomains</code> et les règles d'autorisation <code>WebFetch(domain: ...)</code> des paramètres gérés sont respectés. Les domaines non autorisés sont bloqués automatiquement sans inviter l'utilisateur. Les domaines refusés fusionnent toujours à partir de toutes les sources
<code>strictKnownMarketplaces</code>	Contrôle quels marketplaces de plugins les utilisateurs peuvent ajouter. Voir restrictions de marketplace gérées

Paramètre	Description
<code>allow_remote_sessions</code>	Lorsque <code>true</code> , permet aux utilisateurs de démarrer Remote Control et les sessions web . Par défaut <code>true</code> . Définissez sur <code>false</code> pour empêcher l'accès aux sessions distantes

Précédence des paramètres

Les règles d'autorisation suivent la même [précédence des paramètres](#) que tous les autres paramètres de Claude Code :

1. **Paramètres gérés** : ne peuvent pas être remplacés par aucun autre niveau, y compris les arguments de ligne de commande
2. **Arguments de ligne de commande** : remplacements de session temporaires
3. **Paramètres de projet local** (`.claude/settings.local.json`)
4. **Paramètres de projet partagés** (`.claude/settings.json`)
5. **Paramètres utilisateur** (`~/.claude/settings.json`)

Si un outil est refusé à n'importe quel niveau, aucun autre niveau ne peut l'autoriser. Par exemple, un refus de paramètres gérés ne peut pas être remplacé par `--allowedTools`, et `--disallowedTools` peut ajouter des restrictions au-delà de ce que les paramètres gérés définissent.

Si une autorisation est autorisée dans les paramètres utilisateur mais refusée dans les paramètres de projet, le paramètre de projet prend la priorité et l'autorisation est bloquée.

Exemples de configurations

Ce [référentiel](#) inclut des configurations de paramètres de démarrage pour les scénarios de déploiement courants. Utilisez-les comme points de départ et ajustez-les pour répondre à vos besoins.

Voir aussi

- [Paramètres](#) : référence de configuration complète incluant le tableau des paramètres d'autorisation
- [Sandboxing](#) : isolation du système de fichiers et du réseau au niveau du système d'exploitation pour les commandes Bash
- [Authentification](#) : configurer l'accès utilisateur à Claude Code

- [Sécurité](#) : garanties de sécurité et meilleures pratiques
- [Hooks](#) : automatiser les flux de travail et étendre l'évaluation des autorisations

Comment Claude se souvient de votre projet

*Donnez à Claude des instructions persistantes avec les fichiers **CLAUDE.md**, et laissez Claude accumuler automatiquement les apprentissages avec la mémoire automatique.*

Chaque session Claude Code commence avec une fenêtre de contexte vierge. Deux mécanismes transportent les connaissances entre les sessions :

- **Fichiers **CLAUDE.md**** : instructions que vous écrivez pour donner à Claude un contexte persistant
- **Mémoire automatique** : notes que Claude écrit lui-même en fonction de vos corrections et préférences

Cette page couvre comment :

- [Écrire et organiser les fichiers **CLAUDE.md**](#)
- [Limiter les règles à des types de fichiers spécifiques](#) avec `.claude/rules/`
- [Configurer la mémoire automatique](#) pour que Claude prenne des notes automatiquement
- [Dépanner](#) quand les instructions ne sont pas suivies

CLAUDE.md vs mémoire automatique

Claude Code dispose de deux systèmes de mémoire complémentaires. Les deux sont chargés au début de chaque conversation. Claude les traite comme du contexte, pas comme une configuration appliquée. Plus vos instructions sont spécifiques et concises, plus Claude les suit régulièrement.

	Fichiers CLAUDE.md	Mémoire automatique
Qui l'écrit	Vous	Claude
Ce qu'il contient	Instructions et règles	Apprentissages et modèles
Portée	Projet, utilisateur ou organisation	Par arborescence de travail
Chargé dans	Chaque session	Chaque session (premières 200 lignes)

	Fichiers CLAUDE.md	Mémoire automatique
À utiliser pour	Normes de codage, flux de travail, architecture du projet	Commandes de compilation, insights de débogage, préférences que Claude découvre

Utilisez les fichiers CLAUDE.md quand vous voulez guider le comportement de Claude. La mémoire automatique permet à Claude d'apprendre de vos corrections sans effort manuel.

Les subagents peuvent également maintenir leur propre mémoire automatique. Voir [configuration des subagents](#) pour plus de détails.

Fichiers CLAUDE.md

Les fichiers CLAUDE.md sont des fichiers markdown qui donnent à Claude des instructions persistantes pour un projet, votre flux de travail personnel ou toute votre organisation. Vous écrivez ces fichiers en texte brut ; Claude les lit au début de chaque session.

Choisir où placer les fichiers CLAUDE.md

Les fichiers CLAUDE.md peuvent se trouver à plusieurs endroits, chacun avec une portée différente. Les emplacements plus spécifiques ont la priorité sur les plus larges.

Portée	Emplacement	Objectif	Exemples de cas d'usage	Partagé avec
Politique gérée	• macOS : <code>/Library/Application Support/ClaudeCode/CLAUDE.md</code> • Linux et WSL : <code>/etc/claude-code/CLAUDE.md</code> • Windows : <code>C:\Program Files\ClaudeCode\CLAUDE.md</code>	Instructions à l'échelle de l'organisation gérées par l'informatique/DevOps	Normes de codage de l'entreprise, politiques de sécurité, exigences de conformité	Tous les utilisateurs de l'organisation
Instructions du projet	<code>./CLAUDE.md</code> ou <code>./.claude/CLAUDE.md</code>	Instructions partagées par l'équipe pour le projet	Architecture du projet, normes de codage, flux de travail courants	Membres de l'équipe via le contrôle de source

Portée	Emplacement	Objectif	Exemples de cas d'usage	Partagé avec
Instructions utilisateur	<code>~/.claude/CLAUDE.md</code>	Préférences personnelles pour tous les projets	Préférences de style de code, raccourcis d'outils personnels	Juste vous (tous les projets)

Les fichiers CLAUDE.md dans la hiérarchie de répertoires au-dessus du répertoire de travail sont chargés en intégralité au lancement. Les fichiers CLAUDE.md dans les sous-répertoires se chargent à la demande quand Claude lit les fichiers de ces répertoires. Voir [Comment les fichiers CLAUDE.md se chargent](#) pour l'ordre de résolution complet.

Pour les grands projets, vous pouvez diviser les instructions en fichiers spécifiques à un sujet en utilisant [les règles du projet](#). Les règles vous permettent de limiter les instructions à des types de fichiers ou des sous-répertoires spécifiques.

Configurer un CLAUDE.md de projet

Un CLAUDE.md de projet peut être stocké dans `./CLAUDE.md` ou `./.claude/CLAUDE.md`. Créez ce fichier et ajoutez des instructions qui s'appliquent à quiconque travaille sur le projet : commandes de compilation et de test, normes de codage, décisions architecturales, conventions de nommage et flux de travail courants. Ces instructions sont partagées avec votre équipe via le contrôle de version, donc concentrez-vous sur les normes au niveau du projet plutôt que sur les préférences personnelles.

Tip:

Exécutez `/init` pour générer automatiquement un CLAUDE.md de démarrage. Claude analyse votre base de code et crée un fichier avec les commandes de compilation, les instructions de test et les conventions de projet qu'il découvre. Si un CLAUDE.md existe déjà, `/init` suggère des améliorations plutôt que de le remplacer. Affinez-le à partir de là avec les instructions que Claude ne découvrirait pas de lui-même.

Écrire des instructions efficaces

Les fichiers CLAUDE.md sont chargés dans la fenêtre de contexte au début de chaque session, consommant des tokens aux côtés de votre conversation. Parce qu'ils sont du contexte plutôt qu'une configuration appliquée, la façon dont vous écrivez les instructions affecte la fiabilité avec laquelle Claude les suit. Les instructions spécifiques, concises et bien structurées fonctionnent mieux.

Taille : visez moins de 200 lignes par fichier CLAUDE.md. Les fichiers plus longs consomment plus de contexte et réduisent l'adhérence. Si vos instructions deviennent trop grandes, divisez-les en utilisant [les imports](#) ou les fichiers `.claude/rules/`.

Structure : utilisez les en-têtes markdown et les puces pour regrouper les instructions connexes. Claude scanne la structure de la même manière que les lecteurs : les sections organisées sont plus faciles à suivre que les paragraphes denses.

Spécificité : écrivez des instructions suffisamment concrètes pour être vérifiables. Par exemple :

- « Utiliser l'indentation à 2 espaces » au lieu de « Formater le code correctement »
- « Exécuter `npm test` avant de valider » au lieu de « Testez vos modifications »
- « Les gestionnaires d'API se trouvent dans `src/api/handlers/` » au lieu de « Gardez les fichiers organisés »

Cohérence : si deux règles se contredisent, Claude peut en choisir une arbitrairement. Examinez régulièrement vos fichiers CLAUDE.md, les fichiers CLAUDE.md imbriqués dans les sous-répertoires et les fichiers `.claude/rules/` pour supprimer les instructions obsolètes ou conflictuelles. Dans les monorepos, utilisez `claudeMdExcludes` pour ignorer les fichiers CLAUDE.md d'autres équipes qui ne sont pas pertinents pour votre travail.

Importer des fichiers supplémentaires

Les fichiers CLAUDE.md peuvent importer des fichiers supplémentaires en utilisant la syntaxe `@path/to/import`. Les fichiers importés sont développés et chargés dans le contexte au lancement aux côtés du CLAUDE.md qui les référence.

Les chemins relatifs et absolus sont autorisés. Les chemins relatifs se résolvent par rapport au fichier contenant l'import, pas au répertoire de travail. Les fichiers importés peuvent importer récursivement d'autres fichiers, avec une profondeur maximale de cinq sauts.

Pour inclure un README, package.json et un guide de flux de travail, référencez-les avec la syntaxe `@` n'importe où dans votre CLAUDE.md :

```
Voir @README pour un aperçu du projet et @package.json pour les commandes npm disponibles pour ce projet.
```

```
## Instructions supplémentaires
- flux de travail git @docs/git-instructions.md
```

Pour les préférences personnelles que vous ne voulez pas valider, importez un fichier de votre répertoire personnel. L'import se trouve dans le CLAUDE.md partagé, mais le fichier vers lequel il pointe reste sur votre machine :

```
## Préférences individuelles
- @~/ .claude/my-project-instructions.md
```

Warning:

La première fois que Claude Code rencontre des imports externes dans un projet, il affiche une boîte de dialogue d'approbation listant les fichiers. Si vous refusez, les imports restent désactivés et la boîte de dialogue n'apparaît plus.

Pour une approche plus structurée de l'organisation des instructions, voir

[.claude/rules/](#).

Comment les fichiers CLAUDE.md se chargent

Claude Code lit les fichiers CLAUDE.md en remontant l'arborescence des répertoires à partir de votre répertoire de travail actuel, en vérifiant chaque répertoire en chemin. Cela signifie que si vous exécutez Claude Code dans `foo/bar/`, il charge les instructions de `foo/bar/CLAUDE.md` et `foo/CLAUDE.md`.

Claude découvre également les fichiers CLAUDE.md dans les sous-répertoires sous votre répertoire de travail actuel. Au lieu de les charger au lancement, ils sont inclus quand Claude lit les fichiers de ces sous-répertoires.

Si vous travaillez dans un grand monorepo où les fichiers CLAUDE.md d'autres équipes sont détectés, utilisez `claudeMdExcludes` pour les ignorer.

Charger à partir de répertoires supplémentaires

Le drapeau `--add-dir` donne à Claude accès à des répertoires supplémentaires en dehors de votre répertoire de travail principal. Par défaut, les fichiers CLAUDE.md de ces répertoires ne sont pas chargés.

Pour charger également les fichiers CLAUDE.md des répertoires supplémentaires, y compris `CLAUDE.md`, `.claude/CLAUDE.md` et `.claude/rules/*.md`, définissez la variable d'environnement `CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD` :

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

Organiser les règles avec `.claude/rules/`

Pour les projets plus grands, vous pouvez organiser les instructions en plusieurs fichiers en utilisant le répertoire `.claude/rules/`. Cela garde les instructions modulaires et plus faciles à maintenir pour les équipes. Les règles peuvent également être [limitées à des chemins de fichiers spécifiques](#), donc elles ne se chargent dans le contexte que quand Claude travaille avec des fichiers correspondants, réduisant le bruit et économisant l'espace de contexte.

Note:

Les règles se chargent dans le contexte à chaque session ou quand les fichiers correspondants sont ouverts. Pour les instructions spécifiques à une tâche qui n'ont pas besoin d'être dans le contexte tout le temps, utilisez [skills](#) à la place, qui ne se chargent que quand vous les invoquez ou quand Claude détermine qu'elles sont pertinentes pour votre invite.

Configurer les règles

Placez les fichiers markdown dans le répertoire `.claude/rules/` de votre projet. Chaque fichier doit couvrir un sujet, avec un nom de fichier descriptif comme `testing.md` ou `api-design.md`. Tous les fichiers `.md` sont découverts récursivement, donc vous pouvez organiser les règles en sous-répertoires comme `frontend/` ou `backend/` :

```
your-project/
├── .claude/
│   ├── CLAUDE.md          # Instructions principales du projet
│   └── rules/
│       ├── code-style.md  # Directives de style de code
│       ├── testing.md     # Conventions de test
│       └── security.md    # Exigences de sécurité
```

Les règles sans [frontmatter paths](#) sont chargées au lancement avec la même priorité que `.claude/CLAUDE.md`.

Règles spécifiques au chemin

Les règles peuvent être limitées à des fichiers spécifiques en utilisant le frontmatter YAML avec le champ `paths`. Ces règles conditionnelles ne s'appliquent que quand Claude travaille avec des fichiers correspondant aux modèles spécifiés.


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

## Règles de développement d'API

- Tous les points de terminaison d'API doivent inclure la validation des entrées
- Utilisez le format de réponse d'erreur standard
- Incluez les commentaires de documentation OpenAPI
```

Les règles sans champ `paths` sont chargées sans condition et s'appliquent à tous les fichiers. Les règles limitées au chemin se déclenchent quand Claude lit les fichiers correspondant au modèle, pas à chaque utilisation d'outil.

Utilisez les modèles glob dans le champ `paths` pour faire correspondre les fichiers par extension, répertoire ou toute combinaison :

Modèle	Correspond à
<code>**/*.ts</code>	Tous les fichiers TypeScript dans n'importe quel répertoire
<code>src/**/*.*</code>	Tous les fichiers sous le répertoire <code>src/</code>
<code>*.md</code>	Fichiers Markdown à la racine du projet
<code>src/components/*.tsx</code>	Composants React dans un répertoire spécifique

Vous pouvez spécifier plusieurs modèles et utiliser l'expansion d'accolades pour faire correspondre plusieurs extensions dans un modèle :

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

Partager les règles entre les projets avec des liens symboliques

Le répertoire `.claude/rules/` supporte les liens symboliques, donc vous pouvez maintenir un ensemble partagé de règles et les lier dans plusieurs projets. Les liens symboliques sont résolus et chargés normalement, et les liens symboliques circulaires sont détectés et gérés correctement.

Cet exemple lie à la fois un répertoire partagé et un fichier individuel :

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

Règles au niveau utilisateur

Les règles personnelles dans `~/.claude/rules/` s'appliquent à chaque projet sur votre machine. Utilisez-les pour les préférences qui ne sont pas spécifiques au projet :

```
~/.claude/rules/
├─ preferences.md    # Vos préférences de codage personnelles
└─ workflows.md     # Vos flux de travail préférés
```

Les règles au niveau utilisateur sont chargées avant les règles du projet, donnant aux règles du projet une priorité plus élevée.

Gérer CLAUDE.md pour les grandes équipes

Pour les organisations déployant Claude Code sur plusieurs équipes, vous pouvez centraliser les instructions et contrôler quels fichiers CLAUDE.md sont chargés.

Déployer un CLAUDE.md à l'échelle de l'organisation

Les organisations peuvent déployer un CLAUDE.md géré centralement qui s'applique à tous les utilisateurs sur une machine. Ce fichier ne peut pas être exclu par les paramètres individuels.

Step 1: Créer le fichier à l'emplacement de la politique gérée

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

Step 2: Déployer avec votre système de gestion de configuration

Utilisez MDM, Group Policy, Ansible ou des outils similaires pour distribuer le fichier sur les machines des développeurs. Voir [paramètres gérés](#) pour d'autres options de configuration à l'échelle de l'organisation.

Exclure des fichiers CLAUDE.md spécifiques

Dans les grands monorepos, les fichiers CLAUDE.md ancêtres peuvent contenir des instructions qui ne sont pas pertinentes pour votre travail. Le paramètre `claudeMdExcludes` vous permet d'ignorer des fichiers spécifiques par chemin ou modèle glob.

Cet exemple exclut un CLAUDE.md de niveau supérieur et un répertoire de règles d'un dossier parent. Ajoutez-le à `.claude/settings.local.json` pour que l'exclusion reste locale à votre machine :

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

Les modèles sont comparés aux chemins de fichiers absolus en utilisant la syntaxe glob. Vous pouvez configurer `claudeMdExcludes` à n'importe quel [niveau de paramètres](#) : utilisateur, projet, local ou politique gérée. Les tableaux fusionnent entre les niveaux.

Les fichiers CLAUDE.md de politique gérée ne peuvent pas être exclus. Cela garantit que les instructions à l'échelle de l'organisation s'appliquent toujours indépendamment des paramètres individuels.

Mémoire automatique

La mémoire automatique permet à Claude d'accumuler des connaissances entre les sessions sans que vous n'écriviez rien. Claude enregistre des notes pour lui-même au fur et à mesure qu'il travaille : commandes de compilation, insights de débogage, notes d'architecture, préférences de style de code et habitudes de flux de travail. Claude ne sauvegarde pas quelque chose à chaque session. Il décide ce qui vaut la peine d'être mémorisé en fonction de si l'information serait utile dans une conversation future.

Note:

La mémoire automatique nécessite Claude Code v2.1.59 ou ultérieur. Vérifiez votre version avec `claude --version`.

Activer ou désactiver la mémoire automatique

La mémoire automatique est activée par défaut. Pour la basculer, ouvrez `/memory` dans une session et utilisez le bouton bascule de mémoire automatique, ou définissez `autoMemoryEnabled` dans vos paramètres de projet :

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

Pour désactiver la mémoire automatique via une variable d'environnement, définissez `CLAUDE_CODE_DISABLE_AUTO_MEMORY=1`.

Emplacement de stockage

Chaque projet obtient son propre répertoire de mémoire à `~/.claude/projects/<project>/memory/`. Le chemin `<project>` est dérivé du référentiel git, donc tous les worktrees et sous-répertoires dans le même référentiel partagent un répertoire de mémoire automatique. En dehors d'un référentiel git, la racine du projet est utilisée à la place.

Pour stocker la mémoire automatique dans un emplacement différent, définissez `autoMemoryDirectory` dans vos paramètres utilisateur ou locaux :

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

Ce paramètre est accepté à partir des paramètres de politique, locaux et utilisateur. Il n'est pas accepté à partir des paramètres de projet (`.claude/settings.json`) pour empêcher un projet partagé de rediriger les écritures de mémoire automatique vers des emplacements sensibles.

Le répertoire contient un point d'entrée `MEMORY.md` et des fichiers de sujet optionnels :

```
~/.claude/projects/<project>/memory/  
├─ MEMORY.md          # Index concis, chargé dans chaque session  
├─ debugging.md       # Notes détaillées sur les modèles de débogage  
├─ api-conventions.md # Décisions de conception d'API  
└─ ...                # Tout autre fichier de sujet que Claude crée
```

`MEMORY.md` agit comme un index du répertoire de mémoire. Claude lit et écrit des fichiers dans ce répertoire tout au long de votre session, en utilisant `MEMORY.md` pour garder une trace de ce qui est stocké où.

La mémoire automatique est locale à la machine. Tous les worktrees et sous-répertoires dans le même référentiel git partagent un répertoire de mémoire automatique. Les fichiers ne sont pas partagés entre les machines ou les environnements cloud.

Comment ça marche

Les 200 premières lignes de `MEMORY.md` sont chargées au début de chaque conversation. Le contenu au-delà de la ligne 200 n'est pas chargé au démarrage de la session. Claude garde `MEMORY.md` concis en déplaçant les notes détaillées dans des fichiers de sujet séparés.

Cette limite de 200 lignes s'applique uniquement à `MEMORY.md`. Les fichiers `CLAUDE.md` sont chargés en intégralité indépendamment de la longueur, bien que les fichiers plus courts produisent une meilleure adhérence.

Les fichiers de sujet comme `debugging.md` ou `patterns.md` ne sont pas chargés au démarrage. Claude les lit à la demande en utilisant ses outils de fichiers standard quand il a besoin de l'information.

Claude lit et écrit les fichiers de mémoire pendant votre session. Quand vous voyez « Writing memory » ou « Recalled memory » dans l'interface Claude Code, Claude met activement à jour ou lit à partir de `~/.claude/projects/<project>/memory/`.

Auditer et modifier votre mémoire

Les fichiers de mémoire automatique sont du markdown brut que vous pouvez modifier ou supprimer à tout moment. Exécutez `/memory` pour parcourir et ouvrir les fichiers de mémoire à partir d'une session.

Afficher et modifier avec `/memory`

La commande `/memory` liste tous les fichiers CLAUDE.md et rules chargés dans votre session actuelle, vous permet de basculer la mémoire automatique activée ou désactivée, et fournit un lien pour ouvrir le dossier de mémoire automatique. Sélectionnez n'importe quel fichier pour l'ouvrir dans votre éditeur.

Quand vous demandez à Claude de se souvenir de quelque chose, comme « toujours utiliser npm, pas npx » ou « se souvenir que les tests d'API nécessitent une instance Redis locale », Claude l'enregistre dans la mémoire automatique. Pour ajouter des instructions à CLAUDE.md à la place, demandez à Claude directement, comme « ajouter ceci à CLAUDE.md », ou modifiez le fichier vous-même via `/memory`.

Dépanner les problèmes de mémoire

Ce sont les problèmes les plus courants avec CLAUDE.md et la mémoire automatique, ainsi que les étapes pour les déboguer.

Claude ne suit pas mon CLAUDE.md

CLAUDE.md est du contexte, pas une application. Claude le lit et essaie de le suivre, mais il n'y a aucune garantie de conformité stricte, surtout pour les instructions vagues ou conflictuelles.

Pour déboguer :

- Exécutez `/memory` pour vérifier que vos fichiers CLAUDE.md sont chargés. Si un fichier n'est pas listé, Claude ne peut pas le voir.
- Vérifiez que le CLAUDE.md pertinent se trouve dans un emplacement qui se charge pour votre session (voir [Choisir où placer les fichiers CLAUDE.md](#)).
- Rendez les instructions plus spécifiques. « Utiliser l'indentation à 2 espaces » fonctionne mieux que « formater le code correctement ».
- Recherchez les instructions conflictuelles entre les fichiers CLAUDE.md. Si deux fichiers donnent des conseils différents pour le même comportement, Claude peut en choisir un arbitrairement.

Tip:

Utilisez le `hook InstructionsLoaded` pour enregistrer exactement quels fichiers d'instructions sont chargés, quand ils se chargent et pourquoi. Ceci est utile pour déboguer les règles spécifiques au chemin ou les fichiers chargés tardivement dans les sous-répertoires.

Je ne sais pas ce que la mémoire automatique a sauvegardé

Exécutez `/memory` et sélectionnez le dossier de mémoire automatique pour parcourir ce que Claude a sauvegardé. Tout est du markdown brut que vous pouvez lire, modifier ou supprimer.

Mon CLAUDE.md est trop volumineux

Les fichiers de plus de 200 lignes consomment plus de contexte et peuvent réduire l'adhérence. Déplacez le contenu détaillé dans des fichiers séparés référencés avec des imports `@path` (voir [Importer des fichiers supplémentaires](#)), ou divisez vos instructions entre les fichiers `.claude/rules/`.

Les instructions semblent perdues après `/compact`

CLAUDE.md survit complètement à la compaction. Après `/compact`, Claude relit votre CLAUDE.md à partir du disque et le réinjecte à nouveau dans la session. Si une instruction a disparu après la compaction, elle a été donnée uniquement dans la conversation, pas écrite dans CLAUDE.md. Ajoutez-la à CLAUDE.md pour la rendre persistante entre les sessions.

Voir [Écrire des instructions efficaces](#) pour des conseils sur la taille, la structure et la spécificité.

Ressources connexes

- [Skills](#) : empaquetez les flux de travail répétables qui se chargent à la demande
- [Paramètres](#) : configurez le comportement de Claude Code avec les fichiers de paramètres
- [Gérer les sessions](#) : gérez le contexte, reprenez les conversations et exécutez des sessions parallèles
- [Mémoire des subagents](#) : laissez les subagents maintenir leur propre mémoire automatique

Paramètres Claude Code

Configurez Claude Code avec des paramètres globaux et au niveau du projet, ainsi que des variables d'environnement.

Claude Code offre une variété de paramètres pour configurer son comportement selon vos besoins. Vous pouvez configurer Claude Code en exécutant la commande `/config` lors de l'utilisation du REPL interactif, ce qui ouvre une interface Paramètres avec onglets où vous pouvez afficher les informations d'état et modifier les options de configuration.

Portées de configuration

Claude Code utilise un **système de portées** pour déterminer où les configurations s'appliquent et qui les partage. Comprendre les portées vous aide à décider comment configurer Claude Code pour un usage personnel, une collaboration d'équipe ou un déploiement en entreprise.

Portées disponibles

Portée	Emplacement	Qui est affecté	Partagé avec l'équipe ?
Managed	Paramètres gérés par le serveur, plist / registre, ou <code>managed-settings.json</code> au niveau système	Tous les utilisateurs de la machine	Oui (déployé par l'IT)
User	Répertoire <code>~/.claude/</code>	Vous, sur tous les projets	Non
Project	<code>.claude/</code> dans le référentiel	Tous les collaborateurs de ce référentiel	Oui (commité dans git)
Local	<code>.claude/settings.local.json</code>	Vous, dans ce référentiel uniquement	Non (ignoré par gitignore)

Quand utiliser chaque portée

La portée **Managed** est destinée à :

- Les politiques de sécurité qui doivent être appliquées à l'échelle de l'organisation

- Les exigences de conformité qui ne peuvent pas être contournées
- Les configurations standardisées déployées par l'IT/DevOps

La portée **User** est idéale pour :

- Les préférences personnelles que vous voulez partout (thèmes, paramètres d'éditeur)
- Les outils et plugins que vous utilisez sur tous les projets
- Les clés API et l'authentification (stockées de manière sécurisée)

La portée **Project** est idéale pour :

- Les paramètres partagés par l'équipe (permissions, hooks, serveurs MCP)
- Les plugins que toute l'équipe devrait avoir
- La standardisation des outils entre collaborateurs

La portée **Local** est idéale pour :

- Les remplacements personnels pour un projet spécifique
- Les configurations de test avant de les partager avec l'équipe
- Les paramètres spécifiques à la machine qui ne fonctionneront pas pour les autres

Comment les portées interagissent

Lorsque le même paramètre est configuré dans plusieurs portées, les portées plus spécifiques ont la priorité :

1. **Managed** (la plus élevée) - ne peut pas être remplacée par quoi que ce soit
2. **Arguments de ligne de commande** - remplacements de session temporaires
3. **Local** - remplace les paramètres du projet et de l'utilisateur
4. **Project** - remplace les paramètres de l'utilisateur
5. **User** (la plus basse) - s'applique quand rien d'autre ne spécifie le paramètre

Par exemple, si une permission est autorisée dans les paramètres utilisateur mais refusée dans les paramètres du projet, le paramètre du projet a la priorité et la permission est bloquée.

Ce qui utilise les portées

Les portées s'appliquent à de nombreuses fonctionnalités de Claude Code :

Fonction- nalité	Emplacement utilisateur	Emplacement du projet	Emplacement lo- cal
Para- mètres	<code>~/.claude/settings.json</code>	<code>.claude/ settings.json</code>	<code>.claude/ settings.local.js on</code>
Sub- agents	<code>~/.claude/agents/</code>	<code>.claude/agents/</code>	—
Serveurs MCP	<code>~/.claude.json</code>	<code>.mcp.json</code>	<code>~/.claude.json</code> (par projet)
Plugins	<code>~/.claude/settings.json</code>	<code>.claude/ settings.json</code>	<code>.claude/ settings.local.js on</code>
CLAUDE. md	<code>~/.claude/CLAUDE.md</code>	<code>CLAUDE.md</code> ou <code>.claude/ CLAUDE.md</code>	—

Fichiers de paramètres

Le fichier `settings.json` est notre mécanisme officiel pour configurer Claude Code via des paramètres hiérarchiques :

- Les **paramètres utilisateur** sont définis dans `~/.claude/settings.json` et s'appliquent à tous les projets.
- Les **paramètres du projet** sont enregistrés dans votre répertoire de projet :
 - `.claude/settings.json` pour les paramètres qui sont vérifiés dans le contrôle de source et partagés avec votre équipe
 - `.claude/settings.local.json` pour les paramètres qui ne sont pas vérifiés, utiles pour les préférences personnelles et l'expérimentation. Claude Code configurera git pour ignorer `.claude/settings.local.json` lors de sa création.
- **Paramètres gérés** : Pour les organisations qui ont besoin d'un contrôle centralisé, Claude Code prend en charge plusieurs mécanismes de livraison pour les paramètres gérés. Tous utilisent le même format JSON et ne peuvent pas être remplacés par les paramètres utilisateur ou du projet :
 - **Paramètres gérés par le serveur** : livrés depuis les serveurs d'Anthropic via la console d'administration Claude.ai. Voir [paramètres gérés par le serveur](#).

- **Politiques MDM/au niveau du système d'exploitation** : livrées via la gestion native des appareils sur macOS et Windows :
 - macOS : domaine de préférences gérées `com.anthropic.claudecode` (déployé via des profils de configuration dans Jamf, Kandji ou d'autres outils MDM)
 - Windows : clé de registre `HKLM\SOFTWARE\Policies\ClaudeCode` avec une valeur `Settings` (REG_SZ ou REG_EXPAND_SZ) contenant du JSON (déployé via la stratégie de groupe ou Intune)
 - Windows (au niveau utilisateur) : `HKCU\SOFTWARE\Policies\ClaudeCode` (priorité de politique la plus basse, utilisée uniquement quand aucune source au niveau administrateur n'existe)
- **Basé sur fichier** : `managed-settings.json` et `managed-mcp.json` déployés dans les répertoires système :
 - macOS : `/Library/Application Support/ClaudeCode/`
 - Linux et WSL : `/etc/claude-code/`
 - Windows : `C:\Program Files\ClaudeCode\`

Voir [paramètres gérés](#) et [Configuration MCP gérée](#) pour plus de détails.

Note:

Les déploiements gérés peuvent également restreindre les **ajouts de marketplace de plugins** en utilisant `strictKnownMarketplaces`. Pour plus d'informations, voir [Restrictions de marketplace gérées](#).

- **Autre configuration** est stockée dans `~/.claude.json`. Ce fichier contient vos préférences (thème, paramètres de notification, mode d'éditeur), session OAuth, configurations de [serveur MCP](#) pour les portées utilisateur et locale, état par projet (outils autorisés, paramètres de confiance), et divers caches. Les serveurs MCP à portée de projet sont stockés séparément dans `.mcp.json`.

Note:

Claude Code crée automatiquement des sauvegardes horodatées des fichiers de configuration et conserve les cinq sauvegardes les plus récentes pour éviter la perte de données.

```
{
  "$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 ligne `$schema` dans l'exemple ci-dessus pointe vers le [schéma JSON officiel](https://json.schemastore.org/claude-code-settings.json) pour les paramètres Claude Code. L'ajouter à votre `settings.json` active l'autocomplétion et la validation en ligne dans VS Code, Cursor et tout autre éditeur qui prend en charge la validation de schéma JSON.

Paramètres disponibles

`settings.json` prend en charge un certain nombre d'options :

Clé	Description	Exemple
<code>apiKeyHelper</code>	Script personnalisé, à exécuter dans <code>/bin/sh</code> , pour générer une valeur d'authentification. Cette valeur sera envoyée comme en-têtes <code>X-API-Key</code> et <code>Authorization: Bearer</code> pour les demandes de modèle	<code>/bin/generate_temp_api_key.sh</code>
<code>cleanupPeriodDays</code>	Les sessions inactives plus longtemps que cette période sont supprimées au démarrage. La définition à <code>0</code> supprime immédiatement toutes les sessions. (par défaut : 30 jours)	<code>20</code>
<code>companyAnnouncements</code>	Annonce à afficher aux utilisateurs au démarrage. Si plusieurs annonces sont fournies, elles seront parcourues aléatoirement.	<code>["Welcome to Acme Corp! Review our code guidelines at docs.acme.com"]</code>
<code>env</code>	Variables d'environnement qui seront appliquées à chaque session	<code>{"FOO": "bar"}</code>
<code>attribution</code>	Personnalisez l'attribution pour les commits git et les demandes de tirage. Voir Paramètres d'attribution	<code>{"commit": "🤖 Generated with Claude Code", "pr": ""}</code>
<code>includeCoAuthoredBy</code>	Obsolète : Utilisez <code>attribution</code> à la place. S'il faut inclure la ligne <code>co-authored-by Claude</code> dans les commits git et les demandes de tirage (par défaut : <code>true</code>)	<code>false</code>
<code>includeGitInstructions</code>	Inclure les instructions de workflow de commit et de PR intégrées dans l'invite système de Claude (par défaut : <code>true</code>). Définissez à <code>false</code> pour supprimer ces instructions, par exemple lors de l'utilisation de vos propres skills de workflow git. La variable d'environnement <code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code> a la priorité sur ce paramètre lorsqu'elle est définie	<code>false</code>

Clé	Description	Exemple
<code>permissions</code>	Voir le tableau ci-dessous pour la structure des permissions.	
<code>hooks</code>	Configurez des commandes personnalisées à exécuter lors d'événements du cycle de vie. Voir documentation des hooks pour le format	Voir hooks
<code>disableAllHooks</code>	Désactiver tous les hooks et toute ligne d'état personnalisée	<code>true</code>
<code>allowManagedHooksOnly</code>	(Paramètres gérés uniquement) Empêcher le chargement des hooks utilisateur, projet et plugin. Autorise uniquement les hooks gérés et les hooks SDK. Voir Configuration des hooks	<code>true</code>
<code>allowedHttpHookUrls</code>	Liste blanche des modèles d'URL que les hooks HTTP peuvent cibler. Prend en charge <code>*</code> comme caractère générique. Lorsqu'elle est définie, les hooks avec des URL non correspondantes sont bloqués. Non défini = pas de restriction, tableau vide = bloquer tous les hooks HTTP. Les tableaux fusionnent entre les sources de paramètres. Voir Configuration des hooks	<code>["https://hooks.example.com/*"]</code>
<code>httpHookAllowedEnvVars</code>	Liste blanche des noms de variables d'environnement que les hooks HTTP peuvent interpoler dans les en-têtes. Lorsqu'elle est définie, le <code>allowedEnvVars</code> effectif de chaque hook est l'intersection avec cette liste. Non défini = pas de restriction. Les tableaux fusionnent entre les sources de paramètres. Voir Configuration des hooks	<code>["MY_TOKEN", "HOOK_SECRET"]</code>

Clé	Description	Exemple
<code>allowManagedPermissionsRulesOnly</code>	(Paramètres gérés uniquement) Empêcher les paramètres utilisateur et projet de définir les règles de permission <code>allow</code> , <code>ask</code> ou <code>deny</code> . Seules les règles des paramètres gérés s'appliquent. Voir Paramètres réservés aux gérés	<code>true</code>
<code>allowManagedMcpServersOnly</code>	(Paramètres gérés uniquement) Seul <code>allowedMcpServers</code> des paramètres gérés est respecté. <code>deniedMcpServers</code> fusionne toujours à partir de toutes les sources. Les utilisateurs peuvent toujours ajouter des serveurs MCP, mais seule la liste blanche définie par l'administrateur s'applique. Voir Configuration MCP gérée	<code>true</code>
<code>model</code>	Remplacer le modèle par défaut à utiliser pour Claude Code	<code>"claude-sonnet-4-6"</code>
<code>availableModels</code>	Restreindre les modèles que les utilisateurs peuvent sélectionner via <code>/model</code> , <code>--model</code> , l'outil Config ou <code>ANTHROPIC_MODEL</code> . N'affecte pas l'option Par défaut. Voir Restreindre la sélection de modèle	<code>["sonnet", "haiku"]</code>
<code>modelOverrides</code>	Mapper les ID de modèle Anthropic aux ID de modèle spécifiques au fournisseur tels que les ARN de profil d'inférence Bedrock. Chaque entrée du sélecteur de modèle utilise sa valeur mappée lors de l'appel de l'API du fournisseur. Voir Remplacer les ID de modèle par version	<code>{"claude-opus-4-6": "arn:aws:bedrock:..." }</code>
<code>otelHeadersHelper</code>	Script pour générer des en-têtes OpenTelemetry dynamiques. S'exécute au démarrage et périodiquement (voir En-têtes dynamiques)	<code>/bin/ generate_otel_headers .sh</code>

Clé	Description	Exemple
<code>statusLine</code>	Configurez une ligne d'état personnalisée pour afficher le contexte. Voir documentation <code>statusLine</code>	<pre>{"type": "command", "command": "~/ .claude/ statusline.sh"}</pre>
<code>fileSuggestion</code>	Configurez un script personnalisé pour l'autocomplétion de fichier <code>@</code> . Voir Paramètres de suggestion de fichier	<pre>{"type": "command", "command": "~/ .claude/file- suggestion.sh"}</pre>
<code>respectGitignore</code>	Contrôlez si le sélecteur de fichier <code>@</code> respecte les modèles <code>.gitignore</code> . Lorsque <code>true</code> (par défaut), les fichiers correspondant aux modèles <code>.gitignore</code> sont exclus des suggestions	<code>false</code>
<code>outputStyle</code>	Configurez un style de sortie pour ajuster l'invite système. Voir documentation des styles de sortie	<code>"Explanatory"</code>
<code>forceLoginMethod</code>	Utilisez <code>claudeai</code> pour restreindre la connexion aux comptes Claude.ai, <code>console</code> pour restreindre la connexion aux comptes Claude Console (facturation d'utilisation d'API)	<code>claudeai</code>
<code>forceLoginOrgUUID</code>	Spécifiez l'UUID d'une organisation pour la sélectionner automatiquement lors de la connexion, en contournant l'étape de sélection d'organisation. Nécessite que <code>forceLoginMethod</code> soit défini	<code>"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"</code>
<code>enableAllProjectMcpServers</code>	Approuver automatiquement tous les serveurs MCP définis dans les fichiers <code>.mcp.json</code> du projet	<code>true</code>
<code>enabledMcpjsonServers</code>	Liste des serveurs MCP spécifiques des fichiers <code>.mcp.json</code> à approuver	<code>["memory", "github"]</code>

Clé	Description	Exemple
<code>disabledMcpjsonServers</code>	Liste des serveurs MCP spécifiques des fichiers <code>.mcp.json</code> à rejeter	<code>["filesystem"]</code>
<code>allowedMcpServers</code>	Lorsqu'elle est définie dans <code>managed-settings.json</code> , liste blanche des serveurs MCP que les utilisateurs peuvent configurer. Non défini = pas de restrictions, tableau vide = verrouillage. S'applique à toutes les portées. La liste noire a la priorité. Voir Configuration MCP gérée	<code>{ "serverName": "github" }</code>
<code>deniedMcpServers</code>	Lorsqu'elle est définie dans <code>managed-settings.json</code> , liste noire des serveurs MCP qui sont explicitement bloqués. S'applique à toutes les portées, y compris les serveurs gérés. La liste noire a la priorité sur la liste blanche. Voir Configuration MCP gérée	<code>{ "serverName": "filesystem" }</code>
<code>strictKnownMarketplaces</code>	Lorsqu'elle est définie dans <code>managed-settings.json</code> , liste blanche des marketplaces de plugins que les utilisateurs peuvent ajouter. Non défini = pas de restrictions, tableau vide = verrouillage. S'applique uniquement aux ajouts de marketplace. Voir Restrictions de marketplace gérées	<code>{ "source": "github", "repo": "acme-corp/plugins" }</code>
<code>blockedMarketplaces</code>	(Paramètres gérés uniquement) Liste noire des sources de marketplace. Les sources bloquées sont vérifiées avant le téléchargement, elles ne touchent donc jamais le système de fichiers. Voir Restrictions de marketplace gérées	<code>{ "source": "github", "repo": "untrusted/plugins" }</code>

Clé	Description	Exemple
<code>pluginTrustMessage</code>	(Paramètres gérés uniquement) Message personnalisé ajouté à l'avertissement de confiance du plugin affiché avant l'installation. Utilisez ceci pour ajouter un contexte spécifique à l'organisation, par exemple pour confirmer que les plugins de votre marketplace interne sont vérifiés.	<code>"All plugins from our marketplace are approved by IT"</code>
<code>awsAuthRefresh</code>	Script personnalisé qui modifie le répertoire <code>.aws</code> (voir configuration avancée des identifiants)	<code>aws sso login --profile myprofile</code>
<code>awsCredentialExport</code>	Script personnalisé qui génère du JSON avec les identifiants AWS (voir configuration avancée des identifiants)	<code>/bin/generate_aws_grant.sh</code>
<code>alwaysThinkingEnabled</code>	Activer la réflexion étendue par défaut pour toutes les sessions. Généralement configuré via la commande <code>/config</code> plutôt que d'éditer directement	<code>true</code>
<code>plansDirectory</code>	Personnalisez où les fichiers de plan sont stockés. Le chemin est relatif à la racine du projet. Par défaut : <code>~/.claude/plans</code>	<code>"./plans"</code>
<code>showTurnDuration</code>	Afficher les messages de durée de tour après les réponses (par exemple, « Cooked for 1m 6s »). Définissez à <code>false</code> pour masquer ces messages	<code>true</code>
<code>spinnerVerbs</code>	Personnalisez les verbes d'action affichés dans le spinner et les messages de durée de tour. Définissez <code>mode</code> à <code>"replace"</code> pour utiliser uniquement vos verbes, ou <code>"append"</code> pour les ajouter aux valeurs par défaut	<code>{"mode": "append", "verbs": ["Pondering", "Crafting"]}</code>

Clé	Description	Exemple
<code>language</code>	Configurez la langue de réponse préférée de Claude (par exemple, <code>"japanese"</code> , <code>"spanish"</code> , <code>"french"</code>). Claude répondra dans cette langue par défaut	<code>"japanese"</code>
<code>autoUpdatesChannel</code>	Canal de version à suivre pour les mises à jour. Utilisez <code>"stable"</code> pour une version généralement une semaine plus ancienne et qui ignore les versions avec des régressions majeures, ou <code>"latest"</code> (par défaut) pour la version la plus récente	<code>"stable"</code>
<code>spinnerTipsEnabled</code>	Afficher les conseils dans le spinner pendant que Claude travaille. Définissez à <code>false</code> pour désactiver les conseils (par défaut : <code>true</code>)	<code>false</code>
<code>spinnerTipsOverride</code>	Remplacer les conseils du spinner par des chaînes personnalisées. <code>tips</code> : tableau de chaînes de conseil. <code>excludeDefault</code> : si <code>true</code> , afficher uniquement les conseils personnalisés ; si <code>false</code> ou absent, les conseils personnalisés sont fusionnés avec les conseils intégrés	<pre>{ "excludeDefault": true, "tips": ["Use our internal tool X"] }</pre>
<code>terminalProgressBarEnabled</code>	Activer la barre de progression du terminal qui affiche la progression dans les terminaux pris en charge comme Windows Terminal et iTerm2 (par défaut : <code>true</code>)	<code>false</code>
<code>prefersReducedMotion</code>	Réduire ou désactiver les animations de l'interface utilisateur (spinners, shimmer, effets flash) pour l'accessibilité	<code>true</code>

Clé	Description	Exemple
<code>fastModePerSessionOptIn</code>	Lorsque <code>true</code> , le mode rapide ne persiste pas entre les sessions. Chaque session commence avec le mode rapide désactivé, ce qui nécessite que les utilisateurs l'activent avec <code>/fast</code> . La préférence de mode rapide de l'utilisateur est toujours enregistrée. Voir Exiger l'opt-in par session	<code>true</code>
<code>teammateMode</code>	Comment les coéquipiers de l'équipe d'agents s'affichent : <code>auto</code> (choisit les volets divisés dans tmux ou iTerm2, en processus sinon), <code>in-process</code> ou <code>tmux</code> . Voir configurer les équipes d'agents	<code>"in-process"</code>

Paramètres de permission

Clés	Description	Exemple
<code>allow</code>	Tableau des règles de permission pour autoriser l'utilisation d'outils. Voir Syntaxe des règles de permission ci-dessous pour les détails de correspondance de modèles	<code>["Bash(git diff *)"]</code>
<code>ask</code>	Tableau des règles de permission pour demander une confirmation lors de l'utilisation d'outils. Voir Syntaxe des règles de permission ci-dessous	<code>["Bash(git push *)"]</code>
<code>deny</code>	Tableau des règles de permission pour refuser l'utilisation d'outils. Utilisez ceci pour exclure les fichiers sensibles de l'accès de Claude Code. Voir Syntaxe des règles de permission et Limitations de permission Bash	<code>["WebFetch", "Bash(curl *)", "Read(./env)", "Read(./secrets/**)"]</code>

Clés	Description	Exemple
<code>additionalDirectories</code>	Répertoires de travail supplémentaires auxquels Claude a accès	<code>["../docs/"]</code>
<code>defaultMode</code>	Mode de permission par défaut lors de l'ouverture de Claude Code	<code>"acceptEdits"</code>
<code>disableBypassPermissionsMode</code>	Définissez à <code>"disable"</code> pour empêcher l'activation du mode <code>bypassPermissions</code> . Cela désactive l'indicateur de ligne de commande <code>--dangerously-skip-permissions</code> . Voir paramètres gérés	<code>"disable"</code>

Syntaxe des règles de permission

Les règles de permission suivent le format `Tool` ou `Tool(specifieur)`. Les règles sont évaluées dans l'ordre : d'abord les règles de refus, puis de demande, puis d'autorisation. La première règle correspondante gagne.

Exemples rapides :

Règle	Effet
<code>Bash</code>	Correspond à toutes les commandes Bash
<code>Bash(npm run *)</code>	Correspond aux commandes commençant par <code>npm run</code>
<code>Read(./env)</code>	Correspond à la lecture du fichier <code>env</code>
<code>WebFetch(domain:example.com)</code>	Correspond aux demandes de récupération vers <code>example.com</code>

Pour la référence complète de la syntaxe des règles, y compris le comportement des caractères génériques, les modèles spécifiques aux outils pour Read, Edit, WebFetch, MCP et Agent, et les limitations de sécurité des modèles Bash, voir [Syntaxe des règles de permission](#).

Paramètres de sandbox

Configurez le comportement avancé du sandboxing. Le sandboxing isole les commandes bash de votre système de fichiers et de votre réseau. Voir [Sandboxing](#) pour plus de détails.

Clés	Description	Exemple
<code>enabled</code>	Activer le sandboxing bash (macOS, Linux et WSL2). Par défaut : false	<code>true</code>
<code>autoAllowBashIfSandboxed</code>	Approuver automatiquement les commandes bash lorsqu'elles sont sandboxées. Par défaut : true	<code>true</code>
<code>excludedCommands</code>	Commandes qui doivent s'exécuter en dehors du sandbox	<code>["git", "docker"]</code>
<code>allowUnsandboxedCommands</code>	Autoriser les commandes à s'exécuter en dehors du sandbox via le paramètre <code>dangerouslyDisableSandbox</code> . Lorsqu'elle est définie à <code>false</code> , l'échappatoire <code>dangerouslyDisableSandbox</code> est complètement désactivée et toutes les commandes doivent s'exécuter en sandbox (ou être dans <code>excludedCommands</code>). Utile pour les politiques d'entreprise qui nécessitent un sandboxing strict. Par défaut : true	<code>false</code>
<code>filesystem.allowWrite</code>	Chemins supplémentaires où les commandes sandboxées peuvent écrire. Les tableaux sont fusionnés sur toutes les portées de paramètres : les chemins utilisateur, projet et gérés sont combinés, non remplacés. Également fusionnés avec les chemins des règles de permission <code>Edit(...)</code> allow. Voir préfixes de chemin ci-dessous.	<code>["//tmp/build", "~/.kube"]</code>
<code>filesystem.denyWrite</code>	Chemins où les commandes sandboxées ne peuvent pas écrire. Les tableaux sont fusionnés sur toutes les portées de paramètres. Également fusionnés avec les chemins des règles de permission <code>Edit(...)</code> deny.	<code>["//etc", "//usr/local/bin"]</code>

Clés	Description	Exemple
<code>filesystem.denyRead</code>	Chemins où les commandes sandboxées ne peuvent pas lire. Les tableaux sont fusionnés sur toutes les portées de paramètres. Également fusionnés avec les chemins des règles de permission <code>Read(...)</code> deny.	<code>["~/.aws/credentials"]</code>
<code>network.allowUnixSockets</code>	Chemins de socket Unix accessibles dans le sandbox (pour les agents SSH, etc.)	<code>["~/.ssh/agent-socket"]</code>
<code>network.allowAllUnixSockets</code>	Autoriser toutes les connexions de socket Unix dans le sandbox. Par défaut : false	<code>true</code>
<code>network.allowLocalBinding</code>	Autoriser la liaison aux ports localhost (macOS uniquement). Par défaut : false	<code>true</code>
<code>network.allowedDomains</code>	Tableau des domaines à autoriser pour le trafic réseau sortant. Prend en charge les caractères génériques (par exemple, <code>*.example.com</code>).	<code>["github.com", "*.npmjs.org"]</code>
<code>network.allowManagedDomainsOnly</code>	(Paramètres gérés uniquement) Seul <code>allowedDomains</code> et les règles <code>allowWebFetch(domain:...)</code> des paramètres gérés sont respectés. Les domaines des paramètres utilisateur, projet et local sont ignorés. Les domaines non autorisés sont bloqués automatiquement sans inviter l'utilisateur. Les domaines refusés sont toujours respectés à partir de toutes les sources. Par défaut : false	<code>true</code>
<code>network.httpProxyPort</code>	Port du proxy HTTP utilisé si vous souhaitez apporter votre propre proxy. S'il n'est pas spécifié, Claude exécutera son propre proxy.	<code>8080</code>

Clés	Description	Exemple
<code>network.socksProxyPort</code>	Port du proxy SOCKS5 utilisé si vous souhaitez apporter votre propre proxy. S'il n'est pas spécifié, Claude exécutera son propre proxy.	<code>8081</code>
<code>enableWeakerNestedSandbox</code>	Activer un sandbox plus faible pour les environnements Docker non privilégiés (Linux et WSL2 uniquement). Réduit la sécurité . Par défaut : false	<code>true</code>
<code>enableWeakerNetworkIsolation</code>	(macOS uniquement) Autoriser l'accès au service de confiance TLS du système (<code>com.apple.trustd.agent</code>) dans le sandbox. Requis pour que les outils basés sur Go comme <code>gh</code> , <code>gcloud</code> et <code>terraform</code> vérifient les certificats TLS lors de l'utilisation de <code>httpProxyPort</code> avec un proxy MITM et une CA personnalisée. Réduit la sécurité en ouvrant un chemin potentiel d'exfiltration de données. Par défaut : false	<code>true</code>

Préfixes de chemin du sandbox

Les chemins dans `filesystem.allowWrite`, `filesystem.denyWrite` et `filesystem.denyRead` prennent en charge ces préfixes :

Préfixe	Signification	Exemple
<code>//</code>	Chemin absolu à partir de la racine du système de fichiers	<code>//tmp/build</code> devient <code>/tmp/build</code>
<code>~/</code>	Relatif au répertoire personnel	<code>~/kubernetes</code> devient <code>\$HOME/kubernetes</code>
<code>/</code>	Relatif au répertoire du fichier de paramètres	<code>/build</code> devient <code>\$SETTINGS_DIR/build</code>
<code>./</code> ou pas de préfixe	Chemin relatif (résolu par le runtime du sandbox)	<code>./output</code>

Exemple de configuration :

```
{
  "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
    }
  }
}
```

Les **restrictions de système de fichiers et de réseau** peuvent être configurées de deux façons qui sont fusionnées ensemble :

- **Paramètres `sandbox.filesystem`** (affichés ci-dessus) : Contrôlez les chemins à la limite du sandbox au niveau du système d'exploitation. Ces restrictions s'appliquent à toutes les commandes de sous-processus (par exemple, `kubectl`, `terraform`, `npm`), pas seulement aux outils de fichier de Claude.
- **Règles de permission** : Utilisez les règles allow/deny `Edit` pour contrôler l'accès à l'outil de fichier de Claude, les règles deny `Read` pour bloquer les lectures, et les règles allow/deny `WebFetch` pour contrôler les domaines réseau. Les chemins de ces règles sont également fusionnés dans la configuration du sandbox.

Paramètres d'attribution

Claude Code ajoute l'attribution aux commits git et aux demandes de tirage. Ceux-ci sont configurés séparément :

- Les commits utilisent les [trailers git](#) (comme `Co-Authored-By`) par défaut, qui peuvent être personnalisés ou désactivés

- Les descriptions des demandes de tirage sont du texte brut

Clés	Description
<code>commit</code>	Attribution pour les commits git, y compris tous les trailers. La chaîne vide masque l'attribution du commit
<code>pr</code>	Attribution pour les descriptions des demandes de tirage. La chaîne vide masque l'attribution de la demande de tirage

Attribution de commit par défaut :

 Generated with [Claude Code](https://claude.com/claude-code)

Co-Authored-By: Claude Sonnet 4.6 <noreply@anthropic.com>

Attribution de demande de tirage par défaut :

 Generated with [Claude Code](https://claude.com/claude-code)

Exemple :

```
{
  "attribution": {
    "commit": "Generated with AI\n\nCo-Authored-By: AI <ai@example.com>",
    "pr": ""
  }
}
```

Note:

Le paramètre `attribution` a la priorité sur le paramètre obsolète `includeCoAuthoredBy`. Pour masquer toute attribution, définissez `commit` et `pr` à des chaînes vides.

Paramètres de suggestion de fichier

Configurez une commande personnalisée pour l'autocomplétion du chemin de fichier `@`. La suggestion de fichier intégrée utilise la traversée rapide du système de fichiers, mais les grands monorepos peuvent bénéficier d'une indexation spécifique au projet telle qu'un index de fichier pré-construit ou un outillage personnalisé.

```
{
  "fileSuggestion": {
    "type": "command",
    "command": "~/.claude/file-suggestion.sh"
  }
}
```

La commande s'exécute avec les mêmes variables d'environnement que les [hooks](#), y compris `CLAUDE_PROJECT_DIR`. Elle reçoit du JSON via stdin avec un champ `query` :

```
{"query": "src/comp"}
```

Générez les chemins de fichier séparés par des sauts de ligne vers stdout (actuellement limité à 15) :

```
src/components/Button.tsx
src/components/Modal.tsx
src/components/Form.tsx
```

Exemple :

```
#!/bin/bash
query=$(cat | jq -r '.query')
your-repo-file-index --query "$query" | head -20
```

Configuration des hooks

Ces paramètres contrôlent les hooks autorisés à s'exécuter et ce que les hooks HTTP peuvent accéder. Le paramètre `allowManagedHooksOnly` ne peut être configuré que dans les [paramètres gérés](#). Les listes blanches d'URL et de variables d'environnement peuvent être définies à n'importe quel niveau de paramètres et fusionnent entre les sources.

Comportement lorsque `allowManagedHooksOnly` est `true` :

- Les hooks gérés et les hooks SDK sont chargés
- Les hooks utilisateur, projet et plugin sont bloqués

Restreindre les URL des hooks HTTP :

Limitez les URL que les hooks HTTP peuvent cibler. Prend en charge `*` comme caractère générique pour la correspondance. Lorsque le tableau est défini, les hooks HTTP ciblant des URL non correspondantes sont silencieusement bloqués.

```
{
  "allowedHttpHookUrls": ["https://hooks.example.com/*", "http://localhost:*"]
}
```

Restreindre les variables d'environnement des hooks HTTP :

Limitez les noms de variables d'environnement que les hooks HTTP peuvent interpoler dans les valeurs d'en-tête. Le `allowedEnvVars` effectif de chaque hook est l'intersection de sa propre liste et de ce paramètre.

```
{
  "httpHookAllowedEnvVars": ["MY_TOKEN", "HOOK_SECRET"]
}
```

Précédence des paramètres

Les paramètres s'appliquent dans l'ordre de précedence. Du plus élevé au plus bas :

1. **Paramètres gérés** ([gérés par le serveur](#), [politiques MDM/au niveau du système d'exploitation](#), ou [paramètres gérés](#))
 - Politiques déployées par l'IT via la livraison par serveur, les profils de configuration MDM, les politiques de registre ou les fichiers de paramètres gérés
 - Ne peuvent pas être remplacés par aucun autre niveau, y compris les arguments de ligne de commande
 - Au sein du niveau géré, la précedence est : gérés par le serveur > politiques MDM/au niveau du système d'exploitation > `managed-settings.json` > registre HKCU (Windows uniquement). Une seule source gérée est utilisée ; les sources ne fusionnent pas.
2. **Arguments de ligne de commande**
 - Remplacements temporaires pour une session spécifique
3. **Paramètres du projet local** (`.claude/settings.local.json`)
 - Paramètres personnels spécifiques au projet

4. Paramètres du projet partagé (`.claude/settings.json`)

- Paramètres du projet partagés par l'équipe dans le contrôle de source

5. Paramètres utilisateur (`~/.claude/settings.json`)

- Paramètres globaux personnels

Cette hiérarchie garantit que les politiques organisationnelles sont toujours appliquées tout en permettant aux équipes et aux individus de personnaliser leur expérience.

Par exemple, si vos paramètres utilisateur autorisent `Bash(npm run *)` mais que les paramètres partagés d'un projet le refusent, le paramètre du projet a la priorité et la commande est bloquée.

Note:

Les paramètres de tableau fusionnent entre les portées. Lorsque le même paramètre à valeur de tableau (tel que `sandbox.filesystem.allowWrite` ou `permissions.allow`) apparaît dans plusieurs portées, les tableaux sont **concaténés et dédupliqués**, non remplacés. Cela signifie que les portées de priorité inférieure peuvent ajouter des entrées sans remplacer celles définies par les portées de priorité supérieure, et vice versa. Par exemple, si les paramètres gérés définissent `allowWrite` à `["//opt/company-tools"]` et qu'un utilisateur ajoute `["~/.kube"]`, les deux chemins sont inclus dans la configuration finale.

Vérifier les paramètres actifs

Exécutez `/status` dans Claude Code pour voir quelles sources de paramètres sont actives et d'où elles proviennent. La sortie affiche chaque couche de configuration (gérée, utilisateur, projet) ainsi que son origine, telle que `Enterprise managed settings (remote)`, `Enterprise managed settings (plist)`, `Enterprise managed settings (HKLM)` ou `Enterprise managed settings (file)`. Si un fichier de paramètres contient des erreurs, `/status` signale le problème afin que vous puissiez le corriger.

Points clés du système de configuration

- **Fichiers de mémoire (`CLAUDE.md`)** : Contiennent les instructions et le contexte que Claude charge au démarrage
- **Fichiers de paramètres (JSON)** : Configurez les permissions, les variables d'environnement et le comportement des outils
- **Skills** : Invites personnalisées qui peuvent être invoquées avec `/skill-name` ou chargées automatiquement par Claude
- **Serveurs MCP** : Étendez Claude Code avec des outils et des intégrations supplémentaires

- **Précédence** : Les configurations de niveau supérieur (Managed) remplacent celles de niveau inférieur (User/Project)
- **Héritage** : Les paramètres sont fusionnés, les paramètres plus spécifiques s'ajoutant à ou remplaçant les paramètres plus larges

Invite système

L'invite système interne de Claude Code n'est pas publiée. Pour ajouter des instructions personnalisées, utilisez les fichiers `CLAUDE.md` ou l'indicateur `--append-system-prompt`.

Exclure les fichiers sensibles

Pour empêcher Claude Code d'accéder aux fichiers contenant des informations sensibles comme les clés API, les secrets et les fichiers d'environnement, utilisez le paramètre `permissions.deny` dans votre fichier `.claude/settings.json` :

```
{
  "permissions": {
    "deny": [
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)",
      "Read(./config/credentials.json)",
      "Read(./build)"
    ]
  }
}
```

Cela remplace la configuration obsolète `ignorePatterns`. Les fichiers correspondant à ces modèles sont exclus de la découverte de fichiers et des résultats de recherche, et les opérations de lecture sur ces fichiers sont refusées.

Configuration des subagents

Claude Code prend en charge les subagents IA personnalisés qui peuvent être configurés aux niveaux utilisateur et projet. Ces subagents sont stockés sous forme de fichiers Markdown avec du frontmatter YAML :

- **Subagents utilisateur** : `~/.claude/agents/` - Disponibles sur tous vos projets
- **Subagents du projet** : `.claude/agents/` - Spécifiques à votre projet et peuvent être partagés avec votre équipe

Les fichiers de subagent définissent des assistants IA spécialisés avec des invites personnalisées et des permissions d'outils. En savoir plus sur la création et l'utilisation de subagents dans la [documentation des subagents](#).

Configuration des plugins

Claude Code prend en charge un système de plugins qui vous permet d'étendre les fonctionnalités avec des skills, des agents, des hooks et des serveurs MCP. Les plugins sont distribués via des marketplaces et peuvent être configurés aux niveaux utilisateur et référentiel.

Paramètres des plugins

Paramètres liés aux plugins dans `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`

Contrôle les plugins activés. Format : `"plugin-name@marketplace-name": true/false`

Portées :

- **Paramètres utilisateur** (`~/.claude/settings.json`) : Préférences personnelles des plugins
- **Paramètres du projet** (`.claude/settings.json`) : Plugins spécifiques au projet partagés avec l'équipe
- **Paramètres locaux** (`.claude/settings.local.json`) : Remplacements par machine (non commités)

Exemple :

```
{
  "enabledPlugins": {
    "code-formatter@team-tools": true,
    "deployment-tools@team-tools": true,
    "experimental-features@personal": false
  }
}
```

extraKnownMarketplaces

Définit les marketplaces supplémentaires qui doivent être mises à disposition pour le référentiel. Généralement utilisé dans les paramètres au niveau du référentiel pour s'assurer que les membres de l'équipe ont accès aux sources de plugins requises.

Lorsqu'un référentiel inclut `extraKnownMarketplaces` :

1. Les membres de l'équipe sont invités à installer la marketplace lorsqu'ils font confiance au dossier
2. Les membres de l'équipe sont ensuite invités à installer les plugins de cette marketplace
3. Les utilisateurs peuvent ignorer les marketplaces ou les plugins indésirables (stockés dans les paramètres utilisateur)
4. L'installation respecte les limites de confiance et nécessite un consentement explicite

Exemple :


```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/claude-plugins"
      }
    },
    "security-plugins": {
      "source": {
        "source": "git",
        "url": "https://git.example.com/security/plugins.git"
      }
    }
  }
}
```

Types de sources de marketplace :

- `github` : Référentiel GitHub (utilise `repo`)
- `git` : N'importe quelle URL git (utilise `url`)
- `directory` : Chemin du système de fichiers local (utilise `path` , pour le développement uniquement)
- `hostPattern` : Modèle regex pour correspondre aux hôtes de marketplace (utilise `hostPattern`)

`strictKnownMarketplaces`

Paramètres gérés uniquement : Contrôle les marketplaces de plugins que les utilisateurs sont autorisés à ajouter. Ce paramètre ne peut être configuré que dans les [paramètres gérés](#) et fournit aux administrateurs un contrôle strict sur les sources de marketplace.

Emplacements des fichiers de paramètres gérés :

- **macOS** : `/Library/Application Support/ClaudeCode/managed-settings.json`
- **Linux et WSL** : `/etc/claude-code/managed-settings.json`
- **Windows** : `C:\Program Files\ClaudeCode\managed-settings.json`

Caractéristiques clés :

- Disponible uniquement dans les paramètres gérés (`managed-settings.json`)
- Ne peut pas être remplacé par les paramètres utilisateur ou projet (priorité la plus élevée)
- Appliqué AVANT les opérations de réseau/système de fichiers (les sources bloquées ne s'exécutent jamais)
- Utilise la correspondance exacte pour les spécifications de source (y compris `ref` , `path` pour les sources git), sauf `hostPattern` , qui utilise la correspondance regex

Comportement de la liste blanche :

- `undefined` (par défaut) : Pas de restrictions - les utilisateurs peuvent ajouter n'importe quelle marketplace
- Tableau vide `[]` : Verrouillage complet - les utilisateurs ne peuvent pas ajouter de nouvelles marketplaces
- Liste de sources : Les utilisateurs ne peuvent ajouter que les marketplaces qui correspondent exactement

Tous les types de sources pris en charge :

La liste blanche prend en charge sept types de sources de marketplace. La plupart des sources utilisent la correspondance exacte, tandis que `hostPattern` utilise la correspondance regex par rapport à l'hôte de la marketplace.

1. Référentiels 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": "marketplace" }
```

Champs : `repo` (requis), `ref` (optionnel : branche/tag/SHA), `path` (optionnel : sous-répertoire)

1. Référentiels Git :

```
{ "source": "git", "url": "https://gitlab.example.com/tools/plugins.git" }
{ "source": "git", "url": "https://bitbucket.org/acme-corp/plugins.git", "ref": "production" }
{ "source": "git", "url": "ssh://git@git.example.com/plugins.git", "ref": "v3.1", "path": "approved" }
```

Champs : **url** (requis), **ref** (optionnel : branche/tag/SHA), **path** (optionnel : sous-répertoire)

1. Marketplaces basées sur URL :

```
{ "source": "url", "url": "https://plugins.example.com/marketplace.json" }
{ "source": "url", "url": "https://cdn.example.com/marketplace.json", "headers":
{ "Authorization": "Bearer ${TOKEN}" } }
```

Champs : **url** (requis), **headers** (optionnel : en-têtes HTTP pour l'accès authentifié)

Note:

Les marketplaces basées sur URL téléchargent uniquement le fichier **marketplace.json**. Elles ne téléchargent pas les fichiers de plugin à partir du serveur. Les plugins dans les marketplaces basées sur URL doivent utiliser des sources externes (URLs GitHub, npm ou git) plutôt que des chemins relatifs. Pour les plugins avec des chemins relatifs, utilisez une marketplace basée sur Git à la place. Voir [Dépannage](#) pour plus de détails.

1. Packages NPM :

```
{ "source": "npm", "package": "@acme-corp/claude-plugins" }
{ "source": "npm", "package": "@acme-corp/approved-marketplace" }
```

Champs : **package** (requis, prend en charge les packages à portée)

1. Chemins de fichier :

```
{ "source": "file", "path": "/usr/local/share/claude/acme-marketplace.json" }
{ "source": "file", "path": "/opt/acme-corp/plugins/marketplace.json" }
```

Champs : `path` (requis : chemin absolu vers le fichier `marketplace.json`)

1. Chemins de répertoire :

```
{ "source": "directory", "path": "/usr/local/share/claude/acme-plugins" }
{ "source": "directory", "path": "/opt/acme-corp/approved-marketplaces" }
```

Champs : `path` (requis : chemin absolu vers le répertoire contenant `.claude-plugin/marketplace.json`)

1. Correspondance de modèle d'hôte :

```
{ "source": "hostPattern", "hostPattern": "^github\\.example\\.com$" }
{ "source": "hostPattern", "hostPattern": "^gitlab\\.internal\\.example\\.com$" }
```

Champs : `hostPattern` (requis : modèle regex pour correspondre à l'hôte de la marketplace)

Utilisez la correspondance de modèle d'hôte lorsque vous souhaitez autoriser toutes les marketplaces d'un hôte spécifique sans énumérer chaque référentiel individuellement. Ceci est utile pour les organisations avec des serveurs GitHub Enterprise ou GitLab internes où les développeurs créent leurs propres marketplaces.

Extraction d'hôte par type de source :

- `github` : correspond toujours à `github.com`
- `git` : extrait le nom d'hôte de l'URL (prend en charge les formats HTTPS et SSH)
- `url` : extrait le nom d'hôte de l'URL
- `npm`, `file`, `directory` : non pris en charge pour la correspondance de modèle d'hôte

Exemples de configuration :

Exemple : autoriser uniquement les marketplaces spécifiques :

```
{
  "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"
    }
  ]
}
```

Exemple - Désactiver tous les ajouts de marketplace :

```
{
  "strictKnownMarketplaces": []
}
```

Exemple : autoriser toutes les marketplaces d'un serveur git interne :

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Exigences de correspondance exacte :

Les sources de marketplace doivent correspondre **exactement** pour que l’ajout d’un utilisateur soit autorisé. Pour les sources basées sur git (`github` et `git`), cela inclut tous les champs optionnels :

- Le `repo` ou `url` doit correspondre exactement
- Le champ `ref` doit correspondre exactement (ou les deux être non définis)
- Le champ `path` doit correspondre exactement (ou les deux être non définis)

Exemples de sources qui **ne correspondent PAS** :

```
// Ce sont des sources DIFFÉRENTES :
{ "source": "github", "repo": "acme-corp/plugins" }
{ "source": "github", "repo": "acme-corp/plugins", "ref": "main" }

// Ce sont aussi DIFFÉRENTES :
{ "source": "github", "repo": "acme-corp/plugins", "path": "marketplace" }
{ "source": "github", "repo": "acme-corp/plugins" }
```

Comparaison avec `extraKnownMarketplaces` :

Aspect	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Objectif	Application de la politique organisationnelle	Commodité de l’équipe
Fichier de paramètres	<code>managed-settings.json</code> uniquement	N’importe quel fichier de paramètres

Aspect	<code>strictKnownMarketplaces</code>	<code>extraKnownMarketplaces</code>
Comportement	Bloque les ajouts non autorisés	Auto-installe les marketplaces manquantes
Quand appliqué	Avant les opérations de réseau/ système de fichiers	Après l'invite de confiance de l'utilisateur
Peut être remplacé	Non (priorité la plus élevée)	Oui (par les paramètres de priorité supérieure)
Format de source	Objet source direct	Marketplace nommée avec source imbriquée
Cas d'utilisation	Conformité, restrictions de sécurité	Intégration, standardisation

Différence de format :

`strictKnownMarketplaces` utilise des objets source directs :

```
{
  "strictKnownMarketplaces": [
    { "source": "github", "repo": "acme-corp/plugins" }
  ]
}
```

`extraKnownMarketplaces` nécessite des marketplaces nommées :

```
{
  "extraKnownMarketplaces": {
    "acme-tools": {
      "source": { "source": "github", "repo": "acme-corp/plugins" }
    }
  }
}
```

Notes importantes :

- Les restrictions sont vérifiées AVANT toute demande réseau ou opération du système de fichiers

- Lorsqu'elles sont bloquées, les utilisateurs voient des messages d'erreur clairs indiquant que la source est bloquée par la politique gérée
- La restriction s'applique uniquement à l'ajout de NOUVELLES marketplaces ; les marketplaces précédemment installées restent accessibles
- Les paramètres gérés ont la priorité la plus élevée et ne peuvent pas être remplacés

Voir [Restrictions de marketplace gérées](#) pour la documentation destinée aux utilisateurs.

Gestion des plugins

Utilisez la commande `/plugin` pour gérer les plugins de manière interactive :

- Parcourir les plugins disponibles à partir des marketplaces
- Installer/désinstaller les plugins
- Activer/désactiver les plugins
- Afficher les détails du plugin (commandes, agents, hooks fournis)
- Ajouter/supprimer les marketplaces

En savoir plus sur le système de plugins dans la [documentation des plugins](#).

Variables d'environnement

Claude Code prend en charge les variables d'environnement suivantes pour contrôler son comportement :

Note:

Toutes les variables d'environnement peuvent également être configurées dans `settings.json`. Ceci est utile comme moyen de définir automatiquement les variables d'environnement pour chaque session, ou de déployer un ensemble de variables d'environnement pour toute votre équipe ou organisation.

Variable	Objectif	
<code>ANTHROPIC_API_KEY</code>	Clé API envoyée comme en-tête <code>X-Api-Key</code> , généralement pour le SDK Claude (pour l'utilisation interactive, exécutez <code>/login</code>)	

Variable	Objectif	
<code>ANTHROPIC_AUTH_TOKEN</code>	Valeur personnalisée pour l'en-tête <code>Authorization</code> (la valeur que vous définissez ici sera préfixée avec <code>Bearer</code>)	
<code>ANTHROPIC_CUSTOM_HEADERS</code>	En-têtes personnalisés à ajouter aux demandes (format <code>Name: Value</code> , séparés par des sauts de ligne pour plusieurs en-têtes)	
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Voir Configuration du modèle	
<code>ANTHROPIC_DEFAULT_OPENAI_MODEL</code>	Voir Configuration du modèle	
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Voir Configuration du modèle	
<code>ANTHROPIC_FOUNDRY_API_KEY</code>	Clé API pour l'authentification Microsoft Foundry (voir Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_BASE_URL</code>	URL de base complète pour la ressource Foundry (par exemple, <code>https://my-resource.services.ai.azure.com/anthropic</code>). Alternative à <code>ANTHROPIC_FOUNDRY_RESOURCE</code> (voir Microsoft Foundry)	
<code>ANTHROPIC_FOUNDRY_RESOURCE</code>	Nom de la ressource Foundry (par exemple, <code>my-resource</code>). Requis si <code>ANTHROPIC_FOUNDRY_BASE_URL</code> n'est pas défini (voir Microsoft Foundry)	
<code>ANTHROPIC_MODEL</code>	Nom du paramètre de modèle à utiliser (voir Configuration du modèle)	
<code>ANTHROPIC_SMALL_FAST_MODEL</code>	[OBSOLÈTE] Nom du modèle de classe Haiku pour les tâches de fond	

Variable	Objectif	
<code>ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION</code>	Remplacer la région AWS pour le modèle de classe Haiku lors de l'utilisation de Bedrock	
<code>AWS_BEARER_TOKEN_BEDROCK</code>	Clé API Bedrock pour l'authentification (voir Clés API Bedrock)	
<code>BASH_DEFAULT_TIMEOUT_MS</code>	Délai d'expiration par défaut pour les commandes bash longues	
<code>BASH_MAX_OUTPUT_LENGTH</code>	Nombre maximum de caractères dans les sorties bash avant qu'elles ne soient tronquées au milieu	
<code>BASH_MAX_TIMEOUT_MS</code>	Délai d'expiration maximal que le modèle peut définir pour les commandes bash longues	
<code>CLAUDE_AUTOCOMPACT_PERCENT_OVERRIDE</code>	Définissez le pourcentage de capacité de contexte (1-100) auquel la compaction automatique se déclenche. Par défaut, la compaction automatique se déclenche à environ 95 % de capacité. Utilisez des valeurs inférieures comme <code>50</code> pour compacter plus tôt. Les valeurs au-dessus du seuil par défaut n'ont aucun effet. S'applique aux conversations principales et aux subagents. Ce pourcentage s'aligne avec le champ <code>context_window.used_percentage</code> disponible dans la ligne d'état	
<code>CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR</code>	Retourner au répertoire de travail d'origine après chaque commande Bash	

Variable	Objectif	
<code>CLAUDE_CODE_ACCOUNT_UUID</code>	UUID du compte pour l'utilisateur authentifié. Utilisé par les appelants du SDK pour fournir les informations du compte de manière synchrone, évitant une condition de course où les événements de télémétrie précoces manquent de métadonnées de compte. Nécessite que <code>CLAUDE_CODE_USER_EMAIL</code> et <code>CLAUDE_CODE_ORGANIZATION_UUID</code> soient également définis	
<code>CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD</code>	Définissez à <code>1</code> pour charger les fichiers CLAUDE.md à partir des répertoires spécifiés avec <code>--add-dir</code> . Par défaut, les répertoires supplémentaires ne chargent pas les fichiers de mémoire	<code>1</code>
<code>CLAUDE_CODE_API_KEY_HELPER_TTL_MS</code>	Intervalle en millisecondes auquel les identifiants doivent être actualisés (lors de l'utilisation de <code>apiKeyHelper</code>)	
<code>CLAUDE_CODE_CLIENT_CERT</code>	Chemin vers le fichier de certificat client pour l'authentification mTLS	
<code>CLAUDE_CODE_CLIENT_KEY</code>	Chemin vers le fichier de clé privée client pour l'authentification mTLS	
<code>CLAUDE_CODE_CLIENT_KEY_PASSPHRASE</code>	Phrase de passe pour <code>CLAUDE_CODE_CLIENT_KEY</code> chiffré (optionnel)	
<code>CLAUDE_CODE_DISABLE_1M_CONTEXT</code>	Définissez à <code>1</code> pour désactiver le support de la fenêtre de contexte 1M . Lorsqu'elle est définie, les variantes de modèle 1M ne sont pas disponibles dans le sélecteur de modèle. Utile pour les environnements d'entreprise avec des exigences de conformité	

Variable	Objectif	
<code>CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING</code>	Définissez à <code>1</code> pour désactiver le raisonnement adaptatif pour Opus 4.6 et Sonnet 4.6. Lorsqu'elle est désactivée, ces modèles reviennent au budget de réflexion fixe contrôlé par <code>MAX_THINKING_TOKENS</code> .	
<code>CLAUDE_CODE_DISABLE_AUTO_MEMORY</code>	Définissez à <code>1</code> pour désactiver la mémoire automatique . Définissez à <code>0</code> pour forcer la mémoire automatique pendant le déploiement progressif. Lorsqu'elle est désactivée, Claude ne crée ni ne charge les fichiers de mémoire automatique.	
<code>CLAUDE_CODE_DISABLE_GIT_INSTRUCTIONS</code>	Définissez à <code>1</code> pour supprimer les instructions de workflow de commit et de PR intégrées de l'invite système de Claude. Utile lors de l'utilisation de vos propres skills de workflow git. A la priorité sur le paramètre <code>includeGitInstructions</code> lorsqu'elle est définie.	
<code>CLAUDE_CODE_DISABLE_BACKGROUND_TASKS</code>	Définissez à <code>1</code> pour désactiver toute la fonctionnalité de tâche de fond, y compris le paramètre <code>run_in_background</code> sur les outils Bash et subagent, l'auto-backgrounding et le raccourci Ctrl+B.	
<code>CLAUDE_CODE_DISABLE_CRON</code>	Définissez à <code>1</code> pour désactiver les tâches planifiées . Le skill <code>/loop</code> et les outils cron deviennent indisponibles et toutes les tâches planifiées existantes cessent de se déclencher, y compris les tâches qui s'exécutent déjà en milieu de session.	

Variable	Objectif	
<code>CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS</code>	Définissez à <code>1</code> pour désactiver les en-têtes <code>anthropic-beta</code> spécifiques à l'API Anthropic. Utilisez ceci si vous rencontrez des problèmes comme « Unexpected value(s) for the <code>anthropic-beta</code> header » lors de l'utilisation d'une passerelle LLM avec des fournisseurs tiers	
<code>CLAUDE_CODE_DISABLE_FAST_MODE</code>	Définissez à <code>1</code> pour désactiver le mode rapide	
<code>CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY</code>	Définissez à <code>1</code> pour désactiver les sondages de qualité de session « How is Claude doing? ». Également désactivé lors de l'utilisation de fournisseurs tiers ou lorsque la télémétrie est désactivée. Voir Sondages de qualité de session	
<code>CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC</code>	Équivalent de la définition de <code>DISABLE_AUTOUPDATER</code> , <code>DISABLE_BUG_COMMAND</code> , <code>DISABLE_ERROR_REPORTING</code> et <code>DISABLE_TELEMETRY</code>	
<code>CLAUDE_CODE_DISABLE_TERMINAL_TITLE</code>	Définissez à <code>1</code> pour désactiver les mises à jour automatiques du titre du terminal en fonction du contexte de la conversation	
<code>CLAUDE_CODE_EFFORT_LEVEL</code>	Définissez le niveau d'effort pour les modèles pris en charge. Valeurs : <code>low</code> , <code>medium</code> , <code>high</code> . L'effort inférieur est plus rapide et moins cher, l'effort supérieur fournit un raisonnement plus profond. Pris en charge sur Opus 4.6 et Sonnet 4.6. Voir Ajuster le niveau d'effort	

Variable	Objectif	
<code>CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION</code>	Définissez à <code>false</code> pour désactiver les suggestions de prompt (le bouton bascule « Prompt suggestions » dans <code>/config</code>). Ce sont les prédictions grisées qui apparaissent dans votre entrée de prompt après que Claude réponde. Voir Suggestions de prompt	
<code>CLAUDE_CODE_ENABLE_TASKS</code>	Définissez à <code>false</code> pour revenir temporairement à la liste TODO précédente au lieu du système de suivi des tâches. Par défaut : <code>true</code> . Voir Liste des tâches	
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	Définissez à <code>1</code> pour activer la collecte de données OpenTelemetry pour les métriques et la journalisation. Requis avant de configurer les exportateurs OTel. Voir Surveillance	
<code>CLAUDE_CODE_EXIT_AFTER_STOP_DELAY</code>	Temps en millisecondes à attendre après que la boucle de requête devienne inactive avant de quitter automatiquement. Utile pour les flux de travail automatisés et les scripts utilisant le mode SDK	
<code>CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS</code>	Définissez à <code>1</code> pour activer les équipes d'agents . Les équipes d'agents sont expérimentales et désactivées par défaut	
<code>CLAUDE_CODE_FILE_READ_MAX_OUTPUT_TOKENS</code>	Remplacer la limite de jetons par défaut pour les lectures de fichier. Utile lorsque vous devez lire des fichiers plus volumineux en intégralité	
<code>CLAUDE_CODE_HIDE_ACCOUNT_INFO</code>	Définissez à <code>1</code> pour masquer votre adresse e-mail et le nom de votre organisation de l'interface utilisateur de Claude Code. Utile lors de la diffusion en continu ou de l'enregistrement	

Variable	Objectif	
<code>CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL</code>	Ignorer l'installation automatique des extensions IDE	
<code>CLAUDE_CODE_MAX_OUTPUT_TOKENS</code>	Définissez le nombre maximal de jetons de sortie pour la plupart des demandes. Par défaut : 32 000. Maximum : 64 000. L'augmentation de cette valeur réduit la fenêtre de contexte effective disponible avant que la compaction automatique ne se déclenche.	
<code>CLAUDE_CODE_ORGANIZATION_UUID</code>	UUID de l'organisation pour l'utilisateur authentifié. Utilisé par les appelants du SDK pour fournir les informations du compte de manière synchrone. Nécessite que <code>CLAUDE_CODE_ACCOUNT_UUID</code> et <code>CLAUDE_CODE_USER_EMAIL</code> soient également définis	
<code>CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS</code>	Intervalle pour actualiser les en-têtes OpenTelemetry dynamiques en millisecondes (par défaut : 1740000 / 29 minutes). Voir En-têtes dynamiques	
<code>CLAUDE_CODE_PLAN_MODE_REQUIRED</code>	Auto-défini à <code>true</code> sur les coéquipiers de l' équipe d'agents qui nécessitent l'approbation du plan. Lecture seule : défini par Claude Code lors du lancement des coéquipiers. Voir exiger l'approbation du plan	
<code>CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS</code>	Délai d'expiration en millisecondes pour les opérations git lors de l'installation ou de la mise à jour des plugins (par défaut : 120000). Augmentez cette valeur pour les grands référentiels ou les connexions réseau lentes. Voir Les opérations Git expirent	

Variable	Objectif	
<code>CLAUDE_CODE_PROXY_RESOLVES_HOSTS</code>	Définissez à <code>true</code> pour permettre au proxy d'effectuer la résolution DNS au lieu de l'appelant. Opt-in pour les environnements où le proxy doit gérer la résolution du nom d'hôte	
<code>CLAUDE_CODE_SHELL</code>	Remplacer la détection automatique du shell. Utile lorsque votre shell de connexion diffère de votre shell de travail préféré (par exemple, <code>bash</code> vs <code>zsh</code>)	
<code>CLAUDE_CODE_SHELL_PREFIX</code>	Préfixe de commande pour envelopper toutes les commandes bash (par exemple, pour la journalisation ou l'audit). Exemple : <code>/path/to/logger.sh</code> exécutera <code>/path/to/logger.sh <command></code>	
<code>CLAUDE_CODE_SIMPLE</code>	Définissez à <code>1</code> pour exécuter avec une invite système minimale et uniquement les outils Bash, lecture de fichier et édition de fichier. Désactive les outils MCP, les pièces jointes, les hooks et les fichiers CLAUDE.md	
<code>CLAUDE_CODE_SKIP_BEDROCK_AUTH</code>	Ignorer l'authentification AWS pour Bedrock (par exemple, lors de l'utilisation d'une passerelle LLM)	
<code>CLAUDE_CODE_SKIP_FOUNDRY_AUTH</code>	Ignorer l'authentification Azure pour Microsoft Foundry (par exemple, lors de l'utilisation d'une passerelle LLM)	
<code>CLAUDE_CODE_SKIP_VERTEX_AUTH</code>	Ignorer l'authentification Google pour Vertex (par exemple, lors de l'utilisation d'une passerelle LLM)	
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	Voir Configuration du modèle	

Variable	Objectif	
<code>CLAUDE_CODE_TASK_LIST_ID</code>	Partager une liste de tâches entre les sessions. Définissez le même ID dans plusieurs instances de Claude Code pour coordonner une liste de tâches partagée. Voir Liste des tâches	
<code>CLAUDE_CODE_TEAM_NAME</code>	Nom de l'équipe d'agents à laquelle appartient ce coéquipier. Défini automatiquement sur les membres de l' équipe d'agents	
<code>CLAUDE_CODE_TMPDIR</code>	Remplacer le répertoire temporaire utilisé pour les fichiers temporaires internes. Claude Code ajoute <code>/claude/</code> à ce chemin. Par défaut : <code>/tmp</code> sur Unix/macOS, <code>os.tmpdir()</code> sur Windows	
<code>CLAUDE_CODE_USER_EMAIL</code>	Adresse e-mail pour l'utilisateur authentifié. Utilisé par les appelants du SDK pour fournir les informations du compte de manière synchrone. Nécessite que <code>CLAUDE_CODE_ACCOUNT_UUID</code> et <code>CLAUDE_CODE_ORGANIZATION_UUID</code> soient également définis	
<code>CLAUDE_CODE_USE_BEDROCK</code>	Utiliser Bedrock	
<code>CLAUDE_CODE_USE_FOUNDRY</code>	Utiliser Microsoft Foundry	
<code>CLAUDE_CODE_USE_VERTEX</code>	Utiliser Vertex	
<code>CLAUDE_CONFIG_DIR</code>	Personnalisez où Claude Code stocke ses fichiers de configuration et de données	
<code>DISABLE_AUTOUPDATER</code>	Définissez à <code>1</code> pour désactiver les mises à jour automatiques.	

Variable	Objectif	
<code>DISABLE_BUG_COMMAND</code>	Définissez à <code>1</code> pour désactiver la commande <code>/bug</code>	
<code>DISABLE_COST_WARNING</code>	Définissez à <code>1</code> pour désactiver les messages d'avertissement de coût	
<code>DISABLE_ERROR_REPORTING</code>	Définissez à <code>1</code> pour refuser la création de rapports d'erreur Sentry	
<code>DISABLE_INSTALLATION_CHECKS</code>	Définissez à <code>1</code> pour désactiver les avertissements d'installation. À utiliser uniquement lors de la gestion manuelle de l'emplacement d'installation, car cela peut masquer les problèmes avec les installations standard	
<code>DISABLE_NON_ESSENTIAL_MODEL_CALLS</code>	Définissez à <code>1</code> pour désactiver les appels de modèle pour les chemins non critiques comme le texte de sauveur	
<code>DISABLE_PROMPT_CACHING</code>	Définissez à <code>1</code> pour désactiver la mise en cache des prompts pour tous les modèles (à la priorité sur les paramètres par modèle)	
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Définissez à <code>1</code> pour désactiver la mise en cache des prompts pour les modèles Haiku	
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Définissez à <code>1</code> pour désactiver la mise en cache des prompts pour les modèles Opus	
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Définissez à <code>1</code> pour désactiver la mise en cache des prompts pour les modèles Sonnet	

Variable	Objectif	
<code>DISABLE_TELEMETRY</code>	Définissez à <code>1</code> pour refuser la télé-métrie Statsig (notez que les événements Statsig n'incluent pas les données utilisateur comme le code, les chemins de fichier ou les commandes bash)	
<code>ENABLE_CLAUDEAI_MCP_SERVERS</code>	Définissez à <code>false</code> pour désactiver les serveurs MCP claude.ai dans Claude Code. Activé par défaut pour les utilisateurs connectés	
<code>ENABLE_TOOL_SEARCH</code>	Contrôle la recherche d'outils MCP . Valeurs : <code>auto</code> (par défaut, active à 10 % du contexte), <code>auto:N</code> (seuil personnalisé, par exemple, <code>auto:5</code> pour 5 %), <code>true</code> (toujours activé), <code>false</code> (désactivé)	
<code>FORCE_AUTOUPDATE_PLUGINS</code>	Définissez à <code>true</code> pour forcer les mises à jour automatiques des plugins même lorsque la mise à jour automatique principale est désactivée via <code>DISABLE_AUTOUPDATER</code>	
<code>HTTP_PROXY</code>	Spécifiez le serveur proxy HTTP pour les connexions réseau	
<code>HTTPS_PROXY</code>	Spécifiez le serveur proxy HTTPS pour les connexions réseau	
<code>IS_DEMO</code>	Définissez à <code>true</code> pour activer le mode démo : masque l'e-mail et l'organisation de l'interface utilisateur, ignore l'intégration et masque les commandes internes. Utile pour la diffusion en continu ou l'enregistrement de sessions	

Variable	Objectif	
<code>MAX_MCP_OUTPUT_TOKENS</code>	Nombre maximal de jetons autorisés dans les réponses des outils MCP. Claude Code affiche un avertissement lorsque la sortie dépasse 10 000 jetons (par défaut : 25000)	
<code>MAX_THINKING_TOKENS</code>	Remplacer le budget de réflexion étendue . La réflexion est activée au budget maximal (31 999 jetons) par défaut. Utilisez ceci pour limiter le budget (par exemple, <code>MAX_THINKING_TOKENS=10000</code>) ou désactiver complètement la réflexion (<code>MAX_THINKING_TOKENS=0</code>). Pour Opus 4.6, la profondeur de réflexion est contrôlée par le niveau d'effort à la place, et cette variable est ignorée sauf si elle est définie à <code>0</code> pour désactiver la réflexion.	
<code>MCP_CLIENT_SECRET</code>	Secret client OAuth pour les serveurs MCP qui nécessitent des identifiants pré-configurés . Évite l'invite interactive lors de l'ajout d'un serveur avec <code>--client-secret</code>	
<code>MCP_OAUTH_CALLBACK_PORT</code>	Port fixe pour le rappel de redirection OAuth, comme alternative à <code>--callback-port</code> lors de l'ajout d'un serveur MCP avec des identifiants pré-configurés	
<code>MCP_TIMEOUT</code>	Délai d'expiration en millisecondes pour le démarrage du serveur MCP	
<code>MCP_TOOL_TIMEOUT</code>	Délai d'expiration en millisecondes pour l'exécution de l'outil MCP	
<code>NO_PROXY</code>	Liste des domaines et adresses IP vers lesquels les demandes seront émises directement, en contournant le proxy	

Variable	Objectif	
<code>SLASH_COMMAND_TOOL_CHARACTER_BUDGET</code>	Remplacer le budget de caractères pour les métadonnées de skill affichées à l' outil Skill . Le budget s'adapte dynamiquement à 2 % de la fenêtre de contexte, avec un repli de 16 000 caractères. Nom hérité conservé pour la compatibilité rétro-active	
<code>USE_BUILTIN_RIPGREP</code>	Définissez à <code>0</code> pour utiliser <code>rg</code> installé sur le système au lieu de <code>rg</code> inclus avec Claude Code	
<code>VERTEX_REGION_CLAUDE_3_5_HAIKU</code>	Remplacer la région pour Claude 3.5 Haiku lors de l'utilisation de Vertex AI	
<code>VERTEX_REGION_CLAUDE_3_7_SONNET</code>	Remplacer la région pour Claude 3.7 Sonnet lors de l'utilisation de Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_OPUS</code>	Remplacer la région pour Claude 4.0 Opus lors de l'utilisation de Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_0_SONNET</code>	Remplacer la région pour Claude 4.0 Sonnet lors de l'utilisation de Vertex AI	
<code>VERTEX_REGION_CLAUDE_4_1_OPUS</code>	Remplacer la région pour Claude 4.1 Opus lors de l'utilisation de Vertex AI	

Outils disponibles pour Claude

Claude Code a accès à un ensemble d'outils puissants qui l'aident à comprendre et à modifier votre base de code :

Outil	Description	Permission requise
Agent	Génère un subagent avec sa propre fenêtre de contexte pour gérer une tâche	Non

Outil	Description	Permission requise
AskUserQuestion	Pose des questions à choix multiples pour recueillir les exigences ou clarifier l'ambiguïté	Non
Bash	Exécute les commandes shell dans votre environnement. Voir Comportement de l'outil Bash	Oui
CronCreate	Planifie une invite récurrente ou unique dans la session actuelle (disparaît lorsque Claude quitte). Voir tâches planifiées	Non
CronDelete	Annule une tâche planifiée par ID	Non
CronList	Liste toutes les tâches planifiées dans la session	Non
Edit	Effectue des modifications ciblées sur des fichiers spécifiques	Oui
EnterPlanMode	Bascule en mode plan pour concevoir une approche avant de coder	Non
EnterWorktree	Crée un git worktree isolé et y bascule	Non
ExitPlanMode	Présente un plan pour approbation et quitte le mode plan	Oui
ExitWorktree	Quitte une session worktree et retourne au répertoire d'origine	Non
Glob	Trouve les fichiers en fonction de la correspondance de modèles	Non
Grep	Recherche les modèles dans le contenu des fichiers	Non
ListMcpResourcesTool	Liste les ressources exposées par les serveurs MCP connectés	Non

Outil	Description	Permission requise
LSP	Intelligence du code via les serveurs de langage. Signale automatiquement les erreurs de type et les avertissements après les modifications de fichier. Prend également en charge les opérations de navigation : aller aux définitions, trouver les références, obtenir les informations de type, lister les symboles, trouver les implémentations, tracer les hiérarchies d'appels. Nécessite un plugin d'intelligence du code et son binaire de serveur de langage	Non
NotebookEdit	Modifie les cellules du notebook Jupyter	Oui
Read	Lit le contenu des fichiers	Non
ReadMcpResourceTool	Lit une ressource MCP spécifique par URI	Non
Skill	Exécute un skill dans la conversation principale	Oui
TaskCreate	Crée une nouvelle tâche dans la liste des tâches	Non
TaskGet	Récupère les détails complets d'une tâche spécifique	Non
TaskList	Liste toutes les tâches avec leur statut actuel	Non
TaskOutput	Récupère la sortie d'une tâche de fond	Non
TaskStop	Tue une tâche de fond en cours d'exécution par ID	Non
TaskUpdate	Met à jour le statut, les dépendances, les détails de la tâche ou supprime les tâches	Non

Outil	Description	Permission requise
TodoWrite	Gère la liste de contrôle des tâches de session. Disponible en mode non interactif et dans le SDK Agent ; les sessions interactives utilisent TaskCreate, TaskGet, TaskList et TaskUpdate à la place	Non
ToolSearch	Recherche et charge les outils différés lorsque la recherche d'outils est activée	Non
WebFetch	Récupère le contenu d'une URL spécifiée	Oui
WebSearch	Effectue des recherches sur le web	Oui
Write	Crée ou remplace les fichiers	Oui

Les règles de permission peuvent être configurées en utilisant `/allowed-tools` ou dans les [paramètres de permission](#). Voir aussi [Règles de permission spécifiques aux outils](#).

Comportement de l'outil Bash

L'outil Bash exécute les commandes shell avec le comportement de persistance suivant :

- **Le répertoire de travail persiste** : Lorsque Claude change le répertoire de travail (par exemple, `cd /path/to/dir`), les commandes Bash suivantes s'exécutent dans ce répertoire. Vous pouvez utiliser `CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR=1` pour réinitialiser le répertoire du projet après chaque commande.
- **Les variables d'environnement ne persistent PAS** : Les variables d'environnement définies dans une commande Bash (par exemple, `export MY_VAR=value`) ne sont **pas** disponibles dans les commandes Bash suivantes. Chaque commande Bash s'exécute dans un environnement shell frais.

Pour rendre les variables d'environnement disponibles dans les commandes Bash, vous avez **trois options** :

Option 1 : Activer l'environnement avant de démarrer Claude Code (approche la plus simple)

Activez votre environnement virtuel dans votre terminal avant de lancer Claude Code :


```
conda activate myenv
## ou : source /path/to/venv/bin/activate
claude
```

Cela fonctionne pour les environnements shell, mais les variables d'environnement définies dans les commandes Bash de Claude ne persisteront pas entre les commandes.

Option 2 : Définir `CLAUDE_ENV_FILE` avant de démarrer Claude Code (configuration d'environnement persistante)

Exportez le chemin vers un script shell contenant votre configuration d'environnement :

```
export CLAUDE_ENV_FILE=/path/to/env-setup.sh
claude
```

Où `/path/to/env-setup.sh` contient :

```
conda activate myenv
## ou : source /path/to/venv/bin/activate
## ou : export MY_VAR=value
```

Claude Code sourcera ce fichier avant chaque commande Bash, rendant l'environnement persistant sur toutes les commandes.

Option 3 : Utiliser un hook `SessionStart` (configuration spécifique au projet)

Configurez dans `.claude/settings.json` :

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "startup",
      "hooks": [{
        "type": "command",
        "command": "echo 'conda activate myenv' >> \"\$CLAUDE_ENV_FILE\""
      }]
    }]
  }
}
```

Le hook écrit dans `$CLAUDE_ENV_FILE`, qui est ensuite sourcé avant chaque commande Bash. Ceci est idéal pour les configurations de projet partagées par l'équipe.

Voir [Hooks SessionStart](#) pour plus de détails sur l'option 3.

Extension des outils avec des hooks

Vous pouvez exécuter des commandes personnalisées avant ou après l'exécution de n'importe quel outil en utilisant les [hooks Claude Code](#).

Par exemple, vous pourriez exécuter automatiquement un formateur Python après que Claude modifie les fichiers Python, ou empêcher les modifications des fichiers de configuration de production en bloquant les opérations Write vers certains chemins.

Voir aussi

- [Permissions](#) : système de permissions, syntaxe des règles, modèles spécifiques aux outils et politiques gérées
- [Authentification](#) : configurer l'accès utilisateur à Claude Code
- [Dépannage](#) : solutions pour les problèmes de configuration courants

Configuration du modèle

Découvrez la configuration du modèle Claude Code, y compris les alias de modèle comme `opusplan`

Modèles disponibles

Pour le paramètre `model` dans Claude Code, vous pouvez configurer l'un des éléments suivants :

- Un **alias de modèle**
- Un **nom de modèle**
 - API Anthropic : un [nom de modèle](#) complet
 - Bedrock : un ARN de profil d'inférence
 - Foundry : un nom de déploiement
 - Vertex : un nom de version

Alias de modèle

Les alias de modèle offrent un moyen pratique de sélectionner les paramètres du modèle sans avoir à mémoriser les numéros de version exacts :

Alias de modèle	Comportement
<code>default</code>	Paramètre de modèle recommandé, selon votre type de compte
<code>sonnet</code>	Utilise le dernier modèle Sonnet (actuellement Sonnet 4.6) pour les tâches de codage quotidiennes
<code>opus</code>	Utilise le dernier modèle Opus (actuellement Opus 4.6) pour les tâches de raisonnement complexe
<code>haiku</code>	Utilise le modèle Haiku rapide et efficace pour les tâches simples
<code>sonnet[1m]</code>	Utilise Sonnet avec une fenêtre de contexte de 1 million de tokens pour les sessions longues
<code>opusplan</code>	Mode spécial qui utilise <code>opus</code> pendant le mode plan, puis bascule vers <code>sonnet</code> pour l'exécution

Les alias pointent toujours vers la dernière version. Pour épingler une version spécifique, utilisez le nom de modèle complet (par exemple, `claude-opus-4-6`) ou définissez la variable d'environnement correspondante comme `ANTHROPIC_DEFAULT_OPUS_MODEL`.

Définir votre modèle

Vous pouvez configurer votre modèle de plusieurs façons, énumérées par ordre de priorité :

1. **Pendant la session** - Utilisez `/model <alias|name>` pour basculer les modèles en cours de session
2. **Au démarrage** - Lancez avec `claude --model <alias|name>`
3. **Variable d'environnement** - Définissez `ANTHROPIC_MODEL=<alias|name>`
4. **Paramètres** - Configurez de manière permanente dans votre fichier de paramètres en utilisant le champ `model`.

Exemple d'utilisation :

```
## Démarrer avec Opus
claude --model opus

## Basculer vers Sonnet pendant la session
/model sonnet
```

Exemple de fichier de paramètres :

```
{
  "permissions": {
    ...
  },
  "model": "opus"
}
```

Restreindre la sélection du modèle

Les administrateurs d'entreprise peuvent utiliser `availableModels` dans les [paramètres gérés ou de politique](#) pour restreindre les modèles que les utilisateurs peuvent sélectionner.

Lorsque `availableModels` est défini, les utilisateurs ne peuvent pas basculer vers des modèles ne figurant pas dans la liste via `/model`, le drapeau `--model`, l'outil Config ou la variable d'environnement `ANTHROPIC_MODEL`.

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

Comportement du modèle par défaut

L'option Par défaut dans le sélecteur de modèle n'est pas affectée par `availableModels`. Elle reste toujours disponible et représente la valeur par défaut du runtime du système [basée sur le niveau d'abonnement de l'utilisateur](#).

Même avec `availableModels: []`, les utilisateurs peuvent toujours utiliser Claude Code avec le modèle Par défaut pour leur niveau.

Contrôler le modèle sur lequel les utilisateurs s'exécutent

Pour contrôler complètement l'expérience du modèle, utilisez `availableModels` avec le paramètre `model` :

- **availableModels** : restreint ce vers quoi les utilisateurs peuvent basculer
- **model** : définit le remplacement de modèle explicite, prenant la priorité sur le Par défaut

Cet exemple garantit que tous les utilisateurs exécutent Sonnet 4.6 et ne peuvent choisir qu'entre Sonnet et Haiku :

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku"]
}
```

Comportement de fusion

Lorsque `availableModels` est défini à plusieurs niveaux, comme les paramètres utilisateur et les paramètres de projet, les tableaux sont fusionnés et dédoublés. Pour appliquer une liste d'autorisation stricte, définissez `availableModels` dans les paramètres gérés ou de politique qui ont la priorité la plus élevée.

Comportement spécial du modèle

Paramètre de modèle `default`

Le comportement de `default` dépend de votre type de compte :

- **Max et Team Premium** : par défaut Opus 4.6
- **Pro et Team Standard** : par défaut Sonnet 4.6
- **Enterprise** : Opus 4.6 est disponible mais pas par défaut

Claude Code peut automatiquement revenir à Sonnet si vous atteignez un seuil d'utilisation avec Opus.

Paramètre de modèle `opuslan`

L'alias de modèle `opuslan` fournit une approche hybride automatisée :

- **En mode plan** - Utilise `opus` pour le raisonnement complexe et les décisions architecturales
- **En mode exécution** - Bascule automatiquement vers `sonnet` pour la génération de code et l'implémentation

Cela vous donne le meilleur des deux mondes : le raisonnement supérieur d'Opus pour la planification et l'efficacité de Sonnet pour l'exécution.

Ajuster le niveau d'effort

Les [niveaux d'effort](#) contrôlent le raisonnement adaptatif, qui alloue dynamiquement la réflexion en fonction de la complexité de la tâche. Un effort inférieur est plus rapide et moins cher pour les tâches simples, tandis qu'un effort supérieur fournit un raisonnement plus approfondi pour les problèmes complexes.

Trois niveaux sont disponibles : **low**, **medium** et **high**. Opus 4.6 utilise par défaut un effort moyen pour les abonnés Max et Team.

Définir l'effort :

- **Dans `/model`** : utilisez les touches fléchées gauche/droite pour ajuster le curseur d'effort lors de la sélection d'un modèle
- **Variable d'environnement** : définissez `CLAUDE_CODE_EFFORT_LEVEL=low|medium|high`
- **Paramètres** : définissez `effortLevel` dans votre fichier de paramètres

L'effort est pris en charge sur Opus 4.6 et Sonnet 4.6. Le curseur d'effort apparaît dans `/model` lorsqu'un modèle pris en charge est sélectionné. Le niveau d'effort actuel est également affiché à côté du logo et du spinner (par exemple, « with low effort »), afin que vous puissiez confirmer quel paramètre est actif sans ouvrir `/model`.

Pour désactiver le raisonnement adaptatif sur Opus 4.6 et Sonnet 4.6 et revenir au budget de réflexion fixe précédent, définissez `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1`. Lorsqu'il est désactivé, ces modèles utilisent le budget fixe contrôlé par `MAX_THINKING_TOKENS`. Voir [variables d'environnement](#).

Contexte étendu

Opus 4.6 et Sonnet 4.6 prennent en charge une [fenêtre de contexte de 1 million de tokens](#) pour les sessions longues avec de grandes bases de code.

Note:

La fenêtre de contexte 1M est actuellement en bêta. Les fonctionnalités, la tarification et la disponibilité peuvent changer.

Le contexte étendu est disponible pour :

- **Utilisateurs API et pay-as-you-go** : accès complet au contexte 1M
- **Abonnés Pro, Max, Teams et Enterprise** : disponible avec [utilisation supplémentaire](#) activée

Pour désactiver complètement le contexte 1M, définissez

`CLAUDE_CODE_DISABLE_1M_CONTEXT=1`. Cela supprime les variantes de modèle 1M du sélecteur de modèle. Voir [variables d'environnement](#).

La sélection d'un modèle 1M ne change pas immédiatement la facturation. Votre session utilise les tarifs standard jusqu'à ce qu'elle dépasse 200K tokens de contexte. Au-delà de 200K tokens, les demandes sont facturées à la [tarification du contexte long](#) avec des [limites de débit](#) dédiées. Pour les abonnés, les tokens au-delà de 200K sont facturés comme utilisation supplémentaire plutôt que par le biais de l'abonnement.

Si votre compte prend en charge le contexte 1M, l'option apparaît dans le sélecteur de modèle (`/model`) dans les dernières versions de Claude Code. Si vous ne la voyez pas, essayez de redémarrer votre session.

Vous pouvez également utiliser le suffixe `[1m]` avec les alias de modèle ou les noms de modèle complets :

```
## Utiliser l'alias sonnet[1m]
/model sonnet[1m]

## Ou ajouter [1m] à un nom de modèle complet
/model claude-sonnet-4-6[1m]
```

Vérifier votre modèle actuel

Vous pouvez voir quel modèle vous utilisez actuellement de plusieurs façons :

- 1. Dans la [ligne d'état](#) (si configurée)
- 2. Dans `/status` , qui affiche également vos informations de compte.

Variables d'environnement

Vous pouvez utiliser les variables d'environnement suivantes, qui doivent être des **noms de modèle** complets (ou équivalents pour votre fournisseur d'API), pour contrôler les noms de modèle auxquels les alias sont mappés.

Variable d'environnement	Description
<code>ANTHROPIC_DEFAULT_OPUS_MODEL</code>	Le modèle à utiliser pour <code>opus</code> , ou pour <code>opusplan</code> lorsque le mode Plan est actif.
<code>ANTHROPIC_DEFAULT_SONNET_MODEL</code>	Le modèle à utiliser pour <code>sonnet</code> , ou pour <code>opusplan</code> lorsque le mode Plan n'est pas actif.
<code>ANTHROPIC_DEFAULT_HAIKU_MODEL</code>	Le modèle à utiliser pour <code>haiku</code> , ou fonctionnalité d'arrière-plan
<code>CLAUDE_CODE_SUBAGENT_MODEL</code>	Le modèle à utiliser pour les subagents

Remarque : `ANTHROPIC_SMALL_FAST_MODEL` est déprécié au profit de `ANTHROPIC_DEFAULT_HAIKU_MODEL` .

Épingler les modèles pour les déploiements tiers

Lors du déploiement de Claude Code via [Bedrock](#), [Vertex AI](#) ou [Foundry](#), épinglez les versions de modèle avant de les déployer auprès des utilisateurs.

Sans épinglage, Claude Code utilise les alias de modèle (`sonnet` , `opus` , `haiku`) qui se résolvent à la dernière version. Lorsqu'Anthropic publie un nouveau modèle, les utilisateurs dont les comptes n'ont pas la nouvelle version activée échoueront silencieusement.

Warning:

Définissez les trois variables d'environnement de modèle sur des ID de version spécifiques dans le cadre de votre configuration initiale. Ignorer cette étape signifie qu'une mise à jour de Claude Code peut casser vos utilisateurs sans aucune action de votre part.

Utilisez les variables d'environnement suivantes avec des ID de modèle spécifiques à la version pour votre fournisseur :

Four-nisseur	Exemple
Bedrock	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-6-v1'</code>
Vertex AI	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>
Foundry	<code>export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-6'</code>

Appliquez le même modèle pour `ANTHROPIC_DEFAULT_SONNET_MODEL` et `ANTHROPIC_DEFAULT_HAIKU_MODEL` . Pour les ID de modèle actuels et hérités sur tous les fournisseurs, voir [Aperçu des modèles](#). Pour mettre à niveau les utilisateurs vers une nouvelle version de modèle, mettez à jour ces variables d'environnement et redéployez.

Note:

La liste d'autorisation `settings.availableModels` s'applique toujours lors de l'utilisation de fournisseurs tiers. Le filtrage correspond à l'alias de modèle (`opus` , `sonnet` , `haiku`), et non à l'ID de modèle spécifique au fournisseur.

Remplacer les ID de modèle par version

Les variables d'environnement au niveau de la famille ci-dessus configurent un ID de modèle par alias de famille. Si vous devez mapper plusieurs versions au sein de la même famille à des ID de fournisseur distincts, utilisez plutôt le paramètre `modelOverrides` .

`modelOverrides` mappe les ID de modèle Anthropic individuels aux chaînes spécifiques au fournisseur que Claude Code envoie à l'API de votre fournisseur. Lorsqu'un utilisateur sélectionne un modèle mappé dans le sélecteur `/model` , Claude Code utilise votre valeur configurée au lieu de la valeur par défaut intégrée.

Cela permet aux administrateurs d'entreprise d'acheminer chaque version de modèle vers un ARN de profil d'inférence Bedrock spécifique, un nom de version Vertex AI ou un nom de déploiement Foundry pour la gouvernance, l'allocation des coûts ou l'acheminement régional.

Définissez `modelOverrides` dans votre [fichier de paramètres](#) :

```
{
  "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"
  }
}
```

Les clés doivent être des ID de modèle Anthropic tels que listés dans l'[Aperçu des modèles](#). Pour les ID de modèle datés, incluez le suffixe de date exactement tel qu'il apparaît là. Les clés inconnues sont ignorées.

Les remplacements remplacent les ID de modèle intégrés qui soutiennent chaque entrée dans le sélecteur `/model`. Sur Bedrock, les remplacements prennent la priorité sur tous les profils d'inférence que Claude Code découvre automatiquement au démarrage. Les valeurs que vous fournissez directement via `ANTHROPIC_MODEL`, `--model` ou les variables d'environnement `ANTHROPIC_DEFAULT_*_MODEL` sont transmises au fournisseur telles quelles et ne sont pas transformées par `modelOverrides`.

`modelOverrides` fonctionne aux côtés de `availableModels`. La liste d'autorisation est évaluée par rapport à l'ID de modèle Anthropic, et non à la valeur de remplacement, donc une entrée comme `"opus"` dans `availableModels` continue de correspondre même lorsque les versions d'Opus sont mappées à des ARN.

Configuration de la mise en cache des invites

Claude Code utilise automatiquement la [mise en cache des invites](#) pour optimiser les performances et réduire les coûts. Vous pouvez désactiver la mise en cache des invites globalement ou pour des niveaux de modèle spécifiques :

Variable d’environnement	Description
<code>DISABLE_PROMPT_CACHING</code>	Définissez sur <code>1</code> pour désactiver la mise en cache des invites pour tous les modèles (prend la priorité sur les paramètres par modèle)
<code>DISABLE_PROMPT_CACHING_HAIKU</code>	Définissez sur <code>1</code> pour désactiver la mise en cache des invites pour les modèles Haiku uniquement
<code>DISABLE_PROMPT_CACHING_SONNET</code>	Définissez sur <code>1</code> pour désactiver la mise en cache des invites pour les modèles Sonnet uniquement
<code>DISABLE_PROMPT_CACHING_OPUS</code>	Définissez sur <code>1</code> pour désactiver la mise en cache des invites pour les modèles Opus uniquement

Ces variables d’environnement vous donnent un contrôle granulaire sur le comportement de la mise en cache des invites. Le paramètre global `DISABLE_PROMPT_CACHING` prend la priorité sur les paramètres spécifiques au modèle, ce qui vous permet de désactiver rapidement toute la mise en cache si nécessaire. Les paramètres par modèle sont utiles pour un contrôle sélectif, par exemple lors du débogage de modèles spécifiques ou du travail avec des fournisseurs cloud qui peuvent avoir des implémentations de mise en cache différentes.

Styles de sortie

Adaptez Claude Code pour des usages au-delà de l'ingénierie logicielle

Les styles de sortie vous permettent d'utiliser Claude Code comme n'importe quel type d'agent tout en conservant ses capacités principales, telles que l'exécution de scripts locaux, la lecture/écriture de fichiers et le suivi des TODOs.

Styles de sortie intégrés

Le style de sortie **Default** de Claude Code est l'invite système existante, conçue pour vous aider à accomplir efficacement les tâches d'ingénierie logicielle.

Il existe deux styles de sortie intégrés supplémentaires axés sur vous enseigner la base de code et le fonctionnement de Claude :

- **Explanatory** : Fournit des « Insights » éducatifs entre les tâches d'ingénierie logicielle pour vous aider à les accomplir. Vous aide à comprendre les choix d'implémentation et les modèles de base de code.
- **Learning** : Mode collaboratif d'apprentissage par la pratique où Claude ne partagera pas seulement des « Insights » lors du codage, mais vous demandera également de contribuer à de petits éléments de code stratégiques. Claude Code ajoutera des marqueurs `TODO(human)` dans votre code pour que vous les implémentiez.

Fonctionnement des styles de sortie

Les styles de sortie modifient directement l'invite système de Claude Code.

- Tous les styles de sortie excluent les instructions pour une sortie efficace (comme répondre de manière concise).
- Les styles de sortie personnalisés excluent les instructions de codage (comme la vérification du code avec des tests), sauf si `keep-coding-instructions` est true.
- Tous les styles de sortie ont leurs propres instructions personnalisées ajoutées à la fin de l'invite système.
- Tous les styles de sortie déclenchent des rappels pour que Claude adhère aux instructions du style de sortie pendant la conversation.

Modifier votre style de sortie

Exécutez `/config` et sélectionnez **Output style** pour choisir un style dans un menu.

Votre sélection est enregistrée dans `.claude/settings.local.json` au [niveau du projet local](#).

Pour définir un style sans le menu, modifiez directement le champ `outputStyle` dans un fichier de paramètres :

```
{
  "outputStyle": "Explanatory"
}
```

Comme le style de sortie est défini dans l'invite système au démarrage de la session, les modifications prennent effet la prochaine fois que vous démarrez une nouvelle session. Cela maintient l'invite système stable tout au long d'une conversation afin que la mise en cache des invites puisse réduire la latence et les coûts.

Créer un style de sortie personnalisé

Les styles de sortie personnalisés sont des fichiers Markdown avec frontmatter et le texte qui sera ajouté à l'invite système :

```
---
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...]
```

Vous pouvez enregistrer ces fichiers au niveau utilisateur (`~/ .claude/output-styles`) ou au niveau projet (`.claude/output-styles`).

Frontmatter

Les fichiers de style de sortie prennent en charge frontmatter pour spécifier les métadonnées :

Frontmatter	Objectif	Par défaut
<code>name</code>	Nom du style de sortie, s'il ne s'agit pas du nom du fichier	Hérité du nom du fichier
<code>description</code>	Description du style de sortie, affichée dans le sélecteur <code>/config</code>	Aucun
<code>keep-coding-instructions</code>	Indique s'il faut conserver les parties de l'invite système de Claude Code liées au codage.	false

Comparaisons avec les fonctionnalités connexes

Styles de sortie vs. CLAUDE.md vs. --append-system-prompt

Les styles de sortie « désactivent » complètement les parties de l'invite système par défaut de Claude Code spécifiques à l'ingénierie logicielle. Ni CLAUDE.md ni `--append-system-prompt` ne modifient l'invite système par défaut de Claude Code. CLAUDE.md ajoute le contenu en tant que message utilisateur *suivant* l'invite système par défaut de Claude Code. `--append-system-prompt` ajoute le contenu à l'invite système.

Styles de sortie vs. Agents

Les styles de sortie affectent directement la boucle d'agent principal et n'affectent que l'invite système. Les agents sont invoqués pour gérer des tâches spécifiques et peuvent inclure des paramètres supplémentaires tels que le modèle à utiliser, les outils disponibles et un contexte sur le moment d'utiliser l'agent.

Styles de sortie vs. Skills

Les styles de sortie modifient la façon dont Claude répond (formatage, ton, structure) et sont toujours actifs une fois sélectionnés. Les skills sont des invites spécifiques à une tâche que vous invoquez avec `/skill-name` ou que Claude charge automatiquement si pertinent. Utilisez les styles de sortie pour les préférences de formatage cohérentes ; utilisez les skills pour les flux de travail et les tâches réutilisables.

Accélérez les réponses avec le mode rapide

Obtenez des réponses Opus 4.6 plus rapides dans Claude Code en activant le mode rapide.

Note:

Le mode rapide est en [aperçu de recherche](#). La fonctionnalité, la tarification et la disponibilité peuvent changer en fonction des commentaires.

Le mode rapide est une configuration haute vitesse pour Claude Opus 4.6, rendant le modèle 2,5 fois plus rapide à un coût par token plus élevé. Activez-le avec `/fast` quand vous avez besoin de vitesse pour un travail interactif comme l'itération rapide ou le débogage en direct, et désactivez-le quand le coût importe plus que la latence.

Le mode rapide n'est pas un modèle différent. Il utilise le même Opus 4.6 avec une configuration API différente qui privilégie la vitesse à l'efficacité des coûts. Vous obtenez une qualité et des capacités identiques, juste des réponses plus rapides.

Note:

Le mode rapide nécessite Claude Code v2.1.36 ou ultérieur. Vérifiez votre version avec `claude --version`.

Ce qu'il faut savoir :

- Utilisez `/fast` pour activer/désactiver le mode rapide dans Claude Code CLI. Également disponible via `/fast` dans l'extension Claude Code VS Code.
- La tarification du mode rapide pour Opus 4.6 commence à 30 \$/150 MTok. Le mode rapide est disponible avec une réduction de 50 % pour tous les plans jusqu'à 23 h 59 PT le 16 février.
- Disponible pour tous les utilisateurs de Claude Code sur les plans d'abonnement (Pro/Max/Team/Enterprise) et Claude Console.
- Pour les utilisateurs de Claude Code sur les plans d'abonnement (Pro/Max/Team/Enterprise), le mode rapide est disponible via l'utilisation supplémentaire uniquement et n'est pas inclus dans les limites de taux d'utilisation de l'abonnement.

Cette page couvre comment [activer le mode rapide](#), son [compromis de coût](#), [quand l'utiliser](#), les [exigences](#), [l'opt-in par session](#), et le [comportement des limites de taux](#).

Activer le mode rapide

Activez le mode rapide de l'une de ces deux façons :

- Tapez `/fast` et appuyez sur Tab pour activer ou désactiver
- Définissez `"fastMode": true` dans votre [fichier de paramètres utilisateur](#)

Par défaut, le mode rapide persiste entre les sessions. Les administrateurs peuvent configurer le mode rapide pour qu'il se réinitialise à chaque session. Consultez [opt-in par session](#) pour plus de détails.

Pour la meilleure efficacité des coûts, activez le mode rapide au début d'une session plutôt que de basculer en milieu de conversation. Consultez [comprendre le compromis de coût](#) pour plus de détails.

Quand vous activez le mode rapide :

- Si vous êtes sur un modèle différent, Claude Code bascule automatiquement vers Opus 4.6
- Vous verrez un message de confirmation : « Mode rapide ACTIVÉ »
- Une petite icône  apparaît à côté de l'invite pendant que le mode rapide est actif
- Exécutez `/fast` à nouveau à tout moment pour vérifier si le mode rapide est activé ou désactivé

Quand vous désactivez le mode rapide avec `/fast` à nouveau, vous restez sur Opus 4.6. Le modèle ne revient pas à votre modèle précédent. Pour basculer vers un modèle différent, utilisez `/model`.

Comprendre le compromis de coût

Le mode rapide a une tarification par token plus élevée que l'Opus 4.6 standard :

Mode	Entrée (MTok)	Sortie (MTok)
Mode rapide sur Opus 4.6 (<200K)	30 \$	150 \$
Mode rapide sur Opus 4.6 (>200K)	60 \$	225 \$

Le mode rapide est compatible avec la fenêtre de contexte étendue de 1M token.

Quand vous basculez en mode rapide en milieu de conversation, vous payez le prix complet du token d'entrée non mis en cache du mode rapide pour tout le contexte de la conversation. Cela coûte plus cher que si vous aviez activé le mode rapide dès le départ.

Décider quand utiliser le mode rapide

Le mode rapide est idéal pour le travail interactif où la latence de réponse importe plus que le coût :

- Itération rapide sur les modifications de code
- Sessions de débogage en direct
- Travail sensible au temps avec des délais serrés

Le mode standard est meilleur pour :

- Les tâches autonomes longues où la vitesse importe moins
- Le traitement par lots ou les pipelines CI/CD
- Les charges de travail sensibles aux coûts

Mode rapide par rapport au niveau d'effort

Le mode rapide et le niveau d'effort affectent tous deux la vitesse de réponse, mais différemment :

Paramètre	Effet
Mode rapide	Même qualité de modèle, latence inférieure, coût plus élevé
Niveau d'effort inférieur	Moins de temps de réflexion, réponses plus rapides, qualité potentiellement inférieure sur les tâches complexes

Vous pouvez combiner les deux : utilisez le mode rapide avec un [niveau d'effort](#) inférieur pour une vitesse maximale sur les tâches simples.

Exigences

Le mode rapide nécessite tous les éléments suivants :

- **Non disponible sur les fournisseurs cloud tiers** : le mode rapide n'est pas disponible sur Amazon Bedrock, Google Vertex AI ou Microsoft Azure Foundry. Le mode rapide est disponible via l'API Anthropic Console et pour les plans d'abonnement Claude utilisant l'utilisation supplémentaire.

- **Utilisation supplémentaire activée** : votre compte doit avoir l'utilisation supplémentaire activée, ce qui permet la facturation au-delà de l'utilisation incluse dans votre plan. Pour les comptes individuels, activez ceci dans vos [paramètres de facturation Console](#). Pour Teams et Enterprise, un administrateur doit activer l'utilisation supplémentaire pour l'organisation.

Note:

L'utilisation du mode rapide est facturée directement à l'utilisation supplémentaire, même si vous avez une utilisation restante sur votre plan. Cela signifie que les tokens du mode rapide ne comptent pas par rapport à l'utilisation incluse de votre plan et sont facturés au tarif du mode rapide à partir du premier token.

- **Activation par l'administrateur pour Teams et Enterprise** : le mode rapide est désactivé par défaut pour les organisations Teams et Enterprise. Un administrateur doit explicitement [activer le mode rapide](#) avant que les utilisateurs puissent y accéder.

Note:

Si votre administrateur n'a pas activé le mode rapide pour votre organisation, la commande `/fast` affichera « Le mode rapide a été désactivé par votre organisation. »

Activer le mode rapide pour votre organisation

Les administrateurs peuvent activer le mode rapide dans :

- **Console** (clients API) : [Préférences Claude Code](#)
- **Claude AI** (Teams et Enterprise) : [Paramètres administrateur > Claude Code](#)

Une autre option pour désactiver complètement le mode rapide est de définir `CLAUDE_CODE_DISABLE_FAST_MODE=1`. Consultez [Variables d'environnement](#).

Opt-in par session

Par défaut, le mode rapide persiste entre les sessions : si un utilisateur active le mode rapide, il reste activé dans les sessions futures. Les administrateurs sur les plans [Teams](#) ou [Enterprise](#) peuvent empêcher cela en définissant `fastModePerSessionOptIn` à `true` dans les [paramètres gérés](#) ou les [paramètres gérés par le serveur](#). Cela fait que chaque session commence avec le mode rapide désactivé, obligeant les utilisateurs à l'activer explicitement avec `/fast`.

```
{  
  "fastModePerSessionOptIn": true  
}
```

Ceci est utile pour contrôler les coûts dans les organisations où les utilisateurs exécutent plusieurs sessions simultanées. Les utilisateurs peuvent toujours activer le mode rapide avec `/fast` quand ils ont besoin de vitesse, mais il se réinitialise au début de chaque nouvelle session. La préférence du mode rapide de l'utilisateur est toujours enregistrée, donc supprimer ce paramètre restaure le comportement persistant par défaut.

Gérer les limites de taux

Le mode rapide a des limites de taux séparées de l'Opus 4.6 standard. Quand vous atteignez la limite de taux du mode rapide ou que vous manquez de crédits d'utilisation supplémentaire :

1. Le mode rapide bascule automatiquement vers l'Opus 4.6 standard
2. L'icône  devient grise pour indiquer le refroidissement
3. Vous continuez à travailler à la vitesse et à la tarification standard
4. Quand le refroidissement expire, le mode rapide se réactive automatiquement

Pour désactiver manuellement le mode rapide au lieu d'attendre le refroidissement, exécutez `/fast` à nouveau.

Aperçu de recherche

Le mode rapide est une fonctionnalité d'aperçu de recherche. Cela signifie :

- La fonctionnalité peut changer en fonction des commentaires
- La disponibilité et la tarification sont sujettes à changement
- La configuration API sous-jacente peut évoluer

Signalez les problèmes ou les commentaires via vos canaux de support Anthropic habituels.

Voir aussi

- [Configuration du modèle](#) : basculer les modèles et ajuster les niveaux d'effort
- [Gérer les coûts efficacement](#) : suivre l'utilisation des tokens et réduire les coûts
- [Configuration de la ligne d'état](#) : afficher les informations du modèle et du contexte

Gérer les coûts efficacement

Suivez l'utilisation des tokens, définissez des limites de dépenses pour l'équipe, et réduisez les coûts de Claude Code grâce à la gestion du contexte, la sélection du modèle, les paramètres de réflexion étendue et les hooks de prétraitement.

Claude Code consomme des tokens pour chaque interaction. Les coûts varient en fonction de la taille de la base de code, de la complexité des requêtes et de la longueur de la conversation. Le coût moyen est de 6 *pardéveloppeurparjour*, *lescoûtsquotidiensrestantendessousde12* pour 90 % des utilisateurs.

Pour l'utilisation en équipe, Claude Code facture selon la consommation de tokens API. En moyenne, Claude Code coûte environ 100 à 200 \$ par développeur par mois avec Sonnet 4.6, bien qu'il y ait une grande variance selon le nombre d'instances que les utilisateurs exécutent et s'ils l'utilisent dans l'automatisation.

Cette page explique comment [suivre vos coûts](#), [gérer les coûts pour les équipes](#) et [réduire l'utilisation des tokens](#).

Suivre vos coûts

Utiliser la commande `/cost`

Note:

La commande `/cost` affiche l'utilisation des tokens API et est destinée aux utilisateurs d'API. Les abonnés Claude Max et Pro ont l'utilisation incluse dans leur abonnement, donc les données `/cost` ne sont pas pertinentes à des fins de facturation. Les abonnés peuvent utiliser `/stats` pour afficher les modèles d'utilisation.

La commande `/cost` fournit des statistiques détaillées sur l'utilisation des tokens pour votre session actuelle :

```
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
```

Gérer les coûts pour les équipes

Lors de l'utilisation de l'API Claude, vous pouvez [définir des limites de dépenses pour l'espace de travail](#) sur la dépense totale de l'espace de travail Claude Code. Les administrateurs peuvent [afficher les rapports de coûts et d'utilisation](#) dans la Console.

Note:

Lorsque vous authentifiez pour la première fois Claude Code avec votre compte Claude Console, un espace de travail appelé « Claude Code » est automatiquement créé pour vous. Cet espace de travail fournit un suivi et une gestion centralisés des coûts pour toute l'utilisation de Claude Code dans votre organisation. Vous ne pouvez pas créer de clés API pour cet espace de travail ; il est exclusivement destiné à l'authentification et à l'utilisation de Claude Code.

Sur Bedrock, Vertex et Foundry, Claude Code n'envoie pas de métriques depuis votre cloud. Pour obtenir des métriques de coûts, plusieurs grandes entreprises ont signalé l'utilisation de [LiteLLM](#), qui est un outil open-source qui aide les entreprises à [suivre les dépenses par clé](#). Ce projet n'est pas affilié à Anthropic et n'a pas été audité pour la sécurité.

Recommandations de limite de débit

Lors de la configuration de Claude Code pour les équipes, tenez compte de ces recommandations de Token Par Minute (TPM) et Requête Par Minute (RPM) par utilisateur en fonction de la taille de votre organisation :

Taille de l'équipe	TPM par utilisateur	RPM par utilisateur
1-5 utilisateurs	200 000-300 000	5-7
5-20 utilisateurs	100 000-150 000	2,5-3,5
20-50 utilisateurs	50 000-75 000	1,25-1,75
50-100 utilisateurs	25 000-35 000	0,62-0,87
100-500 utilisateurs	15 000-20 000	0,37-0,47
500+ utilisateurs	10 000-15 000	0,25-0,35

Par exemple, si vous avez 200 utilisateurs, vous pourriez demander 20 000 TPM pour chaque utilisateur, soit 4 millions de TPM au total ($200 \times 20\,000 = 4\text{ millions}$).

Le TPM par utilisateur diminue à mesure que la taille de l'équipe augmente, car moins d'utilisateurs ont tendance à utiliser Claude Code simultanément dans les grandes organisations. Ces limites de débit s'appliquent au niveau de l'organisation, et non par utilisateur individuel, ce qui signifie que les utilisateurs individuels peuvent temporairement consommer plus que leur part calculée lorsque d'autres n'utilisent pas activement le service.

Note:

Si vous anticipez des scénarios avec une utilisation concurrente inhabituellement élevée (comme des sessions de formation en direct avec de grands groupes), vous pourriez avoir besoin d'allocations TPM plus élevées par utilisateur.

Coûts en tokens des équipes d'agents

Les [équipes d'agents](#) lancent plusieurs instances de Claude Code, chacune avec sa propre fenêtre de contexte. L'utilisation des tokens augmente avec le nombre de coéquipiers actifs et la durée d'exécution de chacun.

Pour maintenir les coûts des équipes d'agents gérables :

- Utilisez Sonnet pour les coéquipiers. Il équilibre la capacité et le coût pour les tâches de coordination.
- Gardez les équipes petites. Chaque coéquipier exécute sa propre fenêtre de contexte, donc l'utilisation des tokens est à peu près proportionnelle à la taille de l'équipe.
- Gardez les invites de génération concentrées. Les coéquipiers chargent CLAUDE.md, les serveurs MCP et les skills automatiquement, mais tout ce qui se trouve dans l'invite de génération s'ajoute à leur contexte dès le départ.
- Nettoyez les équipes lorsque le travail est terminé. Les coéquipiers actifs continuent à consommer des tokens même s'ils sont inactifs.
- Les équipes d'agents sont désactivées par défaut. Définissez `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` dans votre [settings.json](#) ou dans l'environnement pour les activer. Voir [activer les équipes d'agents](#).

Réduire l'utilisation des tokens

Les coûts des tokens augmentent avec la taille du contexte : plus Claude traite de contexte, plus vous utilisez de tokens. Claude Code optimise automatiquement les coûts grâce à la mise en cache des invites (qui réduit les coûts pour le contenu répété comme les invites système) et à la compaction automatique (qui résume l'historique des conversations en approchant les limites du contexte).

Les stratégies suivantes vous aident à maintenir le contexte petit et à réduire les coûts par message.

Gérer le contexte de manière proactive

Utilisez `/cost` pour vérifier votre utilisation actuelle des tokens, ou [configurez votre ligne d'état](#) pour l'afficher en continu.

- **Effacer entre les tâches** : Utilisez `/clear` pour recommencer à zéro lorsque vous passez à un travail non lié. Le contexte obsolète gaspille des tokens à chaque message suivant. Utilisez `/rename` avant d'effacer pour pouvoir facilement retrouver la session plus tard, puis `/resume` pour y revenir.
- **Ajouter des instructions de compaction personnalisées** : `/compact Focus on code samples and API usage` indique à Claude ce qu'il faut préserver lors de la résumé.

Vous pouvez également personnaliser le comportement de compaction dans votre CLAUDE.md :

```
## Compact instructions
```

```
When you are using compact, please focus on test output and code changes
```

Choisir le bon modèle

Sonnet gère bien la plupart des tâches de codage et coûte moins cher qu'Opus. Réservez Opus pour les décisions architecturales complexes ou le raisonnement multi-étapes. Utilisez `/model` pour changer de modèle en cours de session, ou définissez une valeur par défaut dans `/config`. Pour les tâches simples de subagent, spécifiez `model: haiku` dans votre [configuration de subagent](#).

Réduire la surcharge des serveurs MCP

Chaque serveur MCP ajoute des définitions d'outils à votre contexte, même lorsqu'il est inactif. Exécutez `/context` pour voir ce qui consomme de l'espace.

- **Préférez les outils CLI lorsqu'ils sont disponibles** : Les outils comme `gh`, `aws`, `gcloud` et `sentry-cli` sont plus efficaces en contexte que les serveurs MCP car ils n'ajoutent pas de définitions d'outils persistantes. Claude peut exécuter les commandes CLI directement sans la surcharge.
- **Désactiver les serveurs inutilisés** : Exécutez `/mcp` pour voir les serveurs configurés et désactiver ceux que vous n'utilisez pas activement.

- **La recherche d'outils est automatique** : Lorsque les descriptions des outils MCP dépassent 10 % de votre fenêtre de contexte, Claude Code les reporte automatiquement et charge les outils à la demande via [recherche d'outils](#). Puisque les outils reportés n'entrent en contexte que lorsqu'ils sont réellement utilisés, un seuil inférieur signifie moins de définitions d'outils inactifs consommant de l'espace. Définissez un seuil inférieur avec `ENABLE_TOOL_SEARCH=auto:<N>` (par exemple, `auto:5` se déclenche lorsque les outils dépassent 5 % de votre fenêtre de contexte).

Installer des plugins d'intelligence de code pour les langages typés

Les [plugins d'intelligence de code](#) donnent à Claude une navigation de symboles précise au lieu d'une recherche basée sur le texte, réduisant les lectures de fichiers inutiles lors de l'exploration de code inconnu. Un seul appel « aller à la définition » remplace ce qui pourrait autrement être une recherche grep suivie de la lecture de plusieurs fichiers candidats. Les serveurs de langage installés signalent également automatiquement les erreurs de type après les modifications, donc Claude détecte les erreurs sans exécuter un compilateur.

Déléguer le traitement aux hooks et aux skills

Les [hooks](#) personnalisés peuvent prétraiter les données avant que Claude ne les voie. Au lieu que Claude lise un fichier journal de 10 000 lignes pour trouver les erreurs, un hook peut rechercher `ERROR` et retourner uniquement les lignes correspondantes, réduisant le contexte de dizaines de milliers de tokens à des centaines.

Une [skill](#) peut donner à Claude des connaissances de domaine pour qu'il n'ait pas à explorer. Par exemple, une skill « codebase-overview » pourrait décrire l'architecture de votre projet, les répertoires clés et les conventions de nommage. Lorsque Claude invoque la skill, il obtient ce contexte immédiatement au lieu de dépenser des tokens pour lire plusieurs fichiers pour comprendre la structure.

Par exemple, ce hook PreToolUse filtre la sortie des tests pour afficher uniquement les échecs :

settings.json

Ajoutez ceci à votre [settings.json](#) pour exécuter le hook avant chaque commande Bash :


```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

Le hook appelle ce script, qui vérifie si la commande est un exécuteur de test et la modifie pour afficher uniquement les échecs :

```
#!/bin/bash
input=$(cat)
cmd=$(echo "$input" | jq -r '.tool_input.command')

## If running tests, filter to show only failures
if [[ "$cmd" =~ ^(npm test|pytest|go test) ]]; then
  filtered_cmd="$cmd 2>&1 | grep -A 5 -E '(FAIL|ERROR|error:)' | head -100"
  echo "{\"hookSpecificOutput\":{\"hookEventName\":\"PreToolUse\",\"permissionDecision\":\"allow\",\"updatedInput\":{\"command\":\"$filtered_cmd\"}}}"
else
  echo "{}"
fi
```

Déplacer les instructions de CLAUDE.md vers les skills

Votre fichier [CLAUDE.md](#) est chargé en contexte au démarrage de la session. S'il contient des instructions détaillées pour des flux de travail spécifiques (comme les révisions de PR ou les migrations de base de données), ces tokens sont présents même lorsque vous faites

un travail non lié. Les [skills](#) se chargent à la demande uniquement lorsqu'elles sont invoquées, donc déplacer les instructions spécialisées dans les skills maintient votre contexte de base plus petit. Visez à garder CLAUDE.md en dessous d'environ 500 lignes en incluant uniquement les éléments essentiels.

Ajuster la réflexion étendue

La réflexion étendue est activée par défaut avec un budget de 31 999 tokens car elle améliore considérablement les performances sur les tâches complexes de planification et de raisonnement. Cependant, les tokens de réflexion sont facturés comme des tokens de sortie, donc pour les tâches plus simples où un raisonnement approfondi n'est pas nécessaire, vous pouvez réduire les coûts en abaissant le [niveau d'effort](#) dans `/model` pour Opus 4.6, en désactivant la réflexion dans `/config`, ou en abaissant le budget (par exemple, `MAX_THINKING_TOKENS=8000`).

Déléguer les opérations détaillées aux subagents

L'exécution de tests, la récupération de documentation ou le traitement de fichiers journaux peuvent consommer un contexte important. Déléguez-les aux [subagents](#) pour que la sortie détaillée reste dans le contexte du subagent tandis que seul un résumé revient à votre conversation principale.

Gérer les coûts des équipes d'agents

Les équipes d'agents utilisent environ 7 fois plus de tokens que les sessions standard lorsque les coéquipiers s'exécutent en mode plan, car chaque coéquipier maintient sa propre fenêtre de contexte et s'exécute en tant qu'instance Claude distincte. Gardez les tâches d'équipe petites et autonomes pour limiter l'utilisation des tokens par coéquipier. Voir [équipes d'agents](#) pour plus de détails.

Écrire des invites spécifiques

Les demandes vagues comme « améliorer cette base de code » déclenchent une analyse large. Les demandes spécifiques comme « ajouter la validation des entrées à la fonction de connexion dans auth.ts » permettent à Claude de travailler efficacement avec des lectures de fichiers minimales.

Travailler efficacement sur des tâches complexes

Pour un travail plus long ou plus complexe, ces habitudes aident à éviter les tokens gaspillés en prenant la mauvaise direction :

- **Utilisez le mode plan pour les tâches complexes** : Appuyez sur Maj+Tab pour entrer en [mode plan](#) avant l'implémentation. Claude explore la base de code et propose

une approche pour votre approbation, évitant les retouches coûteuses lorsque la direction initiale est mauvaise.

- **Corriger la trajectoire tôt** : Si Claude commence à aller dans la mauvaise direction, appuyez sur Échap pour arrêter immédiatement. Utilisez `/rewind` ou appuyez deux fois sur Échap pour restaurer la conversation et le code à un point de contrôle précédent.
- **Donner des cibles de vérification** : Incluez des cas de test, collez des captures d'écran ou définissez la sortie attendue dans votre invite. Lorsque Claude peut vérifier son propre travail, il détecte les problèmes avant que vous ayez besoin de demander des corrections.
- **Tester de manière progressive** : Écrivez un fichier, testez-le, puis continuez. Cela détecte les problèmes tôt lorsqu'ils sont bon marché à corriger.

Utilisation des tokens en arrière-plan

Claude Code utilise des tokens pour certaines fonctionnalités en arrière-plan même lorsqu'il est inactif :

- **Résumé des conversations** : Les tâches en arrière-plan qui résument les conversations précédentes pour la fonctionnalité `claude --resume`
- **Traitement des commandes** : Certaines commandes comme `/cost` peuvent générer des requêtes pour vérifier l'état

Ces processus en arrière-plan consomment une petite quantité de tokens (généralement moins de 0,04 \$ par session) même sans interaction active.

Comprendre les changements dans le comportement de Claude Code

Claude Code reçoit régulièrement des mises à jour qui peuvent modifier le fonctionnement des fonctionnalités, y compris la génération de rapports de coûts. Exécutez `claude --version` pour vérifier votre version actuelle. Pour des questions de facturation spécifiques, contactez le support Anthropic via votre [compte Console](#). Pour les déploiements en équipe, commencez par un petit groupe pilote pour établir les modèles d'utilisation avant un déploiement plus large.

Part 5: Extensibility

Référence des hooks

Référence pour les événements de hook Claude Code, le schéma de configuration, les formats d'entrée/sortie JSON, les codes de sortie, les hooks asynchrones, les hooks HTTP, les hooks de prompt et les hooks d'outils MCP.

Tip:

Pour un guide de démarrage rapide avec des exemples, consultez [Automatiser les flux de travail avec les hooks](#).

Les hooks sont des commandes shell définies par l'utilisateur, des points de terminaison HTTP ou des prompts LLM qui s'exécutent automatiquement à des points spécifiques du cycle de vie de Claude Code. Utilisez cette référence pour consulter les schémas d'événements, les options de configuration, les formats d'entrée/sortie JSON et les fonctionnalités avancées comme les hooks asynchrones, les hooks HTTP et les hooks d'outils MCP. Si vous configurez des hooks pour la première fois, commencez plutôt par le [guide](#).

Cycle de vie des hooks

Les hooks se déclenchent à des points spécifiques pendant une session Claude Code. Lorsqu'un événement se déclenche et qu'un matcher correspond, Claude Code transmet le contexte JSON de l'événement à votre gestionnaire de hook. Pour les hooks de commande, l'entrée arrive sur stdin. Pour les hooks HTTP, elle arrive dans le corps de la requête POST. Votre gestionnaire peut alors inspecter l'entrée, prendre une action et éventuellement retourner une décision. Certains événements se déclenchent une fois par session, tandis que d'autres se déclenchent à plusieurs reprises dans la boucle agentique :

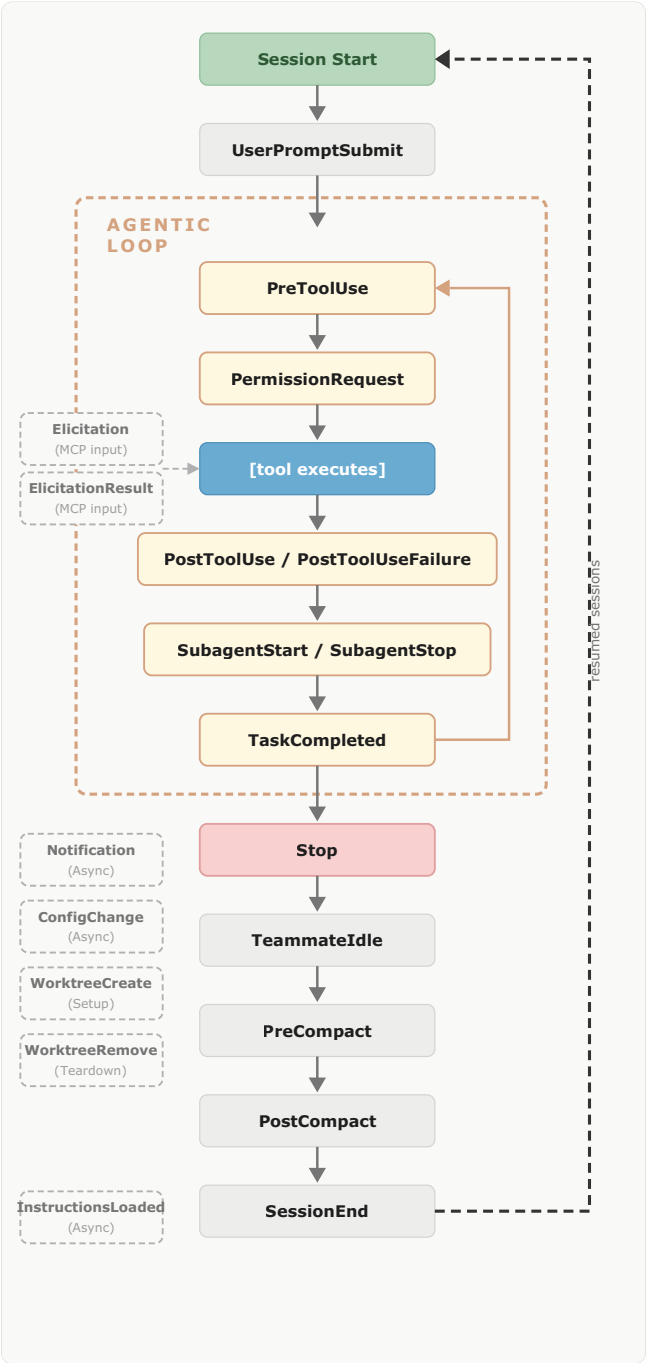


Diagramme du cycle de vie des hooks montrant la séquence des hooks de SessionStart à travers la boucle agentique jusqu'à SessionEnd, avec WorktreeCreate, WorktreeRemove et InstructionsLoaded comme événements asynchrones autonomes

Le tableau ci-dessous résume le moment où chaque événement se déclenche. La section [Événements de hook](#) documente le schéma d'entrée complet et les options de contrôle de décision pour chacun.

Event	When it fires
SessionStart	When a session begins or resumes
UserPromptSubmit	When you submit a prompt, before Claude processes it
PreToolUse	Before a tool call executes. Can block it
PermissionRequest	When a permission dialog appears
PostToolUse	After a tool call succeeds
PostToolUseFailure	After a tool call fails
Notification	When Claude Code sends a notification
SubagentStart	When a subagent is spawned
SubagentStop	When a subagent finishes
Stop	When Claude finishes responding
TeammateIdle	When an agent team teammate is about to go idle
TaskCompleted	When a task is being marked as completed
InstructionsLoaded	When a CLAUDE.md or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
ConfigChange	When a configuration file changes during a session
WorktreeCreate	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
WorktreeRemove	When a worktree is being removed, either at session exit or when a subagent finishes
PreCompact	Before context compaction

Event	When it fires
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

Comment un hook se résout

Pour voir comment ces éléments s'assemblent, considérez ce hook `PreToolUse` qui bloque les commandes shell destructrices. Le hook exécute `block-rm.sh` avant chaque appel d'outil Bash :

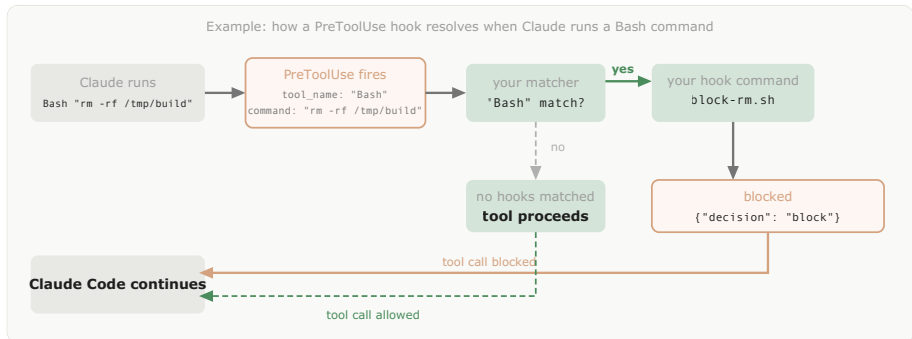
```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

Le script lit l'entrée JSON depuis stdin, extrait la commande et retourne une `permissionDecision` de `"deny"` si elle contient `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
```

Supposons maintenant que Claude Code décide d'exécuter `Bash "rm -rf /tmp/build"`. Voici ce qui se passe :



Flux de résolution du hook : l'événement PreToolUse se déclenche, le matcher vérifie la correspondance Bash, le gestionnaire de hook s'exécute, le résultat revient à Claude Code

Step 1: L'événement se déclenche

L'événement `PreToolUse` se déclenche. Claude Code envoie l'entrée de l'outil en JSON sur stdin au hook :

```
{ "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
```


Step 2: Le matcher vérifie

Le matcher `"Bash"` correspond au nom de l'outil, donc `block-rm.sh` s'exécute. Si vous omettez le matcher ou utilisez `"*"`, le hook s'exécute à chaque occurrence de l'événement. Les hooks ne s'ignorent que lorsqu'un matcher est défini et ne correspond pas.

Step 3: Le gestionnaire de hook s'exécute

Le script extrait `"rm -rf /tmp/build"` de l'entrée et trouve `rm -rf`, donc il imprime une décision sur stdout :

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Destructive command blocked by hook"
  }
}
```

Si la commande avait été sûre (comme `npm test`), le script aurait atteint `exit 0` à la place, ce qui indique à Claude Code d'autoriser l'appel d'outil sans action supplémentaire.

Step 4: Claude Code agit sur le résultat

Claude Code lit la décision JSON, bloque l'appel d'outil et montre la raison à Claude.

La section [Configuration](#) ci-dessous documente le schéma complet, et chaque section [événement de hook](#) documente l'entrée que votre commande reçoit et la sortie qu'elle peut retourner.

Configuration

Les hooks sont définis dans les fichiers de paramètres JSON. La configuration a trois niveaux d'imbrication :

1. Choisissez un [événement de hook](#) auquel répondre, comme `PreToolUse` ou `Stop`
2. Ajoutez un [groupe de matcher](#) pour filtrer quand il se déclenche, comme `'uniquement pour l'outil Bash'`
3. Définissez un ou plusieurs [gestionnaires de hook](#) à exécuter lorsqu'ils correspondent

Consultez [Comment un hook se résout](#) ci-dessus pour une procédure pas à pas complète avec un exemple annoté.

Note:

Cette page utilise des termes spécifiques pour chaque niveau : **événement de hook** pour le point du cycle de vie, **groupe de matcher** pour le filtre et **gestionnaire de hook** pour la commande shell, le point de terminaison HTTP, le prompt ou l'agent qui s'exécute. ' Hook ' seul fait référence à la fonctionnalité générale.

Emplacements des hooks

L'endroit où vous définissez un hook détermine sa portée :

Emplacement	Portée	Partageable
<code>~/.claude/settings.json</code>	Tous vos projets	Non, local à votre machine
<code>.claude/settings.json</code>	Projet unique	Oui, peut être commité dans le repo
<code>.claude/settings.local.json</code>	Projet unique	Non, ignoré par git
Paramètres de politique gérée	À l'échelle de l'organisation	Oui, contrôlé par l'administrateur
<code>Plugin hooks/hooks.json</code>	Lorsque le plugin est activé	Oui, fourni avec le plugin
Frontmatter Skill ou agent	Pendant que le composant est actif	Oui, défini dans le fichier du composant

Pour plus de détails sur la résolution des fichiers de paramètres, consultez [paramètres](#). Les administrateurs d'entreprise peuvent utiliser `allowManagedHooksOnly` pour bloquer les hooks utilisateur, projet et plugin. Consultez [Configuration des hooks](#).

Modèles de matcher

Le champ `matcher` est une chaîne regex qui filtre quand les hooks se déclenchent. Utilisez `"*"`, `""` ou omettez entièrement `matcher` pour correspondre à toutes les occurrences. Chaque type d'événement correspond sur un champ différent :

Événement	Ce que le matcher filtre	Exemples de valeurs de matcher
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	nom de l'outil	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	comment la session a commencé	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	pourquoi la session s'est terminée	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_di</code> <code>sabled</code> , <code>other</code>
<code>Notification</code>	type de notification	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>
<code>SubagentStart</code>	type d'agent	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> ou noms d'agents person- nalisés
<code>PreCompact</code>	ce qui a déclenché la compaction	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	type d'agent	mêmes valeurs que <code>Subag</code> <code>entStart</code>
<code>ConfigChange</code>	source de configuration	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>
<code>UserPromptSubmit</code> , <code>Stop</code> , <code>TeammateIdle</code> , <code>TaskCompleted</code> , <code>WorktreeCreate</code> , <code>WorktreeRemove</code> , <code>InstructionsLoaded</code>	pas de support de matcher	se déclenche toujours à chaque occurrence

Le matcher est une regex, donc `Edit|Write` correspond à l'un ou l'autre outil et `Notebook.*` correspond à tout outil commençant par Notebook. Le matcher s'exécute sur un champ de l'[entrée JSON](#) que Claude Code envoie à votre hook sur stdin. Pour les événements d'outil, ce champ est `tool_name`. Chaque section [événement de hook](#) liste l'ensemble complet des valeurs de matcher et le schéma d'entrée pour cet événement.

Cet exemple exécute un script de linting uniquement lorsque Claude écrit ou édite un fichier :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

`UserPromptSubmit`, `Stop`, `TeammateIdle`, `TaskCompleted`, `WorktreeCreate`, `WorktreeRemove` et `InstructionsLoaded` ne supportent pas les matchers et se déclenchent toujours à chaque occurrence. Si vous ajoutez un champ `matcher` à ces événements, il est silencieusement ignoré.

Correspondre aux outils MCP

Les outils du serveur [MCP](#) apparaissent comme des outils réguliers dans les événements d'outil (`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `PermissionRequest`), vous pouvez donc les faire correspondre de la même manière que tout autre nom d'outil.

Les outils MCP suivent le modèle de nommage `mcp_<server>__<tool>`, par exemple :

- `mcp_memory_create_entities` : outil create entities du serveur Memory
- `mcp_filesystem_read_file` : outil read file du serveur Filesystem
- `mcp_github_search_repositories` : outil search du serveur GitHub

Utilisez des modèles regex pour cibler des outils MCP spécifiques ou des groupes d'outils :

- `mcp__memory__.*` correspond à tous les outils du serveur `memory`
- `mcp__.*__write.*` correspond à tout outil contenant « write » de n'importe quel serveur

Cet exemple enregistre toutes les opérations du serveur `memory` et valide les opérations d'écriture de n'importe quel serveur 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"
          }
        ]
      }
    ]
  }
}
```

Champs du gestionnaire de hook

Chaque objet du tableau `hooks` interne est un gestionnaire de hook : la commande shell, le point de terminaison HTTP, le prompt LLM ou l'agent qui s'exécute lorsque le matcher correspond. Il y a quatre types :

- **Hooks de commande** (`type: "command"`) : exécutent une commande shell. Votre script reçoit l'[entrée JSON](#) de l'événement sur stdin et communique les résultats via les codes de sortie et stdout.
- **Hooks HTTP** (`type: "http"`) : envoient l'entrée JSON de l'événement en tant que requête HTTP POST à une URL. Le point de terminaison communique les résultats via le corps de la réponse en utilisant le même [format de sortie JSON](#) que les hooks de commande.
- **Hooks de prompt** (`type: "prompt"`) : envoient un prompt à un modèle Claude pour une évaluation en un seul tour. Le modèle retourne une décision oui/non en JSON. Consultez [Hooks basés sur des prompts](#).
- **Hooks d'agent** (`type: "agent"`) : lancent un subagent qui peut utiliser des outils comme Read, Grep et Glob pour vérifier les conditions avant de retourner une décision. Consultez [Hooks basés sur des agents](#).

Champs communs

Ces champs s'appliquent à tous les types de hooks :

Champ	Requis	Description
<code>type</code>	oui	<code>"command"</code> , <code>"http"</code> , <code>"prompt"</code> ou <code>"agent"</code>
<code>timeout</code>	non	Secondes avant annulation. Valeurs par défaut : 600 pour command, 30 pour prompt, 60 pour agent
<code>statusMessage</code>	non	Message de spinner personnalisé affiché pendant l'exécution du hook

Champ	Requis	Description
<code>once</code>	non	Si <code>true</code> , s'exécute une seule fois par session puis est supprimé. Skills uniquement, pas agents. Consultez Hooks dans les skills et agents

Champs du hook de commande

En plus des [champs communs](#), les hooks de commande acceptent ces champs :

Champ	Requis	Description
<code>command</code>	oui	Commande shell à exécuter
<code>async</code>	non	Si <code>true</code> , s'exécute en arrière-plan sans bloquer. Consultez Exécuter les hooks en arrière-plan

Champs du hook HTTP

En plus des [champs communs](#), les hooks HTTP acceptent ces champs :

Champ	Requis	Description
<code>url</code>	oui	URL vers laquelle envoyer la requête POST
<code>headers</code>	non	En-têtes HTTP supplémentaires sous forme de paires clé-valeur. Les valeurs supportent l'interpolation de variables d'environnement en utilisant la syntaxe <code>\$VAR_NAME</code> ou <code>\${VAR_NAME}</code> . Seules les variables listées dans <code>allowedEnvVars</code> sont résolues

Champ	Requis	Description
<code>allowedEnvVars</code>	non	Liste des noms de variables d'environnement qui peuvent être interpolés dans les valeurs d'en-tête. Les références aux variables non listées sont remplacées par des chaînes vides. Requis pour que l'interpolation de variables d'environnement fonctionne

Claude Code envoie l'[entrée JSON](#) du hook en tant que corps de la requête POST avec `Content-Type: application/json`. Le corps de la réponse utilise le même [format de sortie JSON](#) que les hooks de commande.

La gestion des erreurs diffère des hooks de commande : les réponses non-2xx, les défaillances de connexion et les délais d'expiration produisent tous des erreurs non-bloquantes qui permettent à l'exécution de continuer. Pour bloquer un appel d'outil ou refuser une permission, retournez une réponse 2xx avec un corps JSON contenant `decision: "block"` ou un `hookSpecificOutput` avec `permissionDecision: "deny"`.

Cet exemple envoie les événements `PreToolUse` à un service de validation local, en s'authentifiant avec un token de la variable d'environnement `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:

Les hooks HTTP doivent être configurés en éditant directement les paramètres JSON. Le menu interactif `/hooks` ne supporte que l'ajout de hooks de commande.

Champs des hooks de prompt et d'agent

En plus des [champs communs](#), les hooks de prompt et d'agent acceptent ces champs :

Champ	Requis	Description
<code>prompt</code>	oui	Texte du prompt à envoyer au modèle. Utilisez <code>\$ARGUMENTS</code> comme placeholder pour l'entrée JSON du hook

Champ	Requis	Description
<code>model</code>	non	Modèle à utiliser pour l'évaluation. Par défaut un modèle rapide

Tous les hooks correspondants s'exécutent en parallèle, et les gestionnaires identiques sont automatiquement déduplicués. Les hooks de commande sont déduplicués par chaîne de commande, et les hooks HTTP sont déduplicués par URL. Les gestionnaires s'exécutent dans le répertoire courant avec l'environnement de Claude Code. La variable d'environnement `$CLAUDE_CODE_REMOTE` est définie à `"true"` dans les environnements web distants et n'est pas définie dans le CLI local.

Référencer les scripts par chemin

Utilisez les variables d'environnement pour référencer les scripts de hook par rapport à la racine du projet ou du plugin, indépendamment du répertoire de travail lorsque le hook s'exécute :

- `$CLAUDE_PROJECT_DIR` : la racine du projet. Enveloppez entre guillemets pour gérer les chemins avec des espaces.
- `${CLAUDE_PLUGIN_ROOT}` : le répertoire racine du plugin, pour les scripts fournis avec un [plugin](#).

Scripts de projet

Cet exemple utilise `$CLAUDE_PROJECT_DIR` pour exécuter un vérificateur de style à partir du répertoire `.claude/hooks/` du projet après tout appel d'outil `Write` ou `Edit` :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/check-style.sh"
          }
        ]
      }
    ]
  }
}

```

Scripts de plugin

Définissez les hooks de plugin dans `hooks/hooks.json` avec un champ `description` optionnel au niveau supérieur. Lorsqu'un plugin est activé, ses hooks fusionnent avec vos hooks utilisateur et projet.

Cet exemple exécute un script de formatage fourni avec le plugin :

```

{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "timeout": 30
          }
        ]
      }
    ]
  }
}

```

Consultez la [référence des composants de plugin](#) pour plus de détails sur la création de hooks de plugin.

Hooks dans les skills et agents

En plus des fichiers de paramètres et des plugins, les hooks peuvent être définis directement dans les [skills](#) et les [subagents](#) en utilisant le frontmatter. Ces hooks sont limités au cycle de vie du composant et ne s'exécutent que lorsque ce composant est actif.

Tous les événements de hook sont supportés. Pour les subagents, les hooks **Stop** sont automatiquement convertis en **SubagentStop** puisque c'est l'événement qui se déclenche lorsqu'un subagent se termine.

Les hooks utilisent le même format de configuration que les hooks basés sur les paramètres mais sont limités à la durée de vie du composant et nettoyés lorsqu'il se termine.

Ce skill définit un hook **PreToolUse** qui exécute un script de validation de sécurité avant chaque commande **Bash** :

```

---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---

```

Les agents utilisent le même format dans leur frontmatter YAML.

Le menu `/hooks`

Tapez `/hooks` dans Claude Code pour ouvrir le gestionnaire de hooks interactif, où vous pouvez afficher, ajouter et supprimer des hooks sans éditer directement les fichiers de paramètres. Pour une procédure pas à pas, consultez [Configurer votre premier hook](#) dans le guide.

Chaque hook du menu est étiqueté avec un préfixe entre crochets indiquant sa source :

- `[User]` : de `~/.claude/settings.json`
- `[Project]` : de `.claude/settings.json`
- `[Local]` : de `.claude/settings.local.json`
- `[Plugin]` : du `hooks/hooks.json` d'un plugin, lecture seule

Désactiver ou supprimer les hooks

Pour supprimer un hook, supprimez son entrée du fichier de paramètres JSON ou utilisez le menu `/hooks` et sélectionnez le hook à supprimer.

Pour désactiver temporairement tous les hooks sans les supprimer, définissez `"disableAllHooks": true` dans votre fichier de paramètres ou utilisez le bouton bascule dans le menu `/hooks`. Il n'y a aucun moyen de désactiver un hook individuel tout en le gardant dans la configuration.

Le paramètre `disableAllHooks` respecte la hiérarchie des paramètres gérés. Si un administrateur a configuré des hooks via les paramètres de politique gérée, `disableAllHooks` défini dans les paramètres utilisateur, projet ou local ne peut pas désactiver ces hooks gérés. Seul `disableAllHooks` défini au niveau des paramètres gérés peut désactiver les hooks gérés.

Les modifications directes des hooks dans les fichiers de paramètres ne prennent pas effet immédiatement. Claude Code capture un snapshot des hooks au démarrage et l'utilise tout au long de la session. Cela empêche les modifications de hooks malveillantes ou accidentelles de prendre effet en milieu de session sans votre examen. Si les hooks sont modifiés en externe, Claude Code vous avertit et exige un examen dans le menu `/hooks` avant que les modifications ne s'appliquent.

Entrée et sortie des hooks

Les hooks de commande reçoivent les données JSON via stdin et communiquent les résultats via les codes de sortie, stdout et stderr. Les hooks HTTP reçoivent le même JSON que le corps de la requête POST et communiquent les résultats via le corps de la réponse HTTP. Cette section couvre les champs et le comportement communs à tous les événements. Chaque section d'événement sous [Événements de hook](#) inclut son schéma d'entrée spécifique et les options de contrôle de décision.

Champs d'entrée communs

Tous les événements de hook reçoivent ces champs en JSON, en plus des champs spécifiques à l'événement documentés dans chaque section [événement de hook](#). Pour les hooks de commande, ce JSON arrive via stdin. Pour les hooks HTTP, il arrive dans le corps de la requête POST.

Champ	Description
<code>session_id</code>	Identifiant de session actuel
<code>transcript_path</code>	Chemin vers le JSON de conversation
<code>cwd</code>	Répertoire de travail actuel lorsque le hook est invoqué
<code>permission_mode</code>	Mode de permission actuel : <code>"default"</code> , <code>"plan"</code> , <code>"acceptEdits"</code> , <code>"dontAsk"</code> ou <code>"bypassPermissions"</code>
<code>hook_event_name</code>	Nom de l'événement qui s'est déclenché

Lors de l'exécution avec `--agent` ou à l'intérieur d'un subagent, deux champs supplémentaires sont inclus :

Champ	Description
<code>agent_id</code>	Identifiant unique pour le subagent. Présent uniquement lorsque le hook se déclenche à l'intérieur d'un appel de subagent. Utilisez ceci pour distinguer les appels de hook de subagent des appels du thread principal.
<code>agent_type</code>	Nom de l'agent (par exemple, <code>"Explore"</code> ou <code>"security-reviewer"</code>). Présent lorsque la session utilise <code>--agent</code> ou que le hook se déclenche à l'intérieur d'un subagent. Pour les subagents, le type du subagent prend précedence sur la valeur <code>--agent</code> de la session.

Par exemple, un hook `PreToolUse` pour une commande Bash reçoit ceci sur 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"
  }
}
```

Les champs `tool_name` et `tool_input` sont spécifiques à l'événement. Chaque section [événement de hook](#) documente les champs supplémentaires pour cet événement.

Sortie du code de sortie

Le code de sortie de votre commande de hook indique à Claude Code si l'action doit procéder, être bloquée ou être ignorée.

Exit 0 signifie succès. Claude Code analyse stdout pour les [champs de sortie JSON](#). La sortie JSON n'est traitée que sur exit 0. Pour la plupart des événements, stdout n'est affiché que en mode verbeux (`Ctrl+0`). Les exceptions sont `UserPromptSubmit` et `SessionStart`, où stdout est ajouté en tant que contexte que Claude peut voir et sur lequel agir.

Exit 2 signifie une erreur bloquante. Claude Code ignore stdout et tout JSON qu'il contient. À la place, le texte stderr est renvoyé à Claude en tant que message d'erreur. L'effet dépend de l'événement : `PreToolUse` bloque l'appel d'outil, `UserPromptSubmit` rejette le prompt, etc. Consultez [comportement du code de sortie 2](#) pour la liste complète.

Tout autre code de sortie est une erreur non-bloquante. stderr est affiché en mode verbeux (`Ctrl+Q`) et l'exécution continue.

Par exemple, un script de commande de hook qui bloque les commandes Bash dangereuses :

```
#!/bin/bash
## Lit l'entrée JSON depuis stdin, vérifie la commande
command=$(jq -r '.tool_input.command' < /dev/stdin)

if [[ "$command" = rm* ]]; then
    echo "Blocked: rm commands are not allowed" >&2
    exit 2 # Erreur bloquante : l'appel d'outil est empêché
fi

exit 0 # Succès : l'appel d'outil procède
```

Comportement du code de sortie 2 par événement

Le code de sortie 2 est la façon pour un hook de signaler « arrêtez, ne faites pas cela ». L'effet dépend de l'événement, car certains événements représentent des actions qui peuvent être bloquées (comme un appel d'outil qui ne s'est pas encore produit) et d'autres représentent des choses qui se sont déjà produites ou ne peuvent pas être empêchées.

Événement de hook	Peut bloquer ?	Que se passe-t-il sur exit 2
<code>PreToolUse</code>	Oui	Bloque l'appel d'outil
<code>PermissionRequest</code>	Oui	Refuse la permission
<code>UserPromptSubmit</code>	Oui	Bloque le traitement du prompt et efface le prompt
<code>Stop</code>	Oui	Empêche Claude de s'arrêter, continue la conversation

Événement de hook	Peut bloquer ?	Que se passe-t-il sur exit 2
<code>SubagentStop</code>	Oui	Empêche le subagent de s'arrêter
<code>TeammateIdle</code>	Oui	Empêche le coéquipier de devenir inactif (le coéquipier continue de travailler)
<code>TaskCompleted</code>	Oui	Empêche la tâche d'être marquée comme complétée
<code>ConfigChange</code>	Oui	Bloque la modification de configuration de prendre effet (sauf <code>policy_settings</code>)
<code>PostToolUse</code>	Non	Affiche stderr à Claude (l'outil a déjà s'exécuté)
<code>PostToolUseFailure</code>	Non	Affiche stderr à Claude (l'outil a déjà échoué)
<code>Notification</code>	Non	Affiche stderr à l'utilisateur uniquement
<code>SubagentStart</code>	Non	Affiche stderr à l'utilisateur uniquement
<code>SessionStart</code>	Non	Affiche stderr à l'utilisateur uniquement
<code>SessionEnd</code>	Non	Affiche stderr à l'utilisateur uniquement
<code>PreCompact</code>	Non	Affiche stderr à l'utilisateur uniquement
<code>WorktreeCreate</code>	Oui	Tout code de sortie non-zéro provoque l'échec de la création du worktree

Événement de hook	Peut bloquer ?	Que se passe-t-il sur exit 2
<code>WorktreeRemove</code>	Non	Les défaillances sont enregistrées en mode debug uniquement
<code>InstructionsLoaded</code>	Non	Le code de sortie est ignoré

Gestion des réponses HTTP

Les hooks HTTP utilisent les codes de statut HTTP et les corps de réponse au lieu des codes de sortie et de stdout :

- **2xx avec un corps vide** : succès, équivalent à exit code 0 sans sortie
- **2xx avec un corps en texte brut** : succès, le texte est ajouté en tant que contexte
- **2xx avec un corps JSON** : succès, analysé en utilisant le même schéma [sortie JSON](#) que les hooks de commande
- **Statut non-2xx** : erreur non-bloquante, l'exécution continue
- **Défaillance de connexion ou délai d'expiration** : erreur non-bloquante, l'exécution continue

Contrairement aux hooks de commande, les hooks HTTP ne peuvent pas signaler une erreur bloquante via les codes de statut seuls. Pour bloquer un appel d'outil ou refuser une permission, retournez une réponse 2xx avec un corps JSON contenant les champs de décision appropriés.

Sortie JSON

Les codes de sortie vous permettent d'autoriser ou de bloquer, mais la sortie JSON vous donne un contrôle plus granulaire. Au lieu de quitter avec le code 2 pour bloquer, quittez 0 et imprimez un objet JSON sur stdout. Claude Code lit les champs spécifiques de ce JSON pour contrôler le comportement, y compris [contrôle de décision](#) pour bloquer, autoriser ou escalader à l'utilisateur.

Note:

Vous devez choisir une approche par hook, pas les deux : soit utiliser les codes de sortie seuls pour signaler, soit quitter 0 et imprimer JSON pour un contrôle structuré. Claude Code ne traite JSON que sur exit 0. Si vous quittez 2, tout JSON est ignoré.

La sortie stdout de votre hook doit contenir uniquement l'objet JSON. Si votre profil shell imprime du texte au démarrage, cela peut interférer avec l'analyse JSON. Consultez [Validation JSON échouée](#) dans le guide de dépannage.

L'objet JSON supporte trois types de champs :

- **Champs universels** comme `continue` fonctionnent sur tous les événements. Ceux-ci sont listés dans le tableau ci-dessous.
- **decision et reason au niveau supérieur** sont utilisés par certains événements pour bloquer ou fournir des commentaires.
- **hookSpecificOutput** est un objet imbriqué pour les événements qui ont besoin d'un contrôle plus riche. Il nécessite un champ `hookEventName` défini au nom de l'événement.

Champ	Défaut	Description
<code>continue</code>	<code>true</code>	Si <code>false</code> , Claude arrête complètement le traitement après l'exécution du hook. Prend précedence sur tous les champs de decision spécifiques à l'événement
<code>stopReason</code>	aucun	Message affiché à l'utilisateur lorsque <code>continue</code> est <code>false</code> . Non affiché à Claude
<code>suppressOutput</code>	<code>false</code>	Si <code>true</code> , masque stdout de la sortie du mode verbeux
<code>systemMessage</code>	aucun	Message d'avertissement affiché à l'utilisateur

Pour arrêter Claude entièrement indépendamment du type d'événement :

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

Contrôle de décision

Tous les événements ne supportent pas le blocage ou le contrôle du comportement via JSON. Les événements qui le font utilisent chacun un ensemble différent de champs pour exprimer cette décision. Utilisez ce tableau comme référence rapide avant d'écrire un hook :

Événements	Modèle de décision	Champs clés
UserPromptSubmit, Post-ToolUse, PostToolUseFailure, Stop, SubagentStop, ConfigChange	<code>decision</code> au niveau supérieur	<code>decision: "block"</code> , <code>reason</code>
TeammateIdle, TaskCompleted	Code de sortie ou <code>continue: false</code>	Le code de sortie 2 bloque l'action avec commentaires stderr. JSON <code>{"continue": false, "stopReason": "..."} </code> arrête également complètement le coéquipier, correspondant au comportement du hook <code>Stop</code>
PreToolUse	<code>hookSpecificOutput</code>	<code>permissionDecision</code> (allow/deny/ask), <code>permissionDecisionReason</code>
PermissionRequest	<code>hookSpecificOutput</code>	<code>decision.behavior</code> (allow/deny)
WorktreeCreate	chemin stdout	Le hook imprime le chemin absolu du worktree créé. La sortie non-zéro échoue la création
WorktreeRemove, Notification, SessionEnd, PreCompact, InstructionsLoaded	Aucun	Pas de contrôle de décision. Utilisé pour les effets secondaires comme la journalisation ou le nettoyage

Voici des exemples de chaque modèle en action :

Décision au niveau supérieur

Utilisé par `UserPromptSubmit`, `PostToolUse`, `PostToolUseFailure`, `Stop`, `SubagentStop` et `ConfigChange`. La seule valeur est `"block"`. Pour autoriser l'action à procéder, omettez `decision` de votre JSON ou quittez o sans aucun JSON :

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

Utilise `hookSpecificOutput` pour un contrôle plus riche : autoriser, refuser ou escalader à l'utilisateur. Vous pouvez également modifier l'entrée de l'outil avant son exécution ou injecter du contexte supplémentaire pour Claude. Consultez [Contrôle de décision PreToolUse](#) pour l'ensemble complet des options.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

Utilise `hookSpecificOutput` pour autoriser ou refuser une demande de permission au nom de l'utilisateur. Lors de l'autorisation, vous pouvez également modifier l'entrée de l'outil ou appliquer des règles de permission afin que l'utilisateur ne soit pas invité à nouveau. Consultez [Contrôle de décision PermissionRequest](#) pour l'ensemble complet des options.

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Pour des exemples étendus incluant la validation de commandes Bash, le filtrage de prompts et les scripts d’approbation automatique, consultez [Ce que vous pouvez automatiser](#) dans le guide et la [implémentation de référence du validateur de commandes Bash](#).

Événements de hook

Chaque événement correspond à un point du cycle de vie de Claude Code où les hooks peuvent s’exécuter. Les sections ci-dessous sont ordonnées pour correspondre au cycle de vie : de la configuration de session à travers la boucle agentique jusqu’à la fin de session. Chaque section décrit quand l’événement se déclenche, quels matchers il supporte, l’entrée JSON qu’il reçoit et comment contrôler le comportement via la sortie.

SessionStart

S’exécute lorsque Claude Code démarre une nouvelle session ou reprend une session existante. Utile pour charger le contexte de développement comme les problèmes existants ou les modifications récentes de votre codebase, ou pour configurer les variables d’environnement. Pour le contexte statique qui ne nécessite pas de script, utilisez [CLAUDE.md](#) à la place.

SessionStart s’exécute à chaque session, donc gardez ces hooks rapides. Seuls les hooks `type: "command"` sont supportés.

La valeur du matcher correspond à la façon dont la session a été initiée :

Matcher	Quand il se déclenche
<code>startup</code>	Nouvelle session

Matcher	Quand il se déclenche
<code>resume</code>	<code>--resume</code> , <code>--continue</code> ou <code>/resume</code>
<code>clear</code>	<code>/clear</code>
<code>compact</code>	Compaction automatique ou manuelle

Entrée SessionStart

En plus des [champs d'entrée communs](#), les hooks SessionStart reçoivent `source` , `model` et optionnellement `agent_type` . Le champ `source` indique comment la session a commencé : `"startup"` pour les nouvelles sessions, `"resume"` pour les sessions reprises, `"clear"` après `/clear` ou `"compact"` après compaction. Le champ `model` contient l'identifiant du modèle. Si vous démarrez Claude Code avec `claude --agent <name>` , un champ `agent_type` contient le nom de l'agent.

```
{
  "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"
}
```

Contrôle de décision SessionStart

Tout texte que votre script de hook imprime sur stdout est ajouté en tant que contexte pour Claude. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, vous pouvez retourner ces champs spécifiques à l'événement :

Champ	Description
<code>additionalContext</code>	Chaîne ajoutée au contexte de Claude. Les valeurs de plusieurs hooks sont concaténées

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "My additional context here"
  }
}
```

Persister les variables d'environnement

Les hooks `SessionStart` ont accès à la variable d'environnement `CLAUDE_ENV_FILE`, qui fournit un chemin de fichier où vous pouvez persister les variables d'environnement pour les commandes Bash suivantes.

Pour définir des variables d'environnement individuelles, écrivez des instructions `export` dans `CLAUDE_ENV_FILE`. Utilisez l'ajout (`>>`) pour préserver les variables définies par d'autres hooks :

```
#!/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
```

Pour capturer tous les changements d'environnement à partir des commandes de configuration, comparez les variables exportées avant et après :


```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

## Exécutez vos commandes de configuration qui modifient l'environnement
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
```

Toutes les variables écrites dans ce fichier seront disponibles dans toutes les commandes Bash suivantes que Claude Code exécute pendant la session.

Note:

`CLAUDE_ENV_FILE` est disponible pour les hooks `SessionStart`. Les autres types de hooks n'ont pas accès à cette variable.

InstructionsLoaded

Se déclenche lorsqu'un fichier `CLAUDE.md` ou `.claude/rules/*.md` est chargé en contexte. Cet événement se déclenche au démarrage de la session pour les fichiers chargés avec impatience et à nouveau plus tard lorsque les fichiers sont chargés avec paresse, par exemple lorsque Claude accède à un sous-répertoire qui contient un `CLAUDE.md` imbriqué ou lorsque les règles conditionnelles avec le frontmatter `paths:` correspondent. Le hook ne supporte pas le blocage ou le contrôle de décision. Il s'exécute de manière asynchrone à des fins d'observabilité.

`InstructionsLoaded` ne supporte pas les matchers et se déclenche à chaque occurrence de chargement.

Entrée InstructionsLoaded

En plus des [champs d'entrée communs](#), les hooks `InstructionsLoaded` reçoivent ces champs :

Champ	Description
<code>file_path</code>	Chemin absolu vers le fichier d'instructions qui a été chargé
<code>memory_type</code>	Portée du fichier : <code>"User"</code> , <code>"Project"</code> , <code>"Local"</code> ou <code>"Managed"</code>
<code>load_reason</code>	Pourquoi le fichier a été chargé : <code>"session_start"</code> , <code>"nested_traversal"</code> , <code>"path_glob_match"</code> ou <code>"include"</code>
<code>globs</code>	Modèles de glob de chemin du frontmatter <code>paths</code> : du fichier, le cas échéant. Présent uniquement pour les chargements <code>path_glob_match</code>
<code>trigger_file_path</code>	Chemin vers le fichier dont l'accès a déclenché ce chargement, pour les chargements paresseux
<code>parent_file_path</code>	Chemin vers le fichier d'instructions parent qui a inclus celui-ci, pour les chargements <code>include</code>

```
{
  "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"
}
```

Contrôle de décision InstructionsLoaded

Les hooks `InstructionsLoaded` n'ont pas de contrôle de décision. Ils ne peuvent pas bloquer ou modifier le chargement des instructions. Utilisez cet événement pour la journalisation d'audit, le suivi de conformité ou l'observabilité.

UserPromptSubmit

S'exécute lorsque l'utilisateur soumet un prompt, avant que Claude ne le traite. Cela vous permet d'ajouter du contexte supplémentaire basé sur le prompt/conversation, de valider les prompts ou de bloquer certains types de prompts.

Entrée UserPromptSubmit

En plus des [champs d'entrée communs](#), les hooks UserPromptSubmit reçoivent le champ `prompt` contenant le texte que l'utilisateur a soumis.

```
{
  "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"
}
```

Contrôle de décision UserPromptSubmit

Les hooks `UserPromptSubmit` peuvent contrôler si un prompt utilisateur est traité et ajouter du contexte. Tous les [champs de sortie JSON](#) sont disponibles.

Il y a deux façons d'ajouter du contexte à la conversation sur exit code 0 :

- **Stdout en texte brut** : tout texte non-JSON écrit sur stdout est ajouté en tant que contexte
- **JSON avec `additionalContext`** : utilisez le format JSON ci-dessous pour plus de contrôle. Le champ `additionalContext` est ajouté en tant que contexte

Le stdout brut est affiché comme sortie de hook dans la transcription. Le champ `additionalContext` est ajouté plus discrètement.

Pour bloquer un prompt, retournez un objet JSON avec `decision` défini à `"block"` :

Champ	Description
<code>decision</code>	<code>"block"</code> empêche le prompt d'être traité et l'efface du contexte. Omettez pour autoriser le prompt à procéder
<code>reason</code>	Affiché à l'utilisateur lorsque <code>decision</code> est <code>"block"</code> . Non ajouté au contexte
<code>additionalContext</code>	Chaîne ajoutée au contexte de Claude

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here"
  }
}
```

Note:

Le format JSON n'est pas requis pour les cas simples. Pour ajouter du contexte, vous pouvez imprimer du texte brut sur stdout avec exit code 0. Utilisez JSON lorsque vous avez besoin de bloquer des prompts ou que vous voulez un contrôle plus structuré.

PreToolUse

S'exécute après que Claude crée les paramètres de l'outil et avant le traitement de l'appel d'outil. Correspond au nom de l'outil : `Bash` , `Edit` , `Write` , `Read` , `Glob` , `Grep` , `Agent` , `WebFetch` , `WebSearch` et tout [nom d'outil MCP](#).

Utilisez [Contrôle de décision PreToolUse](#) pour autoriser, refuser ou demander la permission d'utiliser l'outil.

Entrée PreToolUse

En plus des [champs d'entrée communs](#), les hooks PreToolUse reçoivent `tool_name` , `tool_input` et `tool_use_id` . Les champs `tool_input` dépendent de l'outil :

Bash

Exécute les commandes shell.

Champ	Type	Exemple	Description
<code>command</code>	string	<code>"npm test"</code>	La commande shell à exécuter
<code>description</code>	string	<code>"Run test suite"</code>	Description option-nelle de ce que fait la commande

Champ	Type	Exemple	Description
<code>timeout</code>	number	<code>120000</code>	Délai d'expiration optionnel en millisecondes
<code>run_in_background</code>	boolean	<code>false</code>	Si la commande doit s'exécuter en arrière-plan

Write

Crée ou écrase un fichier.

Champ	Type	Exemple	Description
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Chemin absolu vers le fichier à écrire
<code>content</code>	string	<code>"file content"</code>	Contenu à écrire dans le fichier

Edit

Remplace une chaîne dans un fichier existant.

Champ	Type	Exemple	Description
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Chemin absolu vers le fichier à éditer
<code>old_string</code>	string	<code>"original text"</code>	Texte à trouver et remplacer
<code>new_string</code>	string	<code>"replacement text"</code>	Texte de remplacement
<code>replace_all</code>	boolean	<code>false</code>	Si toutes les occurrences doivent être remplacées

Read

Lit le contenu des fichiers.

Champ	Type	Exemple	Description
<code>file_path</code>	string	<code>"/path/to/file.txt"</code>	Chemin absolu vers le fichier à lire
<code>offset</code>	number	<code>10</code>	Numéro de ligne optionnel à partir duquel commencer la lecture
<code>limit</code>	number	<code>50</code>	Nombre optionnel de lignes à lire

Glob

Trouve les fichiers correspondant à un modèle glob.

Champ	Type	Exemple	Description
<code>pattern</code>	string	<code>"**/*.ts"</code>	Modèle glob pour correspondre aux fichiers
<code>path</code>	string	<code>"/path/to/dir"</code>	Répertoire optionnel à rechercher. Par défaut le répertoire de travail actuel

Grep

Recherche le contenu des fichiers avec des expressions régulières.

Champ	Type	Exemple	Description
<code>pattern</code>	string	<code>"TODO.*fix"</code>	Modèle d'expression régulière à rechercher
<code>path</code>	string	<code>"/path/to/dir"</code>	Fichier ou répertoire optionnel à rechercher
<code>glob</code>	string	<code>"*.ts"</code>	Modèle glob optionnel pour filtrer les fichiers

Champ	Type	Exemple	Description
<code>output_mode</code>	string	<code>"content"</code>	<code>"content"</code> , <code>"files_with_matches"</code> ou <code>"count"</code> . Par défaut <code>"files_with_matches"</code>
<code>-i</code>	boolean	<code>true</code>	Recherche insensible à la casse
<code>multiline</code>	boolean	<code>false</code>	Activer la correspondance multiligne

WebFetch

Récupère et traite le contenu web.

Champ	Type	Exemple	Description
<code>url</code>	string	<code>"https://example.com/api"</code>	URL à partir de laquelle récupérer le contenu
<code>prompt</code>	string	<code>"Extract the API endpoints"</code>	Prompt à exécuter sur le contenu récupéré

WebSearch

Recherche sur le web.

Champ	Type	Exemple	Description
<code>query</code>	string	<code>"react hooks best practices"</code>	Requête de recherche
<code>allowed_domains</code>	array	<code>["docs.example.com"]</code>	Optionnel : inclure uniquement les résultats de ces domaines
<code>blocked_domains</code>	array	<code>["spam.example.com"]</code>	Optionnel : exclure les résultats de ces domaines

Agent

Lance un [subagent](#).

Champ	Type	Exemple	Description
<code>prompt</code>	string	<code>"Find all API endpoints"</code>	La tâche pour l'agent à effectuer
<code>description</code>	string	<code>"Find API endpoints"</code>	Description courte de la tâche
<code>subagent_type</code>	string	<code>"Explore"</code>	Type d'agent spécialisé à utiliser
<code>model</code>	string	<code>"sonnet"</code>	Alias de modèle optionnel pour remplacer la valeur par défaut

Contrôle de décision PreToolUse

Les hooks `PreToolUse` peuvent contrôler si un appel d'outil procède. Contrairement aux autres hooks qui utilisent un champ `decision` au niveau supérieur, `PreToolUse` retourne sa décision à l'intérieur d'un objet `hookSpecificOutput`. Cela lui donne un contrôle plus riche : trois résultats (autoriser, refuser ou demander) plus la capacité de modifier l'entrée de l'outil avant l'exécution.

Champ	Description
<code>permissionDecision</code>	<code>"allow"</code> contourne le système de permission, <code>"deny"</code> empêche l'appel d'outil, <code>"ask"</code> demande à l'utilisateur de confirmer
<code>permissionDecisionReason</code>	Pour <code>"allow"</code> et <code>"ask"</code> , affiché à l'utilisateur mais pas à Claude. Pour <code>"deny"</code> , affiché à Claude
<code>updatedInput</code>	Modifie les paramètres d'entrée de l'outil avant l'exécution. Combinez avec <code>"allow"</code> pour approuver automatiquement ou <code>"ask"</code> pour afficher l'entrée modifiée à l'utilisateur
<code>additionalContext</code>	Chaîne ajoutée au contexte de Claude avant l'exécution de l'outil


```

{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}

```

Note:

PreToolUse utilisait auparavant les champs `decision` et `reason` au niveau supérieur, mais ceux-ci sont dépréciés pour cet événement. Utilisez `hookSpecificOutput.permissionDecision` et `hookSpecificOutput.permissionDecisionReason` à la place. Les valeurs dépréciées `"approve"` et `"block"` correspondent à `"allow"` et `"deny"` respectivement. D'autres événements comme PostToolUse et Stop continuent d'utiliser `decision` et `reason` au niveau supérieur comme format actuel.

PermissionRequest

S'exécute lorsque l'utilisateur est présenté avec un dialogue de permission. Utilisez [Contrôle de décision PermissionRequest](#) pour autoriser ou refuser au nom de l'utilisateur.

Correspond au nom de l'outil, mêmes valeurs que PreToolUse.

Entrée PermissionRequest

Les hooks PermissionRequest reçoivent les champs `tool_name` et `tool_input` comme les hooks PreToolUse, mais sans `tool_use_id`. Un tableau optionnel `permission_suggestions` contient les options « toujours autoriser » que l'utilisateur verrait normalement dans le dialogue de permission. La différence est quand le hook se déclenche : les hooks PermissionRequest s'exécutent lorsqu'un dialogue de permission est sur le point d'être affiché à l'utilisateur, tandis que les hooks PreToolUse s'exécutent avant l'exécution de l'outil indépendamment du statut de permission.

```
{
  "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" }
  ]
}
```

Contrôle de décision PermissionRequest

Les hooks `PermissionRequest` peuvent autoriser ou refuser les demandes de permission. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, votre script de hook peut retourner un objet `decision` avec ces champs spécifiques à l'événement :

Champ	Description
<code>behavior</code>	<code>"allow"</code> accorde la permission, <code>"deny"</code> la refuse
<code>updatedInput</code>	Pour <code>"allow"</code> uniquement : modifie les paramètres d'entrée de l'outil avant l'exécution
<code>updatedPermissions</code>	Pour <code>"allow"</code> uniquement : applique les mises à jour des règles de permission, équivalent à l'utilisateur sélectionnant une option « toujours autoriser »
<code>message</code>	Pour <code>"deny"</code> uniquement : indique à Claude pourquoi la permission a été refusée
<code>interrupt</code>	Pour <code>"deny"</code> uniquement : si <code>true</code> , arrête Claude

```

{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
}
}

```

PostToolUse

S'exécute immédiatement après qu'un outil se termine avec succès.

Correspond au nom de l'outil, mêmes valeurs que PreToolUse.

Entrée PostToolUse

Les hooks `PostToolUse` se déclenchent après qu'un outil s'est déjà exécuté avec succès.

L'entrée inclut à la fois `tool_input`, les arguments envoyés à l'outil, et `tool_response`, le résultat qu'il a retourné. Le schéma exact pour les deux dépend de l'outil.

```
{
  "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..."
}
```

Contrôle de décision PostToolUse

Les hooks `PostToolUse` peuvent fournir des commentaires à Claude après l'exécution de l'outil. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, votre script de hook peut retourner ces champs spécifiques à l'événement :

Champ	Description
<code>decision</code>	<code>"block"</code> demande à Claude avec la <code>reason</code> . Omettez pour autoriser l'action à procéder
<code>reason</code>	Explication affichée à Claude lorsque <code>decision</code> est <code>"block"</code>
<code>additionalContext</code>	Contexte supplémentaire pour Claude à considérer
<code>updatedMCPToolOutput</code>	Pour les outils MCP uniquement : remplace la sortie de l'outil par la valeur fournie

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude"
  }
}
```

PostToolUseFailure

S'exécute lorsqu'une exécution d'outil échoue. Cet événement se déclenche pour les appels d'outil qui lèvent des erreurs ou retournent des résultats d'échec. Utilisez ceci pour enregistrer les défaillances, envoyer des alertes ou fournir des commentaires correctifs à Claude.

Correspond au nom de l'outil, mêmes valeurs que PreToolUse.

Entrée PostToolUseFailure

Les hooks PostToolUseFailure reçoivent les mêmes champs `tool_name` et `tool_input` que PostToolUse, ainsi que les informations d'erreur en tant que champs au niveau supérieur :

```
{
  "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
}
```

Champ	Description
<code>error</code>	Chaîne décrivant ce qui s'est mal passé
<code>is_interrupt</code>	Booléen optionnel indiquant si l'échec a été causé par une interruption utilisateur

Contrôle de décision `PostToolUseFailure`

Les hooks `PostToolUseFailure` peuvent fournir du contexte à Claude après l'échec d'un outil. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, votre script de hook peut retourner ces champs spécifiques à l'événement :

Champ	Description
<code>additionalContext</code>	Contexte supplémentaire pour Claude à considérer aux côtés de l'erreur

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

Notification

S'exécute lorsque Claude Code envoie des notifications. Correspond au type de notification : `permission_prompt`, `idle_prompt`, `auth_success`, `elicitation_dialog`. Omettez le matcher pour exécuter les hooks pour tous les types de notification.

Utilisez des matchers séparés pour exécuter différents gestionnaires selon le type de notification. Cette configuration déclenche un script d'alerte spécifique à la permission lorsque Claude a besoin d'approbation de permission et une notification différente lorsque Claude a été inactif :

```

{
  "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"
          }
        ]
      }
    ]
  }
}

```

Entrée Notification

En plus des [champs d'entrée communs](#), les hooks Notification reçoivent `message` avec le texte de notification, un `title` optionnel et `notification_type` indiquant quel type s'est déclenché.


```
{
  "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"
}
```

Les hooks Notification ne peuvent pas bloquer ou modifier les notifications. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, vous pouvez retourner `additionalContext` pour ajouter du contexte à la conversation :

Champ	Description
<code>additionalContext</code>	Chaîne ajoutée au contexte de Claude

SubagentStart

S'exécute lorsqu'un subagent Claude Code est lancé via l'outil Agent. Supporte les matchers pour filtrer par nom de type d'agent (agents intégrés comme `Bash` , `Explore` , `Plan` ou noms d'agents personnalisés de `.claude/agents/`).

Entrée SubagentStart

En plus des [champs d'entrée communs](#), les hooks SubagentStart reçoivent `agent_id` avec l'identifiant unique du subagent et `agent_type` avec le nom de l'agent (agents intégrés comme `"Bash"` , `"Explore"` , `"Plan"` ou noms d'agents personnalisés).

```
{
  "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"
}
```

Les hooks SubagentStart ne peuvent pas bloquer la création de subagent, mais ils peuvent injecter du contexte dans le subagent. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, vous pouvez retourner :

Champ	Description
<code>additionalContext</code>	Chaîne ajoutée au contexte du subagent

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

SubagentStop

S'exécute lorsqu'un subagent Claude Code a terminé sa réponse. Correspond au type d'agent, mêmes valeurs que SubagentStart.

Entrée SubagentStop

En plus des [champs d'entrée communs](#), les hooks SubagentStop reçoivent `stop_hook_active`, `agent_id`, `agent_type`, `agent_transcript_path` et `last_assistant_message`. Le champ `agent_type` est la valeur utilisée pour le filtrage du matcher. Le `transcript_path` est la transcription de la session principale, tandis que `agent_transcript_path` est la propre transcription du subagent stockée dans un dossier

`subagents/` imbriqué. Le champ `last_assistant_message` contient le contenu textuel de la réponse finale du subagent, donc les hooks peuvent y accéder sans analyser le fichier de transcription.

```
{
  "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..."
}
```

Les hooks `SubagentStop` utilisent le même format de contrôle de décision que les [hooks Stop](#).

Stop

S'exécute lorsque l'agent Claude Code principal a terminé sa réponse. Ne s'exécute pas si l'arrêt s'est produit en raison d'une interruption utilisateur.

Entrée Stop

En plus des [champs d'entrée communs](#), les hooks Stop reçoivent `stop_hook_active` et `last_assistant_message`. Le champ `stop_hook_active` est `true` lorsque Claude Code continue déjà en raison d'un hook stop. Vérifiez cette valeur ou traitez la transcription pour empêcher Claude Code de s'exécuter indéfiniment. Le champ `last_assistant_message` contient le contenu textuel de la réponse finale de Claude, donc les hooks peuvent y accéder sans analyser le fichier de transcription.

```
{
  "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..."
}
```

Contrôle de décision Stop

Les hooks `Stop` et `SubagentStop` peuvent contrôler si Claude continue. En plus des [champs de sortie JSON](#) disponibles pour tous les hooks, votre script de hook peut retourner ces champs spécifiques à l'événement :

Champ	Description
<code>decision</code>	<code>"block"</code> empêche Claude de s'arrêter. Omettez pour autoriser Claude à s'arrêter
<code>reason</code>	Requis lorsque <code>decision</code> est <code>"block"</code> . Indique à Claude pourquoi il doit continuer

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

TeammateIdle

S'exécute lorsqu'un coéquipier d'une [équipe d'agents](#) est sur le point de devenir inactif après avoir terminé son tour. Utilisez ceci pour appliquer des portes de qualité avant qu'un coéquipier ne cesse de travailler, comme exiger des vérifications de lint réussies ou vérifier que les fichiers de sortie existent.

Lorsqu'un hook `TeammateIdle` quitte avec le code 2, le coéquipier reçoit le message stderr en tant que commentaire et continue de travailler au lieu de devenir inactif. Pour arrêter complètement le coéquipier au lieu de le relancer, retournez JSON avec `{"continue": false, "stopReason": "..."} .` Les hooks `TeammateIdle` ne supportent pas les matchers et se déclenchent à chaque occurrence.

Entrée `TeammateIdle`

En plus des [champs d'entrée communs](#), les hooks `TeammateIdle` reçoivent `teammate_name` et `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"
}
```

Champ	Description
<code>teammate_name</code>	Nom du coéquipier qui est sur le point de devenir inactif
<code>team_name</code>	Nom de l'équipe

Contrôle de décision `TeammateIdle`

Les hooks `TeammateIdle` supportent deux façons de contrôler le comportement du coéquipier :

- **Code de sortie 2** : le coéquipier reçoit le message stderr en tant que commentaire et continue de travailler au lieu de devenir inactif.
- **JSON** `{"continue": false, "stopReason": "..."} :` arrête complètement le coéquipier, correspondant au comportement du hook `Stop` . Le `stopReason` est affiché à l'utilisateur.

Cet exemple vérifie qu'un artefact de build existe avant d'autoriser un coéquipier à devenir inactif :

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
    echo "Build artifact missing. Run the build before stopping." >&2
    exit 2
fi

exit 0
```

TaskCompleted

S'exécute lorsqu'une tâche est marquée comme complétée. Cela se déclenche dans deux situations : lorsqu'un agent marque explicitement une tâche comme complétée via l'outil TaskUpdate, ou lorsqu'un coéquipier d'une [équipe d'agents](#) termine son tour avec des tâches en cours. Utilisez ceci pour appliquer les critères d'achèvement comme passer les tests ou les vérifications de lint avant qu'une tâche ne puisse se fermer.

Lorsqu'un hook `TaskCompleted` quitte avec le code 2, la tâche n'est pas marquée comme complétée et le message stderr est renvoyé au modèle en tant que commentaire. Pour arrêter complètement le coéquipier au lieu de le relancer, retournez JSON avec `{"continue": false, "stopReason": "..."}.` Les hooks TaskCompleted ne supportent pas les matchers et se déclenchent à chaque occurrence.

Entrée TaskCompleted

En plus des [champs d'entrée communs](#), les hooks TaskCompleted reçoivent `task_id`, `task_subject` et optionnellement `task_description`, `teammate_name` et `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"
}
```

Champ	Description
<code>task_id</code>	Identifiant de la tâche en cours de complétion
<code>task_subject</code>	Titre de la tâche
<code>task_description</code>	Description détaillée de la tâche. Peut être absent
<code>teammate_name</code>	Nom du coéquipier complétant la tâche. Peut être absent
<code>team_name</code>	Nom de l'équipe. Peut être absent

Contrôle de décision TaskCompleted

Les hooks TaskCompleted supportent deux façons de contrôler l'achèvement de la tâche :

- **Code de sortie 2** : la tâche n'est pas marquée comme complétée et le message stderr est renvoyé au modèle en tant que commentaire.
- **JSON** `{"continue": false, "stopReason": "..."} :` arrête complètement le coéquipier, correspondant au comportement du hook `Stop` . Le `stopReason` est affiché à l'utilisateur.

Cet exemple exécute les tests et bloque l'achèvement de la tâche s'ils échouent :

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

## Exécutez la suite de tests
if ! npm test 2>&1; then
    echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
    exit 2
fi

exit 0
```

ConfigChange

S'exécute lorsqu'un fichier de configuration change pendant une session. Utilisez ceci pour auditer les modifications de paramètres, appliquer les politiques de sécurité ou bloquer les modifications non autorisées aux fichiers de configuration.

Les hooks ConfigChange se déclenchent pour les modifications des fichiers de paramètres, les paramètres de politique gérée et les fichiers de skill. Le champ `source` dans l'entrée vous indique quel type de configuration a changé, et le champ optionnel `file_path` fournit le chemin vers le fichier modifié.

Le matcher filtre sur la source de configuration :

Matcher	Quand il se déclenche
<code>user_settings</code>	<code>~/.claude/settings.json</code> change
<code>project_settings</code>	<code>.claude/settings.json</code> change
<code>local_settings</code>	<code>.claude/settings.local.json</code> change
<code>policy_settings</code>	Les paramètres de politique gérée changent
<code>skills</code>	Un fichier de skill dans <code>.claude/skills/</code> change

Cet exemple enregistre toutes les modifications de configuration pour l'audit de sécurité :


```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/audit-config-
change.sh"
          }
        ]
      }
    ]
  }
}
```

Entrée ConfigChange

En plus des [champs d'entrée communs](#), les hooks ConfigChange reçoivent `source` et optionnellement `file_path`. Le champ `source` indique quel type de configuration a changé, et `file_path` fournit le chemin vers le fichier spécifique qui a été modifié.

```
{
  "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"
}
```

Contrôle de décision ConfigChange

Les hooks ConfigChange peuvent bloquer les modifications de configuration de prendre effet. Utilisez le code de sortie 2 ou un JSON `decision` pour empêcher la modification. Lorsqu'elle est bloquée, les nouveaux paramètres ne sont pas appliqués à la session en cours.

Champ	Description
<code>decision</code>	"block" empêche la modification de configuration de prendre effet. Omettez pour autoriser la modification
<code>reason</code>	Explication affichée à l'utilisateur lorsque <code>decision</code> est "block"

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

Les modifications `policy_settings` ne peuvent pas être bloquées. Les hooks se déclenchent toujours pour les sources `policy_settings`, vous pouvez donc les utiliser pour la journalisation d'audit, mais toute décision de blocage est ignorée. Cela garantit que les paramètres gérés par l'entreprise prennent toujours effet.

WorktreeCreate

Lorsque vous exécutez `claude --worktree` ou qu'un [subagent utilise isolation: "worktree"](#), Claude Code crée une copie de travail isolée en utilisant `git worktree`. Si vous configurez un hook WorktreeCreate, il remplace le comportement git par défaut, vous permettant d'utiliser un système de contrôle de version différent comme SVN, Perforce ou Mercurial.

Le hook doit imprimer le chemin absolu du répertoire worktree créé sur stdout. Claude Code utilise ce chemin comme répertoire de travail pour la session isolée.

Cet exemple crée une copie de travail SVN et imprime le chemin pour que Claude Code l'utilise. Remplacez l'URL du référentiel par la vôtre :

```
{
  "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\\\"'"
          }
        ]
      }
    ]
  }
}
```

Le hook lit le **name** du worktree depuis l'entrée JSON sur stdin, extrait une copie fraîche dans un nouveau répertoire et imprime le chemin du répertoire. Le **echo** sur la dernière ligne est ce que Claude Code lit comme le chemin du worktree. Redirigez toute autre sortie vers stderr afin qu'elle n'interfère pas avec le chemin.

Entrée WorktreeCreate

En plus des [champs d'entrée communs](#), les hooks WorktreeCreate reçoivent le champ **name**. C'est un identifiant slug pour le nouveau worktree, soit spécifié par l'utilisateur, soit généré automatiquement (par exemple, **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"
}
```

Sortie WorktreeCreate

Le hook doit imprimer le chemin absolu du répertoire worktree créé sur stdout. Si le hook échoue ou ne produit aucune sortie, la création du worktree échoue avec une erreur.

Les hooks WorktreeCreate n'utilisent pas le modèle de décision autoriser/bloquer standard. Au lieu de cela, le succès ou l'échec du hook détermine le résultat. Seuls les hooks `type: "command"` sont supportés.

WorktreeRemove

L'équivalent de nettoyage de [WorktreeCreate](#). Ce hook se déclenche lorsqu'un worktree est en cours de suppression, soit lorsque vous quittez une session `--worktree` et choisissez de la supprimer, soit lorsqu'un subagent avec `isolation: "worktree"` se termine. Pour les worktrees basés sur git, Claude gère le nettoyage automatiquement avec `git worktree remove`. Si vous avez configuré un hook WorktreeCreate pour un système de contrôle de version non-git, associez-le à un hook WorktreeRemove pour gérer le nettoyage. Sans lui, le répertoire worktree est laissé sur le disque.

Claude Code transmet le chemin que WorktreeCreate a imprimé sur stdout en tant que `worktree_path` dans l'entrée du hook. Cet exemple lit ce chemin et supprime le répertoire :

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

Entrée WorktreeRemove

En plus des [champs d'entrée communs](#), les hooks WorktreeRemove reçoivent le champ `worktree_path`, qui est le chemin absolu du worktree en cours de suppression.

```
{
  "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"
}
```

Les hooks `WorktreeRemove` n'ont pas de contrôle de décision. Ils ne peuvent pas bloquer la suppression du worktree mais peuvent effectuer des tâches de nettoyage comme supprimer l'état du contrôle de version ou archiver les modifications. Les défaillances des hooks sont enregistrées en mode debug uniquement. Seuls les hooks `type: "command"` sont supportés.

PreCompact

S'exécute avant que Claude Code ne soit sur le point d'exécuter une opération de compaction.

La valeur du `matcher` indique si la compaction a été déclenchée manuellement ou automatiquement :

Mat-cher	Quand il se déclenche
<code>manual</code>	<code>/compact</code>
<code>auto</code>	Compaction automatique lorsque la fenêtre de contexte est pleine

Entrée PreCompact

En plus des [champs d'entrée communs](#), les hooks `PreCompact` reçoivent `trigger` et `custom_instructions`. Pour `manual`, `custom_instructions` contient ce que l'utilisateur transmet dans `/compact`. Pour `auto`, `custom_instructions` est vide.

```
{
  "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

S'exécute lorsqu'une session Claude Code se termine. Utile pour les tâches de nettoyage, la journalisation des statistiques de session ou l'enregistrement de l'état de session. Supporte les matchers pour filtrer par raison de sortie.

Le champ `reason` dans l'entrée du hook indique pourquoi la session s'est terminée :

Raison	Description
<code>clear</code>	Session effacée avec la commande <code>/clear</code>
<code>logout</code>	L'utilisateur s'est déconnecté
<code>prompt_input_exit</code>	L'utilisateur a quitté pendant que l'entrée du prompt était visible
<code>bypass_permissions_disabled</code>	Le mode de permissions de contournement a été désactivé
<code>other</code>	Autres raisons de sortie

Entrée SessionEnd

En plus des [champs d'entrée communs](#), les hooks SessionEnd reçoivent un champ `reason` indiquant pourquoi la session s'est terminée. Consultez le [tableau des raisons](#) ci-dessus pour toutes les valeurs.

```
{
  "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"
}
```

Les hooks `SessionEnd` n'ont pas de contrôle de décision. Ils ne peuvent pas bloquer la terminaison de session mais peuvent effectuer des tâches de nettoyage.

Hooks basés sur des prompts

En plus des hooks de commande et HTTP, Claude Code supporte les hooks basés sur des prompts (`type: "prompt"`) qui utilisent un LLM pour évaluer s'il faut autoriser ou bloquer une action, et les hooks d'agent (`type: "agent"`) qui lancent un vérificateur agentique avec accès aux outils. Tous les événements ne supportent pas tous les types de hooks.

Les événements qui supportent les quatre types de hooks (`command` , `http` , `prompt` et `agent`) :

- `PermissionRequest`
- `PostToolUse`
- `PostToolUseFailure`
- `PreToolUse`
- `Stop`
- `SubagentStop`
- `TaskCompleted`
- `UserPromptSubmit`

Les événements qui supportent uniquement les hooks `type: "command"` :

- `ConfigChange`
- `InstructionsLoaded`
- `Notification`
- `PreCompact`
- `SessionEnd`

- `SessionStart`
- `SubagentStart`
- `TeammateIdle`
- `WorktreeCreate`
- `WorktreeRemove`

Comment fonctionnent les hooks basés sur des prompts

Au lieu d'exécuter une commande Bash, les hooks basés sur des prompts :

1. Envoyent l'entrée du hook et votre prompt à un modèle Claude, Haiku par défaut
2. Le LLM répond avec JSON structuré contenant une décision
3. Claude Code traite automatiquement la décision

Configuration des hooks de prompt

Définissez `type` à `"prompt"` et fournissez une chaîne `prompt` au lieu d'une `command`. Utilisez le placeholder `$ARGUMENTS` pour injecter les données d'entrée JSON du hook dans votre texte de prompt. Claude Code envoie le prompt combiné et l'entrée à un modèle Claude rapide, qui retourne une décision JSON.

Ce hook `Stop` demande au LLM d'évaluer si Claude doit s'arrêter avant d'autoriser Claude à terminer :

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all
tasks are complete."
          }
        ]
      }
    ]
  }
}
```


Champ	Requis	Description
type	oui	Doit être "prompt"
prompt	oui	Le texte du prompt à envoyer au LLM. Utilisez \$ARGUMENTS comme placeholder pour l'entrée JSON du hook. Si \$ARGUMENTS n'est pas présent, l'entrée JSON est ajoutée au prompt
model	non	Modèle à utiliser pour l'évaluation. Par défaut un modèle rapide
timeout	non	Délai d'expiration en secondes. Par défaut : 30

Schéma de réponse

Le LLM doit répondre avec JSON contenant :

```
{
  "ok": true | false,
  "reason": "Explanation for the decision"
}
```

Champ	Description
ok	true autorise l'action, false l'empêche
reason	Requis lorsque ok est false . Explication affichée à Claude

Exemple : Hook Stop multi-critères

Ce hook **Stop** utilise un prompt détaillé pour vérifier trois conditions avant d'autoriser Claude à s'arrêter. Si **"ok"** est **false** , Claude continue de travailler avec la raison fournie comme sa prochaine instruction. Les hooks **SubagentStop** utilisent le même format pour évaluer si un **subagent** doit s'arrêter :

```

{
  "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
          }
        ]
      }
    ]
  }
}

```

Hooks basés sur des agents

Les hooks basés sur des agents (`type: "agent"`) sont comme les hooks basés sur des prompts mais avec accès aux outils multi-tours. Au lieu d'un seul appel LLM, un hook d'agent lance un subagent qui peut lire des fichiers, rechercher du code et inspecter la codebase pour vérifier les conditions. Les hooks d'agent supportent les mêmes événements que les hooks basés sur des prompts.

Comment fonctionnent les hooks d'agent

Lorsqu'un hook d'agent se déclenche :

1. Claude Code lance un subagent avec votre prompt et l'entrée JSON du hook
2. Le subagent peut utiliser des outils comme Read, Grep et Glob pour enquêter
3. Après jusqu'à 50 tours, le subagent retourne une décision structurée `{ "ok": true/false }`
4. Claude Code traite la décision de la même manière qu'un hook de prompt

Les hooks d'agent sont utiles lorsque la vérification nécessite d'inspecter les fichiers réels ou la sortie des tests, pas seulement d'évaluer les données d'entrée du hook seules.

Configuration des hooks d'agent

Définissez `type` à `"agent"` et fournissez une chaîne `prompt`. Les champs de configuration sont les mêmes que les [hooks de prompt](#), avec un délai d'expiration par défaut plus long :

Champ	Requis	Description
<code>type</code>	oui	Doit être <code>"agent"</code>
<code>prompt</code>	oui	Prompt décrivant ce à vérifier. Utilisez <code>\$ARGUMENTS</code> comme placeholder pour l'entrée JSON du hook
<code>model</code>	non	Modèle à utiliser. Par défaut un modèle rapide
<code>timeout</code>	non	Délai d'expiration en secondes. Par défaut : 60

Le schéma de réponse est le même que les hooks de prompt : `{ "ok": true }` pour autoriser ou `{ "ok": false, "reason": "..."` pour bloquer.

Ce hook `Stop` vérifie que tous les tests unitaires réussissent avant d'autoriser Claude à terminer :

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Exécuter les hooks en arrière-plan

Par défaut, les hooks bloquent l'exécution de Claude jusqu'à ce qu'ils se terminent. Pour les tâches longues comme les déploiements, les suites de tests ou les appels API externes, définissez `"async": true` pour exécuter le hook en arrière-plan tandis que Claude continue de travailler. Les hooks asynchrones ne peuvent pas bloquer ou contrôler le comportement de Claude : les champs de réponse comme `decision`, `permissionDecision` et `continue` n'ont aucun effet, car l'action qu'ils auraient contrôlée s'est déjà produite.

Configurer un hook asynchrone

Ajoutez `"async": true` à la configuration d'un hook de commande pour l'exécuter en arrière-plan sans bloquer Claude. Ce champ n'est disponible que sur les hooks `type: "command"`.

Ce hook exécute un script de test après chaque appel d'outil `Write`. Claude continue de travailler immédiatement tandis que `run-tests.sh` s'exécute pendant jusqu'à 120 secondes. Lorsque le script se termine, sa sortie est livrée au tour de conversation suivant :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true,
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Le champ `timeout` définit le temps maximum en secondes pour le processus en arrière-plan. S'il n'est pas spécifié, les hooks asynchrones utilisent la même valeur par défaut de 10 minutes que les hooks synchrones.

Comment les hooks asynchrones s'exécutent

Lorsqu'un hook asynchrone se déclenche, Claude Code démarre le processus du hook et continue immédiatement sans attendre qu'il se termine. Le hook reçoit la même entrée JSON via stdin qu'un hook synchrone.

Après que le processus en arrière-plan se termine, si le hook a produit une réponse JSON avec un champ `systemMessage` ou `additionalContext`, ce contenu est livré à Claude en tant que contexte au tour de conversation suivant.

Exemple : exécuter les tests après les modifications de fichiers

Ce hook démarre une suite de tests en arrière-plan chaque fois que Claude écrit un fichier, puis rapporte les résultats à Claude lorsque les tests se terminent. Enregistrez ce script dans `.claude/hooks/run-tests-async.sh` dans votre projet et rendez-le exécutable avec `chmod +x` :

```
#!/bin/bash
## run-tests-async.sh

## Lire l'entrée du hook depuis stdin
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

## Exécutez les tests uniquement pour les fichiers source
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
    exit 0
fi

## Exécutez les tests et rapportez les résultats via 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
```

Ensuite, ajoutez cette configuration à `.claude/settings.json` dans la racine de votre projet. Le drapeau `async: true` permet à Claude de continuer à travailler pendant que les tests s'exécutent :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/run-tests-async.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}

```

Limitations

Les hooks asynchrones ont plusieurs contraintes par rapport aux hooks synchrones :

- Seuls les hooks `type: "command"` supportent `async`. Les hooks basés sur des prompts ne peuvent pas s'exécuter de manière asynchrone.
- Les hooks asynchrones ne peuvent pas bloquer les appels d'outil ou retourner des décisions. Au moment où le hook se termine, l'action qui l'a déclenché s'est déjà produite.
- La sortie du hook est livrée au tour de conversation suivant. Si la session est inactive, la réponse attend jusqu'à la prochaine interaction utilisateur.
- Chaque exécution crée un processus en arrière-plan séparé. Il n'y a pas de déduplication sur plusieurs déclenchements du même hook asynchrone.

Considérations de sécurité

Avertissement

Les hooks de commande s'exécutent avec les permissions complètes de votre utilisateur système.

Warning:

Les hooks de commande exécutent les commandes shell avec vos permissions utilisateur complètes. Ils peuvent modifier, supprimer ou accéder à tous les fichiers auxquels votre compte utilisateur peut accéder. Examinez et testez toutes les commandes de hook avant de les ajouter à votre configuration.

Meilleures pratiques de sécurité

Gardez ces pratiques à l'esprit lors de l'écriture de hooks :

- **Validez et nettoyez les entrées** : ne faites jamais confiance aux données d'entrée aveuglément
- **Toujours citer les variables shell** : utilisez `"$VAR"` pas `$VAR`
- **Bloquez la traversée de répertoires** : vérifiez les `..` dans les chemins de fichiers
- **Utilisez les chemins absolus** : spécifiez les chemins complets pour les scripts, en utilisant `"$CLAUDE_PROJECT_DIR"` pour la racine du projet
- **Ignorez les fichiers sensibles** : évitez `.env` , `.git/` , les clés, etc.

Débugger les hooks

Exécutez `claude --debug` pour voir les détails d'exécution des hooks, y compris les hooks qui ont correspondu, leurs codes de sortie et leur sortie. Basculez le mode verbeux avec `Ctrl+O` pour voir la progression du hook dans la transcription.

```
[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>
```

Pour dépanner les problèmes courants comme les hooks qui ne se déclenchent pas, les boucles infinies de hooks Stop ou les erreurs de configuration, consultez [Limitations et dépannage](#) dans le guide.

Automatiser les workflows avec les hooks

Exécutez automatiquement des commandes shell lorsque Claude Code modifie des fichiers, termine des tâches ou a besoin d'une entrée. Formatez le code, envoyez des notifications, validez les commandes et appliquez les règles du projet.

Les hooks sont des commandes shell définies par l'utilisateur qui s'exécutent à des points spécifiques du cycle de vie de Claude Code. Ils fournissent un contrôle déterministe du comportement de Claude Code, en garantissant que certaines actions se produisent toujours plutôt que de compter sur le LLM pour choisir de les exécuter. Utilisez les hooks pour appliquer les règles du projet, automatiser les tâches répétitives et intégrer Claude Code avec vos outils existants.

Pour les décisions qui nécessitent un jugement plutôt que des règles déterministes, vous pouvez également utiliser des [hooks basés sur des invites](#) ou des [hooks basés sur des agents](#) qui utilisent un modèle Claude pour évaluer les conditions.

Pour d'autres façons d'étendre Claude Code, consultez [skills](#) pour donner à Claude des instructions supplémentaires et des commandes exécutables, [subagents](#) pour exécuter des tâches dans des contextes isolés, et [plugins](#) pour emballer les extensions à partager entre les projets.

Tip:

Ce guide couvre les cas d'usage courants et comment commencer. Pour les schémas d'événements complets, les formats d'entrée/sortie JSON et les fonctionnalités avancées comme les hooks asynchrones et les hooks d'outils MCP, consultez la [référence des Hooks](#).

Configurer votre premier hook

Le moyen le plus rapide de créer un hook est via le menu interactif `/hooks` dans Claude Code. Cette procédure pas à pas crée un hook de notification de bureau, afin que vous soyez alerté chaque fois que Claude attend votre entrée au lieu de regarder le terminal.

Step 1: Ouvrir le menu des hooks

Tapez `/hooks` dans le CLI de Claude Code. Vous verrez une liste de tous les événements de hook disponibles, plus une option pour désactiver tous les hooks. Chaque événement correspond à un point du cycle de vie de Claude où vous pouvez exécuter du code personnalisé. Sélectionnez `Notification` pour créer un hook qui se déclenche lorsque Claude a besoin de votre attention.

Step 2: Configurer le matcher

Le menu affiche une liste de matchers, qui filtrent le moment où le hook se déclenche. Définissez le matcher sur `*` pour se déclencher sur tous les types de notifications. Vous pouvez l'affiner plus tard en changeant le matcher en une valeur spécifique comme `permission_prompt` ou `idle_prompt`.

Step 3: Ajouter votre commande

Sélectionnez `+ Add new hook...`. Le menu vous demande une commande shell à exécuter lorsque l'événement se déclenche. Les hooks exécutent n'importe quelle commande shell que vous fournissez, vous pouvez donc utiliser l'outil de notification intégré de votre plateforme. Copiez la commande pour votre système d'exploitation :

macOS

Utilise `osascript` pour déclencher une notification macOS native via AppleScript :

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

Linux

Utilise `notify-send`, qui est pré-installé sur la plupart des bureaux Linux avec un démon de notification :

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

Windows (PowerShell)

Utilise PowerShell pour afficher une boîte de message native via Windows Forms de .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: Choisir un emplacement de stockage

Le menu vous demande où enregistrer la configuration du hook. Sélectionnez **User settings** pour la stocker dans `~/.claude/settings.json`, ce qui applique le hook à tous vos projets. Vous pouvez également choisir **Project settings** pour le limiter au projet actuel. Consultez [Configurer l'emplacement du hook](#) pour tous les périmètres disponibles.

Step 5: Tester le hook

Appuyez sur **Esc** pour revenir au CLI. Demandez à Claude de faire quelque chose qui nécessite une permission, puis quittez le terminal. Vous devriez recevoir une notification de bureau.

Ce que vous pouvez automatiser

Les hooks vous permettent d'exécuter du code à des points clés du cycle de vie de Claude Code : formater les fichiers après les modifications, bloquer les commandes avant leur exécution, envoyer des notifications lorsque Claude a besoin d'une entrée, injecter du contexte au démarrage de la session, et bien plus. Pour la liste complète des événements de hook, consultez la [référence des Hooks](#).

Chaque exemple inclut un bloc de configuration prêt à l'emploi que vous ajoutez à un [fichier de paramètres](#). Les modèles les plus courants :

- [Être notifié lorsque Claude a besoin d'une entrée](#)
- [Formater automatiquement le code après les modifications](#)
- [Bloquer les modifications des fichiers protégés](#)
- [Réinjecter le contexte après compaction](#)
- [Auditer les modifications de configuration](#)

Être notifié lorsque Claude a besoin d'une entrée

Recevez une notification de bureau chaque fois que Claude termine son travail et a besoin de votre entrée, afin que vous puissiez passer à d'autres tâches sans vérifier le terminal.

Ce hook utilise l'événement **Notification**, qui se déclenche lorsque Claude attend une entrée ou une permission. Chaque onglet ci-dessous utilise la commande de notification native de la plateforme. Ajoutez ceci à `~/.claude/settings.json`, ou utilisez la [procédure pas à pas interactive](#) ci-dessus pour le configurer avec `/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]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.MessageBox]::Show('Claude Code needs your attention', 'Claude Code')\"";
          }
        ]
      }
    ]
  }
}

```

Formater automatiquement le code après les modifications

Exécutez automatiquement [Prettier](#) sur chaque fichier que Claude modifie, afin que le formatage reste cohérent sans intervention manuelle.

Ce hook utilise l'événement `PostToolUse` avec un matcher `Edit|Write`, il s'exécute donc uniquement après les outils d'édition de fichiers. La commande extrait le chemin du fichier modifié avec `jq` et le transmet à Prettier. Ajoutez ceci à `.claude/settings.json` dans la racine de votre projet :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --
write"
          }
        ]
      }
    ]
  }
}

```

Note:

Les exemples Bash sur cette page utilisent `jq` pour l'analyse JSON. Installez-le avec `brew install jq` (macOS), `apt-get install jq` (Debian/Ubuntu), ou consultez les [téléchargements de jq](#).

Bloquer les modifications des fichiers protégés

Empêchez Claude de modifier les fichiers sensibles comme `.env`, `package-lock.json`, ou n'importe quoi dans `.git/`. Claude reçoit un retour d'information expliquant pourquoi la modification a été bloquée, afin qu'il puisse ajuster son approche.

Cet exemple utilise un fichier de script séparé que le hook appelle. Le script vérifie le chemin du fichier cible par rapport à une liste de modèles protégés et quitte avec le code 2 pour bloquer la modification.

Step 1: Créer le script du hook

Enregistrez ceci dans `.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: Rendre le script exécutable (macOS/Linux)

Les scripts de hook doivent être exécutables pour que Claude Code les exécute :

```
chmod +x .claude/hooks/protect-files.sh
```

Step 3: Enregistrer le hook

Ajoutez un hook `PreToolUse` à `.claude/settings.json` qui exécute le script avant tout appel d'outil `Edit` ou `Write` :


```

{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}

```

Réinjecter le contexte après compaction

Lorsque la fenêtre de contexte de Claude se remplit, la compaction résume la conversation pour libérer de l'espace. Cela peut perdre des détails importants. Utilisez un hook `SessionStart` avec un matcher `compact` pour réinjecter le contexte critique après chaque compaction.

Tout texte que votre commande écrit sur stdout est ajouté au contexte de Claude. Cet exemple rappelle à Claude les conventions du projet et le travail récent. Ajoutez ceci à `.claude/settings.json` dans la racine de votre projet :

```

{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}

```

Vous pouvez remplacer `echo` par n'importe quelle commande qui produit une sortie dynamique, comme `git log --oneline -5` pour afficher les commits récents. Pour injecter du contexte au démarrage de chaque session, envisagez d'utiliser [CLAUDE.md](#) à la place. Pour les variables d'environnement, consultez [CLAUDE_ENV_FILE](#) dans la référence.

Auditer les modifications de configuration

Suivez les modifications des fichiers de paramètres ou de skills pendant une session. L'événement `ConfigChange` se déclenche lorsqu'un processus externe ou un éditeur modifie un fichier de configuration, afin que vous puissiez enregistrer les modifications pour la conformité ou bloquer les modifications non autorisées.

Cet exemple ajoute chaque modification à un journal d'audit. Ajoutez ceci à `~/.claude/settings.json` :

```

{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source,
file: .file_path}' >> ~/claude-config-audit.log"
          }
        ]
      }
    ]
  }
}

```

Le matcher filtre par type de configuration : `user_settings` , `project_settings` , `local_settings` , `policy_settings` , ou `skills` . Pour bloquer une modification de prendre effet, quittez avec le code 2 ou retournez `{"decision": "block"}` . Consultez la [référence ConfigChange](#) pour le schéma d'entrée complet.

Comment fonctionnent les hooks

Les événements de hook se déclenchent à des points spécifiques du cycle de vie de Claude Code. Lorsqu'un événement se déclenche, tous les hooks correspondants s'exécutent en parallèle, et les commandes de hook identiques sont automatiquement dédoublées. Le tableau ci-dessous montre chaque événement et quand il se déclenche :

Event	When it fires
<code>SessionStart</code>	When a session begins or resumes
<code>UserPromptSubmit</code>	When you submit a prompt, before Claude processes it
<code>PreToolUse</code>	Before a tool call executes. Can block it
<code>PermissionRequest</code>	When a permission dialog appears
<code>PostToolUse</code>	After a tool call succeeds

Event	When it fires
<code>PostToolUseFailure</code>	After a tool call fails
<code>Notification</code>	When Claude Code sends a notification
<code>SubagentStart</code>	When a subagent is spawned
<code>SubagentStop</code>	When a subagent finishes
<code>Stop</code>	When Claude finishes responding
<code>TeammateIdle</code>	When an agent team teammate is about to go idle
<code>TaskCompleted</code>	When a task is being marked as completed
<code>InstructionsLoaded</code>	When a <code>CLAUDE.md</code> or <code>.claude/rules/*.md</code> file is loaded into context. Fires at session start and when files are lazily loaded during a session
<code>ConfigChange</code>	When a configuration file changes during a session
<code>WorktreeCreate</code>	When a worktree is being created via <code>--worktree</code> or <code>isolation: "worktree"</code> . Replaces default git behavior
<code>WorktreeRemove</code>	When a worktree is being removed, either at session exit or when a subagent finishes
<code>PreCompact</code>	Before context compaction
<code>PostCompact</code>	After context compaction completes
<code>Elicitation</code>	When an MCP server requests user input during a tool call
<code>ElicitationResult</code>	After a user responds to an MCP elicitation, before the response is sent back to the server
<code>SessionEnd</code>	When a session terminates

Chaque hook a un `type` qui détermine comment il s'exécute. La plupart des hooks utilisent `"type": "command"`, qui exécute une commande shell. Trois autres types sont disponibles :

- `"type": "http"` : POST les données d'événement vers une URL. Consultez [HTTP hooks](#).
- `"type": "prompt"` : évaluation LLM à un seul tour. Consultez [Hooks basés sur des invites](#).

- **"type": "agent"** : vérification multi-tour avec accès aux outils. Consultez [Hooks basés sur des agents](#).

Lire l'entrée et retourner la sortie

Les hooks communiquent avec Claude Code via stdin, stdout, stderr et les codes de sortie. Lorsqu'un événement se déclenche, Claude Code transmet les données spécifiques à l'événement en JSON à stdin de votre script. Votre script lit ces données, fait son travail, et dit à Claude Code quoi faire ensuite via le code de sortie.

Entrée du hook

Chaque événement inclut des champs communs comme `session_id` et `cwd`, mais chaque type d'événement ajoute des données différentes. Par exemple, lorsque Claude exécute une commande Bash, un hook `PreToolUse` reçoit quelque chose comme ceci sur 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
  }
}
```

Votre script peut analyser ce JSON et agir sur n'importe lequel de ces champs. Les hooks `UserPromptSubmit` obtiennent le texte `prompt` à la place, les hooks `SessionStart` obtiennent la `source` (startup, resume, clear, compact), et ainsi de suite. Consultez [Champs d'entrée communs](#) dans la référence pour les champs partagés, et la section de chaque événement pour les schémas spécifiques à l'événement.

Sortie du hook

Votre script dit à Claude Code quoi faire ensuite en écrivant sur stdout ou stderr et en quittant avec un code spécifique. Par exemple, un hook `PreToolUse` qui veut bloquer une commande :

```
#!/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
```

Le code de sortie détermine ce qui se passe ensuite :

- **Exit 0** : l'action se poursuit. Pour les hooks `UserPromptSubmit` et `SessionStart`, tout ce que vous écrivez sur stdout est ajouté au contexte de Claude.
- **Exit 2** : l'action est bloquée. Écrivez une raison sur stderr, et Claude la reçoit comme retour d'information afin qu'il puisse s'ajuster.
- **Tout autre code de sortie** : l'action se poursuit. Stderr est enregistré mais non affiché à Claude. Basculez le mode verbeux avec `Ctrl+O` pour voir ces messages dans la transcription.

Sortie JSON structurée

Les codes de sortie vous donnent deux options : autoriser ou bloquer. Pour plus de contrôle, quittez 0 et imprimez un objet JSON sur stdout à la place.

Note:

Utilisez exit 2 pour bloquer avec un message stderr, ou exit 0 avec JSON pour un contrôle structuré. Ne les mélangez pas : Claude Code ignore JSON lorsque vous quittez 2.

Par exemple, un hook `PreToolUse` peut refuser un appel d'outil et dire à Claude pourquoi, ou l'escalader à l'utilisateur pour approbation :

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

Claude Code lit `permissionDecision` et annule l'appel d'outil, puis renvoie `permissionDecisionReason` à Claude comme retour d'information. Ces trois options sont spécifiques à `PreToolUse` :

- `"allow"` : procéder sans afficher une invite de permission
- `"deny"` : annuler l'appel d'outil et envoyer la raison à Claude
- `"ask"` : afficher l'invite de permission à l'utilisateur comme d'habitude

D'autres événements utilisent des modèles de décision différents. Par exemple, les hooks `PostToolUse` et `Stop` utilisent un champ `decision: "block"` au niveau supérieur, tandis que `PermissionRequest` utilise `hookSpecificOutput.decision.behavior`. Consultez le [tableau récapitulatif](#) dans la référence pour une ventilation complète par événement.

Pour les hooks `UserPromptSubmit`, utilisez `additionalContext` à la place pour injecter du texte dans le contexte de Claude. Les hooks basés sur des invites (`type: "prompt"`) gèrent la sortie différemment : consultez [Hooks basés sur des invites](#).

Filtrer les hooks avec des matchers

Sans matcher, un hook se déclenche à chaque occurrence de son événement. Les matchers vous permettent de réduire cela. Par exemple, si vous voulez exécuter un formateur uniquement après les modifications de fichiers (pas après chaque appel d'outil), ajoutez un matcher à votre hook `PostToolUse` :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

Le matcher `"Edit|Write"` est un modèle regex qui correspond au nom de l'outil. Le hook ne se déclenche que lorsque Claude utilise l'outil `Edit` ou `Write`, pas lorsqu'il utilise `Bash`, `Read`, ou tout autre outil.

Chaque type d'événement correspond à un champ spécifique. Les matchers supportent les chaînes exactes et les modèles regex :

Événement	Ce que le matcher filtre	Exemples de valeurs de matcher
<code>PreToolUse</code> , <code>PostToolUse</code> , <code>PostToolUseFailure</code> , <code>PermissionRequest</code>	nom de l'outil	<code>Bash</code> , <code>Edit Write</code> , <code>mcp_.*</code>
<code>SessionStart</code>	comment la session a démarré	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
<code>SessionEnd</code>	pourquoi la session s'est terminée	<code>clear</code> , <code>logout</code> , <code>prompt_input_exit</code> , <code>bypass_permissions_disabled</code> , <code>other</code>
<code>Notification</code>	type de notification	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>

Événement	Ce que le matcher filtre	Exemples de valeurs de matcher
<code>SubagentStart</code>	type d'agent	<code>Bash</code> , <code>Explore</code> , <code>Plan</code> , ou noms d'agents personnalisés
<code>PreCompact</code>	ce qui a déclenché la compaction	<code>manual</code> , <code>auto</code>
<code>SubagentStop</code>	type d'agent	mêmes valeurs que <code>SubagentStart</code>
<code>ConfigChange</code>	source de configuration	<code>user_settings</code> , <code>project_settings</code> , <code>local_settings</code> , <code>policy_settings</code> , <code>skills</code>
<code>UserPromptSubmit</code> , <code>Stop</code> , <code>TeammateIdle</code> , <code>TaskCompleted</code> , <code>WorktreeCreate</code> , <code>WorktreeRemove</code>	pas de support de matcher	se déclenche toujours à chaque occurrence

Quelques autres exemples montrant des matchers sur différents types d'événements :

Enregistrer chaque commande Bash

Correspond uniquement aux appels d'outil `Bash` et enregistre chaque commande dans un fichier. L'événement `PostToolUse` se déclenche après la fin de la commande, donc `tool_input.command` contient ce qui a été exécuté. Le hook reçoit les données d'événement en JSON sur stdin, et `jq -r '.tool_input.command'` extrait juste la chaîne de commande, que `>>` ajoute au fichier journal :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

Correspondre aux outils MCP

Les outils MCP utilisent une convention de nommage différente des outils intégrés :

`mcp_<server>__<tool>`, où `<server>` est le nom du serveur MCP et `<tool>` est l'outil qu'il fournit. Par exemple, `mcp_github__search_repositories` ou

`mcp_filesystem__read_file`. Utilisez un matcher regex pour cibler tous les outils d'un serveur spécifique, ou correspondre entre les serveurs avec un modèle comme

`mcp_.*__write.*`. Consultez [Correspondre aux outils MCP](#) dans la référence pour la liste complète des exemples.

La commande ci-dessous extrait le nom de l'outil de l'entrée JSON du hook avec `jq` et l'écrit sur stderr, où il apparaît en mode verbeux (`Ctrl+O`) :

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp_github_.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

Nettoyer à la fin de la session

L'événement `SessionEnd` supporte les matchers sur la raison de la fin de la session. Ce hook ne se déclenche que sur `clear` (lorsque vous exécutez `/clear`), pas sur les sorties normales :

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

Pour la syntaxe complète du matcher, consultez la [référence des Hooks](#).

Configurer l'emplacement du hook

L'endroit où vous ajoutez un hook détermine son périmètre :

Emplacement	Périmètre	Partageable
<code>~/.claude/settings.json</code>	Tous vos projets	Non, local à votre machine
<code>.claude/settings.json</code>	Projet unique	Oui, peut être commité dans le repo
<code>.claude/settings.local.json</code>	Projet unique	Non, gitignored
Paramètres de politique gérés	À l'échelle de l'organisation	Oui, contrôlé par l'administrateur
<code>Plugin hooks/hooks.json</code>	Lorsque le plugin est activé	Oui, fourni avec le plugin
<code>Skill</code> ou <code>agent</code> frontmatter	Pendant que le skill ou l'agent est actif	Oui, défini dans le fichier du composant

Vous pouvez également utiliser le menu `/hooks` dans Claude Code pour ajouter, supprimer et afficher les hooks de manière interactive. Pour désactiver tous les hooks à la fois, utilisez le bouton bascule en bas du menu `/hooks` ou définissez `"disableAllHooks": true` dans votre fichier de paramètres.

Les hooks ajoutés via le menu `/hooks` prennent effet immédiatement. Si vous modifiez les fichiers de paramètres directement pendant que Claude Code s'exécute, les modifications ne prendront effet que lorsque vous les examinerez dans le menu `/hooks` ou que vous redémarrerez votre session.

Hooks basés sur des invites

Pour les décisions qui nécessitent un jugement plutôt que des règles déterministes, utilisez les hooks `type: "prompt"`. Au lieu d'exécuter une commande shell, Claude Code envoie votre invite et les données d'entrée du hook à un modèle Claude (Haiku par défaut) pour prendre la décision. Vous pouvez spécifier un modèle différent avec le champ `model` si vous avez besoin de plus de capacité.

Le seul travail du modèle est de retourner une décision oui/non en JSON :

- `"ok": true` : l'action se poursuit
- `"ok": false` : l'action est bloquée. La `"reason"` du modèle est renvoyée à Claude afin qu'il puisse s'ajuster.

Cet exemple utilise un hook `Stop` pour demander au modèle si toutes les tâches demandées sont complètes. Si le modèle retourne `"ok": false`, Claude continue à travailler et utilise la `reason` comme sa prochaine instruction :

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with\n{\n\"ok\": false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

Pour les options de configuration complètes, consultez [Hooks basés sur des invites](#) dans la référence.

Hooks basés sur des agents

Lorsque la vérification nécessite d'inspecter des fichiers ou d'exécuter des commandes, utilisez les hooks `type: "agent"`. Contrairement aux hooks d'invite qui font un seul appel LLM, les hooks d'agent génèrent un subagent qui peut lire des fichiers, rechercher du code et utiliser d'autres outils pour vérifier les conditions avant de retourner une décision.

Les hooks d'agent utilisent le même format de réponse `"ok"` / `"reason"` que les hooks d'invite, mais avec un délai d'expiration par défaut plus long de 60 secondes et jusqu'à 50 tours d'utilisation d'outils.

Cet exemple vérifie que les tests réussissent avant de permettre à Claude de s'arrêter :

```

{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and
check the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}

```

Utilisez les hooks d'invite lorsque les données d'entrée du hook seules suffisent pour prendre une décision. Utilisez les hooks d'agent lorsque vous avez besoin de vérifier quelque chose par rapport à l'état réel de la base de code.

Pour les options de configuration complètes, consultez [Hooks basés sur des agents](#) dans la référence.

HTTP hooks

Utilisez les hooks `type: "http"` pour POST les données d'événement vers un point de terminaison HTTP au lieu d'exécuter une commande shell. Le point de terminaison reçoit le même JSON qu'un hook de commande recevrait sur stdin, et retourne les résultats via le corps de la réponse HTTP en utilisant le même format JSON.

Les HTTP hooks sont utiles lorsque vous voulez qu'un serveur web, une fonction cloud ou un service externe gère la logique du hook : par exemple, un service d'audit partagé qui enregistre les événements d'utilisation d'outils dans une équipe.

Cet exemple poste chaque utilisation d'outil vers un service de journalisation local :

```

{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}

```

Le point de terminaison doit retourner un corps de réponse JSON en utilisant le même [format de sortie](#) que les hooks de commande. Pour bloquer un appel d'outil, retournez une réponse 2xx avec les champs `hookSpecificOutput` appropriés. Les codes de statut HTTP seuls ne peuvent pas bloquer les actions.

Les valeurs d'en-tête supportent l'interpolation de variables d'environnement en utilisant la syntaxe `$VAR_NAME` ou `${VAR_NAME}`. Seules les variables listées dans le tableau `allowedEnvVars` sont résolues ; toutes les autres références `$VAR` restent vides.

Note:

Les HTTP hooks doivent être configurés en modifiant directement votre JSON de paramètres. Le menu interactif `/hooks` ne supporte que l'ajout de hooks de commande.

Pour les options de configuration complètes et la gestion des réponses, consultez [HTTP hooks](#) dans la référence.

Limitations et dépannage

Limitations

- Les hooks de commande communiquent uniquement via stdout, stderr et les codes de sortie. Ils ne peuvent pas déclencher directement des commandes ou des appels d'outils. Les HTTP hooks communiquent via le corps de la réponse à la place.
- Le délai d'expiration du hook est de 10 minutes par défaut, configurable par hook avec le champ `timeout` (en secondes).
- Les hooks `PostToolUse` ne peuvent pas annuler les actions puisque l'outil a déjà été exécuté.
- Les hooks `PermissionRequest` ne se déclenchent pas en [mode non-interactif](#) (`-p`). Utilisez les hooks `PreToolUse` pour les décisions de permission automatisées.
- Les hooks `Stop` se déclenchent chaque fois que Claude termine sa réponse, pas seulement à la fin de la tâche. Ils ne se déclenchent pas sur les interruptions de l'utilisateur.

Hook ne se déclenche pas

Le hook est configuré mais ne s'exécute jamais.

- Exécutez `/hooks` et confirmez que le hook apparaît sous l'événement correct
- Vérifiez que le modèle de matcher correspond exactement au nom de l'outil (les matchers sont sensibles à la casse)
- Vérifiez que vous déclenchez le bon type d'événement (par exemple, `PreToolUse` se déclenche avant l'exécution de l'outil, `PostToolUse` se déclenche après)
- Si vous utilisez des hooks `PermissionRequest` en mode non-interactif (`-p`), passez à `PreToolUse` à la place

Erreur du hook dans la sortie

Vous voyez un message comme « `PreToolUse hook error : ...` » dans la transcription.

- Votre script a quitté avec un code non-zéro de manière inattendue. Testez-le manuellement en piping du JSON d'exemple :

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $? # Check the exit code
```

- Si vous voyez « `command not found` », utilisez des chemins absolus ou `$CLAUDE_PROJECT_DIR` pour référencer les scripts

- Si vous voyez « jq: command not found », installez `jq` ou utilisez Python/Node.js pour l'analyse JSON
- Si le script ne s'exécute pas du tout, rendez-le exécutable : `chmod +x ./my-hook.sh`

`/hooks` n'affiche aucun hook configuré

Vous avez modifié un fichier de paramètres mais les hooks n'apparaissent pas dans le menu.

- Redémarrez votre session ou ouvrez `/hooks` pour recharger. Les hooks ajoutés via le menu `/hooks` prennent effet immédiatement, mais les modifications manuelles des fichiers nécessitent un rechargement.
- Vérifiez que votre JSON est valide (les virgules finales et les commentaires ne sont pas autorisés)
- Confirmez que le fichier de paramètres est au bon emplacement : `.claude/settings.json` pour les hooks de projet, `~/.claude/settings.json` pour les hooks globaux

Le hook Stop s'exécute indéfiniment

Claude continue à travailler dans une boucle infinie au lieu de s'arrêter.

Votre script de hook Stop doit vérifier s'il a déjà déclenché une continuation. Analysez le champ `stop_hook_active` de l'entrée JSON et quittez tôt s'il est `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
```

Validation JSON échouée

Claude Code affiche une erreur d'analyse JSON même si votre script de hook produit du JSON valide.

Lorsque Claude Code exécute un hook, il génère un shell qui source votre profil (`~/.zshrc` ou `~/.bashrc`). Si votre profil contient des instructions `echo` inconditionnelles, cette sortie est ajoutée au début de votre JSON du hook :

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

Claude Code essaie d'analyser ceci en JSON et échoue. Pour corriger cela, enveloppez les instructions echo dans votre profil shell afin qu'elles ne s'exécutent que dans les shells interactifs :

```
## In ~/.zshrc or ~/.bashrc
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

La variable `$-` contient les drapeaux du shell, et `i` signifie interactif. Les hooks s'exécutent dans des shells non-interactifs, donc l'echo est ignoré.

Techniques de débogage

Basculez le mode verbeux avec `Ctrl+O` pour voir la sortie du hook dans la transcription, ou exécutez `claude --debug` pour les détails d'exécution complets, y compris les hooks qui ont correspondu et leurs codes de sortie.

En savoir plus

- [Référence des Hooks](#) : schémas d'événements complets, format de sortie JSON, hooks asynchrones et hooks d'outils MCP
- [Considérations de sécurité](#) : examinez avant de déployer les hooks dans des environnements partagés ou de production
- [Exemple de validateur de commande Bash](#) : implémentation de référence complète

Connecter Claude Code aux outils via MCP

Découvrez comment connecter Claude Code à vos outils avec le Model Context Protocol.

Claude Code peut se connecter à des centaines d'outils externes et de sources de données via le [Model Context Protocol \(MCP\)](#), une norme open source pour les intégrations IA-outils. Les serveurs MCP donnent à Claude Code accès à vos outils, bases de données et API.

Ce que vous pouvez faire avec MCP

Avec les serveurs MCP connectés, vous pouvez demander à Claude Code de :

- **Implémenter des fonctionnalités à partir de suivi de problèmes** : « Ajouter la fonctionnalité décrite dans le problème JIRA ENG-4521 et créer une PR sur GitHub. »
- **Analyser les données de surveillance** : « Vérifier Sentry et Statsig pour vérifier l'utilisation de la fonctionnalité décrite dans ENG-4521. »
- **Interroger les bases de données** : « Trouver les e-mails de 10 utilisateurs aléatoires qui ont utilisé la fonctionnalité ENG-4521, en fonction de notre base de données PostgreSQL. »
- **Intégrer les conceptions** : « Mettre à jour notre modèle d'e-mail standard en fonction des nouvelles conceptions Figma qui ont été publiées sur Slack »
- **Automatiser les flux de travail** : « Créer des brouillons Gmail invitant ces 10 utilisateurs à une session de rétroaction sur la nouvelle fonctionnalité. »

Serveurs MCP populaires

Voici quelques serveurs MCP couramment utilisés que vous pouvez connecter à Claude Code :

Warning:

Utilisez les serveurs MCP tiers à vos propres risques - Anthropic n'a pas vérifié l'exactitude ou la sécurité de tous ces serveurs. Assurez-vous que vous faites confiance aux serveurs MCP que vous installez. Soyez particulièrement prudent lors de l'utilisation de serveurs MCP qui pourraient récupérer du contenu non approuvé, car ceux-ci peuvent vous exposer à un risque d'injection de prompt.

Server	Description	Command
Notion	Connect your Notion workspace to search, update, and power workflows across tools	<code>claude mcp add --transport http notion https://mcp.notion.com/mcp</code>
Canva	Search, create, autofill, and export Canva designs	<code>claude mcp add --transport http canva https://mcp.canva.com/mcp</code>
Figma	Generate diagrams and better code from Figma context	<code>claude mcp add --transport http figma-remote-mcp https://mcp.figma.com/mcp</code>
Atlassian	Access Jira & Confluence from Claude	<code>claude mcp add --transport http atlassian https://mcp.atlassian.com/v1/mcp</code>
Linear	Manage issues, projects & team workflows in Linear	<code>claude mcp add --transport http linear https://mcp.linear.app/mcp</code>
monday.com	Manage projects, boards, and workflows in monday.com	<code>claude mcp add --transport http monday https://mcp.monday.com/mcp</code>
Intercom	Access to Intercom data for better customer insights	<code>claude mcp add --transport http intercom https://mcp.intercom.com/mcp</code>
Vercel	Analyze, debug, and manage projects and deployments	<code>claude mcp add --transport http vercel https://mcp.vercel.com</code>

Server	Description	Command
Granola	The AI notepad for meetings	<code>claude mcp add --transport http granola https://mcp.granola.ai/mcp</code>
Asana	Connect to Asana to coordinate tasks, projects, and goals	<code>claude mcp add --transport streamable-http asana https://mcp.asana.com/v2/mcp</code>
Miro	Access and create new content on Miro boards	<code>claude mcp add --transport http miro https://mcp.miro.com/</code>
Sentry	Search, query, and debug errors intelligently	<code>claude mcp add --transport http sentry https://mcp.sentry.dev/mcp</code>
Supabase	Manage databases, authentication, and storage	<code>claude mcp add --transport http supabase https://mcp.supabase.com/mcp</code>
Hugging Face	Access the Hugging Face Hub and thousands of Gradio Apps	<code>claude mcp add --transport http hugging-face https://huggingface.co/mcp</code>
Context7	Up-to-date docs for LLMs and AI code editors	<code>claude mcp add --transport http context7 https://mcp.context7.com/mcp</code>
Stripe	Payment processing and financial infrastructure tools	<code>claude mcp add --transport http stripe https://mcp.stripe.com</code>

Server	Description	Command
Microsoft Learn	Search trusted Microsoft docs to power your development	<code>claude mcp add --transport http microsoft-learn https://learn.microsoft.com/api/mcp</code>
Clay	Find prospects. Research accounts. Personalize outreach	<code>claude mcp add --transport http clay https://api.clay.com/v3/mcp</code>
Webflow	Manage Webflow CMS, pages, assets and sites	<code>claude mcp add --transport http webflow https://mcp.webflow.com/mcp</code>
Cloudflare	Build applications with compute, storage, and AI	<code>claude mcp add --transport http cloudflare https://bindings.mcp.cloudflare.com/mcp</code>
Ramp	Search, access, and analyze your Ramp financial data	<code>claude mcp add --transport http ramp https://ramp-mcp-remote.ramp.com/mcp</code>
ZoomInfo	Enrich contacts & accounts with GTM intelligence	<code>claude mcp add --transport http zoominfo https://mcp.zoominfo.com/mcp</code>
Netlify	Create, deploy, manage, and secure websites on Netlify	<code>claude mcp add --transport http netlify https://netlify-mcp.netlify.app/mcp</code>
Make	Run Make scenarios and manage your Make account	<code>claude mcp add --transport http make https://mcp.make.com</code>

Server	Description	Command
GoDaddy	Search domains and check availability	<code>claude mcp add --transport http godaddy https://api.godaddy.com/v1/domains/mcp</code>
Google Cloud BigQuery	BigQuery: Advanced analytical insights for agents	<code>claude mcp add --transport http bigquery https://bigquery.googleapis.com/mcp</code>
PayPal	Access PayPal payments platform	<code>claude mcp add --transport http paypal https://mcp.paypal.com/mcp</code>
PostHog	Query, analyze, and manage your PostHog insights	<code>claude mcp add --transport http posthog https://mcp.posthog.com/mcp</code>
Similarweb	Real time web, mobile app, and market data	<code>claude mcp add --transport http similarweb https://mcp.similarweb.com</code>
Crypto.com	Real time prices, orders, charts, and more for crypto	<code>claude mcp add --transport http crypto.com https://mcp.crypto.com/market-data/mcp</code>
Attio	Search, manage, and update your Attio CRM from Claude	<code>claude mcp add --transport http attio https://mcp.attio.com/mcp</code>
Trivago	Find your ideal hotel at the best price	<code>claude mcp add --transport http trivago https://mcp.trivago.com/mcp</code>

Server	Description	Command
Jam	Record screen and collect automatic context for issues	<code>claude mcp add --transport http jam https://mcp.jam.dev/mcp</code>
Consensus	Explore scientific research	<code>claude mcp add --transport http consensus https://mcp.consensus.app/mcp</code>
Clockwise	Advanced scheduling and time management for work	<code>claude mcp add --transport http clockwise https://mcp.getclockwise.com/mcp</code>
Square	Search and manage transaction, merchant, and payment data	<code>claude mcp add --transport sse square https://mcp.squareup.com/sse</code>
Egnyte	Securely access and analyze Egnyte content	<code>claude mcp add --transport http egnyte https://mcp-server.egnyte.com/mcp</code>
Pylon	Search and manage Pylon support issues	<code>claude mcp add --transport http pylon https://mcp.usepylon.com/</code>
Honeycomb	Query and explore observability data and SLOs	<code>claude mcp add --transport http honeycomb https://mcp.honeycomb.io/mcp</code>

Note:

Besoin d’une intégration spécifique ? [Trouvez des centaines d’autres serveurs MCP sur GitHub](#), ou créez le vôtre en utilisant le [MCP SDK](#).

Installation des serveurs MCP

Les serveurs MCP peuvent être configurés de trois façons différentes selon vos besoins :

Option 1 : Ajouter un serveur HTTP distant

Les serveurs HTTP sont l'option recommandée pour se connecter aux serveurs MCP distants. C'est le transport le plus largement supporté pour les services basés sur le cloud.

```
## Syntaxe de base
claude mcp add --transport http <name> <url>

## Exemple réel : Se connecter à Notion
claude mcp add --transport http notion https://mcp.notion.com/mcp

## Exemple avec jeton Bearer
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

Option 2 : Ajouter un serveur SSE distant

Warning:

Le transport SSE (Server-Sent Events) est déprécié. Utilisez plutôt les serveurs HTTP, si disponibles.

```
## Syntaxe de base
claude mcp add --transport sse <name> <url>

## Exemple réel : Se connecter à Asana
claude mcp add --transport sse asana https://mcp.asana.com/sse

## Exemple avec en-tête d'authentification
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

Option 3 : Ajouter un serveur stdio local

Les serveurs Stdio s'exécutent en tant que processus locaux sur votre machine. Ils sont idéaux pour les outils qui ont besoin d'un accès direct au système ou de scripts personnalisés.

```
## Syntaxe de base
claude mcp add [options] <name> -- <command> [args...]

## Exemple réel : Ajouter un serveur Airtable
claude mcp add --transport stdio --env AIRTABLE_API_KEY=YOUR_KEY airtable \
  -- npx -y airtable-mcp-server
```

Note:

Important : Ordre des options

Toutes les options (`--transport` , `--env` , `--scope` , `--header`) doivent venir **avant** le nom du serveur. Le `--` (double tiret) sépare ensuite le nom du serveur de la commande et des arguments qui sont passés au serveur MCP.

Par exemple :

- `claude mcp add --transport stdio myserver -- npx server` → exécute `npx server`
- `claude mcp add --transport stdio --env KEY=value myserver -- python server.py --port 8080` → exécute `python server.py --port 8080` avec `KEY=value` dans l'environnement

Cela évite les conflits entre les drapeaux de Claude et les drapeaux du serveur.

Gestion de vos serveurs

Une fois configurés, vous pouvez gérer vos serveurs MCP avec ces commandes :

```
## Lister tous les serveurs configurés
claude mcp list

## Obtenir les détails d'un serveur spécifique
claude mcp get github

## Supprimer un serveur
claude mcp remove github

## (dans Claude Code) Vérifier l'état du serveur
/mcp
```

Mises à jour dynamiques des outils

Claude Code supporte les notifications MCP `list_changed`, permettant aux serveurs MCP de mettre à jour dynamiquement leurs outils, prompts et ressources disponibles sans vous obliger à vous déconnecter et reconnecter. Lorsqu'un serveur MCP envoie une notification `list_changed`, Claude Code actualise automatiquement les capacités disponibles de ce serveur.

Tip:

Conseils :

- Utilisez le drapeau `--scope` pour spécifier où la configuration est stockée :
- `local` (par défaut) : Disponible uniquement pour vous dans le projet actuel (appelé `project` dans les versions antérieures)
- `project` : Partagé avec tous les membres du projet via le fichier `.mcp.json`
- `user` : Disponible pour vous dans tous les projets (appelé `global` dans les versions antérieures)
- Définissez les variables d'environnement avec les drapeaux `--env` (par exemple, `--env KEY=value`)
- Configurez le délai d'expiration du démarrage du serveur MCP en utilisant la variable d'environnement `MCP_TIMEOUT` (par exemple, `MCP_TIMEOUT=10000 claude` définit un délai d'expiration de 10 secondes)
- Claude Code affichera un avertissement lorsque la sortie de l'outil MCP dépasse 10 000 jetons. Pour augmenter cette limite, définissez la variable d'environnement `MAX_MCP_OUTPUT_TOKENS` (par exemple, `MAX_MCP_OUTPUT_TOKENS=50000`)
- Utilisez `/mcp` pour vous authentifier auprès des serveurs distants qui nécessitent une authentification OAuth 2.0

Warning:

Utilisateurs Windows : Sur Windows natif (pas WSL), les serveurs MCP locaux qui utilisent `npm` nécessitent le wrapper `cmd /c` pour assurer une exécution correcte.

```
## Cela crée command="cmd" que Windows peut exécuter
claude mcp add --transport stdio my-server -- cmd /c npm -y @some/package
```

Sans le wrapper `cmd /c`, vous rencontrerez des erreurs « Connection closed » car Windows ne peut pas exécuter directement `npm`. (Voir la note ci-dessus pour une explication du paramètre `--`.)

Serveurs MCP fournis par les plugins

Les [plugins](#) peuvent regrouper des serveurs MCP, fournissant automatiquement des outils et des intégrations lorsque le plugin est activé. Les serveurs MCP des plugins fonctionnent de manière identique aux serveurs configurés par l'utilisateur.

Comment fonctionnent les serveurs MCP des plugins :

- Les plugins définissent les serveurs MCP dans `.mcp.json` à la racine du plugin ou en ligne dans `plugin.json`
- Lorsqu'un plugin est activé, ses serveurs MCP démarrent automatiquement
- Les outils MCP des plugins apparaissent aux côtés des outils MCP configurés manuellement
- Les serveurs des plugins sont gérés via l'installation du plugin (pas via les commandes `/mcp`)

Exemple de configuration MCP du plugin :

Dans `.mcp.json` à la racine du plugin :

```
{
  "database-tools": {
    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
    "env": {
      "DB_URL": "${DB_URL}"
    }
  }
}
```

Ou en ligne dans `plugin.json` :

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

Fonctionnalités MCP du plugin :

- **Cycle de vie automatique** : Les serveurs démarrent lorsque le plugin s'active, mais vous devez redémarrer Claude Code pour appliquer les modifications du serveur MCP (activation ou désactivation)
- **Variables d'environnement** : Utilisez `${CLAUDE_PLUGIN_ROOT}` pour les chemins relatifs au plugin
- **Accès aux variables d'environnement utilisateur** : Accès aux mêmes variables d'environnement que les serveurs configurés manuellement
- **Types de transport multiples** : Support des transports stdio, SSE et HTTP (le support des transports peut varier selon le serveur)

Affichage des serveurs MCP du plugin :

```
## Dans Claude Code, voir tous les serveurs MCP y compris ceux du plugin
/mcp
```

Les serveurs des plugins apparaissent dans la liste avec des indicateurs montrant qu'ils proviennent des plugins.

Avantages des serveurs MCP du plugin :

- **Distribution groupée** : Outils et serveurs emballés ensemble
- **Configuration automatique** : Aucune configuration MCP manuelle nécessaire
- **Cohérence d'équipe** : Tout le monde obtient les mêmes outils lorsque le plugin est installé

Consultez la [référence des composants du plugin](#) pour plus de détails sur le regroupement des serveurs MCP avec les plugins.

Portées d'installation MCP

Les serveurs MCP peuvent être configurés à trois niveaux de portée différents, chacun servant des objectifs distincts pour gérer l'accessibilité et le partage des serveurs. Comprendre ces portées vous aide à déterminer la meilleure façon de configurer les serveurs pour vos besoins spécifiques.

Portée locale

Les serveurs à portée locale représentent le niveau de configuration par défaut et sont stockés dans `~/.claude.json` sous le chemin de votre projet. Ces serveurs restent privés pour vous et ne sont accessibles que lorsque vous travaillez dans le répertoire du projet actuel. Cette portée est idéale pour les serveurs de développement personnels, les configurations expérimentales ou les serveurs contenant des identifiants sensibles qui ne doivent pas être partagés.

Note:

Le terme « portée locale » pour les serveurs MCP diffère des paramètres locaux généraux. Les serveurs MCP à portée locale sont stockés dans `~/.claude.json` (votre répertoire personnel), tandis que les paramètres locaux généraux utilisent `.claude/settings.local.json` (dans le répertoire du projet). Consultez [Paramètres](#) pour plus de détails sur les emplacements des fichiers de paramètres.

```
## Ajouter un serveur à portée locale (par défaut)
claude mcp add --transport http stripe https://mcp.stripe.com

## Spécifier explicitement la portée locale
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

Portée du projet

Les serveurs à portée de projet permettent la collaboration d'équipe en stockant les configurations dans un fichier `.mcp.json` à la racine de votre projet. Ce fichier est conçu pour être archivé dans le contrôle de version, garantissant que tous les membres de l'équipe ont accès aux mêmes outils et services MCP. Lorsque vous ajoutez un serveur à portée de projet, Claude Code crée ou met à jour automatiquement ce fichier avec la structure de configuration appropriée.

```
## Ajouter un serveur à portée de projet
claude mcp add --transport http paypal --scope project https://mcp.paypal.com/mcp
```

Le fichier `.mcp.json` résultant suit un format standardisé :

```
{
  "mcpServers": {
    "shared-server": {
      "command": "/path/to/server",
      "args": [],
      "env": {}
    }
  }
}
```

Pour des raisons de sécurité, Claude Code demande une approbation avant d'utiliser les serveurs à portée de projet à partir des fichiers `.mcp.json`. Si vous devez réinitialiser ces choix d'approbation, utilisez la commande `claude mcp reset-project-choices`.

Portée utilisateur

Les serveurs à portée utilisateur sont stockés dans `~/.claude.json` et offrent une accessibilité inter-projets, les rendant disponibles dans tous les projets de votre machine tout en restant privés pour votre compte utilisateur. Cette portée fonctionne bien pour les serveurs utilitaires personnels, les outils de développement ou les services que vous utilisez fréquemment dans différents projets.

```
## Ajouter un serveur utilisateur
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/anthropic
```

Choisir la bonne portée

Sélectionnez votre portée en fonction de :

- **Portée locale** : Serveurs personnels, configurations expérimentales ou identifiants sensibles spécifiques à un projet
- **Portée du projet** : Serveurs partagés par l'équipe, outils spécifiques au projet ou services requis pour la collaboration
- **Portée utilisateur** : Utilitaires personnels nécessaires dans plusieurs projets, outils de développement ou services fréquemment utilisés

Note:

Où sont stockés les serveurs MCP ?

- **Portée utilisateur et locale** : `~/.claude.json` (dans le champ `mcpServers` ou sous les chemins du projet)
- **Portée du projet** : `.mcp.json` à la racine de votre projet (archivé dans le contrôle de source)
- **Géré** : `managed-mcp.json` dans les répertoires système (voir [Configuration MCP gérée](#))

Hiérarchie de portée et précedence

Les configurations du serveur MCP suivent une hiérarchie de précedence claire. Lorsque des serveurs portant le même nom existent à plusieurs portées, le système résout les conflits en donnant la priorité aux serveurs à portée locale en premier, suivis des serveurs à portée de projet, et enfin des serveurs à portée utilisateur. Cette conception garantit que les configurations personnelles peuvent remplacer les configurations partagées si nécessaire.

Expansion des variables d'environnement dans `.mcp.json`

Claude Code supporte l'expansion des variables d'environnement dans les fichiers `.mcp.json`, permettant aux équipes de partager des configurations tout en maintenant la flexibilité pour les chemins spécifiques à la machine et les valeurs sensibles comme les clés API.

Syntaxe supportée :

- `${VAR}` - Se développe à la valeur de la variable d'environnement `VAR`
- `${VAR:-default}` - Se développe à `VAR` si défini, sinon utilise `default`

Emplacements d'expansion : Les variables d'environnement peuvent être développées dans :

- `command` - Le chemin de l'exécutable du serveur
- `args` - Arguments de la ligne de commande
- `env` - Variables d'environnement passées au serveur
- `url` - Pour les types de serveur HTTP
- `headers` - Pour l'authentification du serveur HTTP

Exemple avec expansion de variable :

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

Si une variable d'environnement requise n'est pas définie et n'a pas de valeur par défaut, Claude Code ne pourra pas analyser la configuration.

Exemples pratiques

`{/* ### Exemple : Automatiser les tests de navigateur avec Playwright`

```
claude mcp add --transport stdio playwright -- npx -y @playwright/mcp@latest
```

Ensuite, écrivez et exécutez des tests de navigateur :

```
Test si le flux de connexion fonctionne avec test@example.com
```

```
Prendre une capture d'écran de la page de paiement sur mobile
```

```
Vérifier que la fonction de recherche retourne des résultats
``` */}

Exemple : Surveiller les erreurs avec Sentry

```bash theme={null}
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Authentifiez-vous avec votre compte Sentry :

```
/mcp
```

Ensuite, déboguez les problèmes de production :

```
Quelles sont les erreurs les plus courantes au cours des 24 dernières heures ?
```

```
Montrez-moi la trace de pile pour l'erreur ID abc123
```

```
Quel déploiement a introduit ces nouvelles erreurs ?
```

Exemple : Se connecter à GitHub pour les révisions de code

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Authentifiez-vous si nécessaire en sélectionnant « Authenticate » pour GitHub :

```
/mcp
```

Ensuite, travaillez avec GitHub :

```
Examinez la PR #456 et suggérez des améliorations
```

```
Créer un nouveau problème pour le bogue que nous venons de trouver
```

```
Montrez-moi toutes les PR ouvertes qui me sont assignées
```

Exemple : Interroger votre base de données PostgreSQL

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \  
--dsn "postgresql://readonly:pass@prod.db.com:5432/anaLytics"
```

Ensuite, interrogez votre base de données naturellement :

```
Quel est notre revenu total ce mois-ci ?
```

```
Montrez-moi le schéma de la table des commandes
```

```
Trouver les clients qui n'ont pas effectué d'achat depuis 90 jours
```

S'authentifier auprès des serveurs MCP distants

De nombreux serveurs MCP basés sur le cloud nécessitent une authentification. Claude Code supporte OAuth 2.0 pour les connexions sécurisées.

Step 1: Ajouter le serveur qui nécessite une authentification

Par exemple :

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

Step 2: Utiliser la commande `/mcp` dans Claude Code

Dans Claude Code, utilisez la commande :

```
/mcp
```

Ensuite, suivez les étapes dans votre navigateur pour vous connecter.

Tip:

Conseils :

- Les jetons d'authentification sont stockés de manière sécurisée et actualisés automatiquement
- Utilisez « Clear authentication » dans le menu `/mcp` pour révoquer l'accès
- Si votre navigateur ne s'ouvre pas automatiquement, copiez l'URL fournie et ouvrez-la manuellement
- Si la redirection du navigateur échoue avec une erreur de connexion après l'authentification, collez l'URL de rappel complète de la barre d'adresse de votre navigateur dans l'invite d'URL qui apparaît dans Claude Code
- L'authentification OAuth fonctionne avec les serveurs HTTP

Utiliser un port de rappel OAuth fixe

Certains serveurs MCP nécessitent un URI de redirection spécifique enregistré à l'avance. Par défaut, Claude Code choisit un port disponible aléatoire pour le rappel OAuth. Utilisez `--callback-port` pour fixer le port afin qu'il corresponde à un URI de redirection pré-enregistré de la forme `http://localhost:PORT/callback`.

Vous pouvez utiliser `--callback-port` seul (avec l'enregistrement dynamique du client) ou ensemble avec `--client-id` (avec les identifiants pré-configurés).

```
## Port de rappel fixe avec enregistrement dynamique du client
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

Utiliser les identifiants OAuth pré-configurés

Certains serveurs MCP ne supportent pas la configuration OAuth automatique. Si vous voyez une erreur comme « Incompatible auth server: does not support dynamic client registration », le serveur nécessite des identifiants pré-configurés. Enregistrez d'abord une application OAuth via le portail des développeurs du serveur, puis fournissez les identifiants lors de l'ajout du serveur.

Step 1: Enregistrer une application OAuth auprès du serveur

Créez une application via le portail des développeurs du serveur et notez votre ID client et votre secret client.

De nombreux serveurs nécessitent également un URI de redirection. Si c'est le cas, choisissez un port et enregistrez un URI de redirection au format

`http://localhost:PORT/callback`. Utilisez ce même port avec `--callback-port` à l'étape suivante.

Step 2: Ajouter le serveur avec vos identifiants

Choisissez l'une des méthodes suivantes. Le port utilisé pour `--callback-port` peut être n'importe quel port disponible. Il doit simplement correspondre à l'URI de redirection que vous avez enregistré à l'étape précédente.

claude mcp add

Utilisez `--client-id` pour passer l'ID client de votre application. Le drapeau `--client-secret` demande le secret avec une entrée masquée :

```
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

Incluez l'objet `oauth` dans la configuration JSON et passez `--client-secret` comme drapeau séparé :

```
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 (port de rappel uniquement)

Utilisez `--callback-port` sans ID client pour fixer le port tout en utilisant l'enregistrement dynamique du client :

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":
  {"callbackPort":8080}}'
```

CI / variable d'environnement

Définissez le secret via une variable d'environnement pour ignorer l'invite interactive :

```
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: S'authentifier dans Claude Code

Exécutez `/mcp` dans Claude Code et suivez le flux de connexion du navigateur.

Tip:

Conseils :

- Le secret client est stocké de manière sécurisée dans votre trousseau système (macOS) ou un fichier d'identifiants, pas dans votre configuration
- Si le serveur utilise un client OAuth public sans secret, utilisez uniquement `--client-id` sans `--client-secret`
- `--callback-port` peut être utilisé avec ou sans `--client-id`
- Ces drapeaux s'appliquent uniquement aux transports HTTP et SSE. Ils n'ont aucun effet sur les serveurs stdio
- Utilisez `claude mcp get <name>` pour vérifier que les identifiants OAuth sont configurés pour un serveur

Remplacer la découverte des métadonnées OAuth

Si votre serveur MCP retourne des erreurs sur le point de terminaison des métadonnées OAuth standard (`/.well-known/oauth-authorization-server`) mais expose un point de terminaison OIDC fonctionnant, vous pouvez dire à Claude Code de récupérer les métadonnées OAuth directement à partir d'une URL que vous spécifiez, en contournant la chaîne de découverte standard.

Définissez `authServerMetadataUrl` dans l'objet `oauth` de la configuration de votre serveur dans `.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 doit utiliser `https://`. Cette option nécessite Claude Code v2.1.64 ou ultérieur.

Ajouter des serveurs MCP à partir de la configuration JSON

Si vous avez une configuration JSON pour un serveur MCP, vous pouvez l'ajouter directement :

Step 1: Ajouter un serveur MCP à partir de JSON

```
## Syntaxe de base
claude mcp add-json <name> '<json>'

## Exemple : Ajouter un serveur HTTP avec configuration JSON
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/mcp","headers":{"Authorization":"Bearer token"}}'

## Exemple : Ajouter un serveur stdio avec configuration JSON
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

## Exemple : Ajouter un serveur HTTP avec identifiants OAuth pré-configurés
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: Vérifier que le serveur a été ajouté

```
claude mcp get weather-api
```

Tip:

Conseils :

- Assurez-vous que le JSON est correctement échappé dans votre shell
- Le JSON doit se conformer au schéma de configuration du serveur MCP
- Vous pouvez utiliser `--scope user` pour ajouter le serveur à votre configuration utilisateur au lieu de celle spécifique au projet

Importer les serveurs MCP à partir de Claude Desktop

Si vous avez déjà configuré des serveurs MCP dans Claude Desktop, vous pouvez les importer :

Step 1: Importer les serveurs à partir de Claude Desktop

```
## Syntaxe de base
claude mcp add-from-claude-desktop
```


Step 2: Sélectionner les serveurs à importer

Après avoir exécuté la commande, vous verrez une boîte de dialogue interactive qui vous permet de sélectionner les serveurs que vous souhaitez importer.

Step 3: Vérifier que les serveurs ont été importés

```
claude mcp list
```

Tip:

Conseils :

- Cette fonctionnalité ne fonctionne que sur macOS et Windows Subsystem for Linux (WSL)
- Elle lit le fichier de configuration de Claude Desktop à partir de son emplacement standard sur ces plates-formes
- Utilisez le drapeau `--scope user` pour ajouter les serveurs à votre configuration utilisateur
- Les serveurs importés auront les mêmes noms que dans Claude Desktop
- Si des serveurs portant les mêmes noms existent déjà, ils recevront un suffixe numérique (par exemple, `server_1`)

Utiliser les serveurs MCP à partir de Claude.ai

Si vous vous êtes connecté à Claude Code avec un compte [Claude.ai](https://claude.ai), les serveurs MCP que vous avez ajoutés dans Claude.ai sont automatiquement disponibles dans Claude Code :

Step 1: Configurer les serveurs MCP dans Claude.ai

Ajoutez les serveurs à claude.ai/settings/connectors. Sur les plans Team et Enterprise, seuls les administrateurs peuvent ajouter des serveurs.

Step 2: Authentifier le serveur MCP

Complétez les étapes d'authentification requises dans Claude.ai.

Step 3: Afficher et gérer les serveurs dans Claude Code

Dans Claude Code, utilisez la commande :

```
/mcp
```

Les serveurs Claude.ai apparaissent dans la liste avec des indicateurs montrant qu'ils proviennent de Claude.ai.

Pour désactiver les serveurs MCP de Claude.ai dans Claude Code, définissez la variable d'environnement `ENABLE_CLAUDEAI_MCP_SERVERS` sur `false` :

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

Utiliser Claude Code comme serveur MCP

Vous pouvez utiliser Claude Code lui-même comme serveur MCP auquel d'autres applications peuvent se connecter :

```
## Démarrer Claude en tant que serveur MCP studio
claude mcp serve
```

Vous pouvez l'utiliser dans Claude Desktop en ajoutant cette configuration à `claude_desktop_config.json` :

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Warning:

Configuration du chemin de l'exécutable : Le champ `command` doit référencer l'exécutable Claude Code. Si la commande `claude` n'est pas dans le PATH de votre système, vous devrez spécifier le chemin complet de l'exécutable.

Pour trouver le chemin complet :

```
which claude
```

Ensuite, utilisez le chemin complet dans votre configuration :

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

Sans le chemin d'exécutable correct, vous rencontrerez des erreurs comme `spawn claude ENOENT`.

Tip:

Conseils :

- Le serveur fournit l'accès aux outils de Claude comme View, Edit, LS, etc.
- Dans Claude Desktop, essayez de demander à Claude de lire les fichiers dans un répertoire, de faire des modifications, et plus encore.
- Notez que ce serveur MCP expose uniquement les outils de Claude Code à votre client MCP, donc votre propre client est responsable de l'implémentation de la confirmation de l'utilisateur pour les appels d'outils individuels.

Limites de sortie MCP et avertissements

Lorsque les outils MCP produisent de grandes sorties, Claude Code aide à gérer l'utilisation des jetons pour éviter de surcharger votre contexte de conversation :

- **Seuil d'avertissement de sortie** : Claude Code affiche un avertissement lorsque la sortie de tout outil MCP dépasse 10 000 jetons
- **Limite configurable** : Vous pouvez ajuster le nombre maximum de jetons de sortie MCP autorisés en utilisant la variable d'environnement `MAX_MCP_OUTPUT_TOKENS`
- **Limite par défaut** : La limite maximale par défaut est de 25 000 jetons

Pour augmenter la limite pour les outils qui produisent de grandes sorties :

```
## Définir une limite plus élevée pour les sorties des outils MCP
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

Ceci est particulièrement utile lorsque vous travaillez avec des serveurs MCP qui :

- Interrogent de grands ensembles de données ou des bases de données
- Génèrent des rapports ou des documentations détaillés
- Traitent des fichiers journaux ou des informations de débogage étendus

Warning:

Si vous rencontrez fréquemment des avertissements de sortie avec des serveurs MCP spécifiques, envisagez d'augmenter la limite ou de configurer le serveur pour paginer ou filtrer ses réponses.

Utiliser les ressources MCP

Les serveurs MCP peuvent exposer des ressources que vous pouvez référencer en utilisant des mentions @, similaire à la façon dont vous référencez les fichiers.

Référencer les ressources MCP

Step 1: Lister les ressources disponibles

Tapez @ dans votre prompt pour voir les ressources disponibles de tous les serveurs MCP connectés. Les ressources apparaissent aux côtés des fichiers dans le menu d'autocomplétion.

Step 2: Référencer une ressource spécifique

Utilisez le format @server:protocol://resource/path pour référencer une ressource :

Pouvez-vous analyser @github:issue://123 et suggérer un correctif ?

Veuillez examiner la documentation API à @docs:file://api/authentication

Step 3: Références de ressources multiples

Vous pouvez référencer plusieurs ressources dans un seul prompt :

Comparez `@postgres:schema://users` avec `@docs:file://database/user-model`

Tip:

Conseils :

- Les ressources sont automatiquement récupérées et incluses en tant que pièces jointes lorsqu'elles sont référencées
- Les chemins des ressources sont recherchables par correspondance floue dans l'autocomplétion de mention @
- Claude Code fournit automatiquement des outils pour lister et lire les ressources MCP lorsque les serveurs les supportent
- Les ressources peuvent contenir n'importe quel type de contenu fourni par le serveur MCP (texte, JSON, données structurées, etc.)

Mettre à l'échelle avec la recherche d'outils MCP

Lorsque vous avez de nombreux serveurs MCP configurés, les définitions d'outils peuvent consommer une partie importante de votre fenêtre de contexte. La recherche d'outils MCP résout ce problème en chargeant les outils à la demande au lieu de les précharger tous.

Comment cela fonctionne

Claude Code active automatiquement la recherche d'outils lorsque vos descriptions d'outils MCP consommeraient plus de 10 % de la fenêtre de contexte. Vous pouvez [ajuster ce seuil](#) ou désactiver complètement la recherche d'outils. Lorsqu'elle est déclenchée :

1. Les outils MCP sont différés plutôt que chargés dans le contexte à l'avance
2. Claude utilise un outil de recherche pour découvrir les outils MCP pertinents si nécessaire
3. Seuls les outils dont Claude a réellement besoin sont chargés dans le contexte
4. Les outils MCP continuent à fonctionner exactement comme avant de votre point de vue

Pour les auteurs de serveurs MCP

Si vous créez un serveur MCP, le champ des instructions du serveur devient plus utile avec la recherche d'outils activée. Les instructions du serveur aident Claude à comprendre quand rechercher vos outils, similaire à la façon dont les [skills](#) fonctionnent.

Ajoutez des instructions de serveur claires et descriptives qui expliquent :

- Quelle catégorie de tâches vos outils gèrent
- Quand Claude doit rechercher vos outils
- Les capacités clés de votre serveur

Configurer la recherche d'outils

La recherche d'outils s'exécute en mode auto par défaut, ce qui signifie qu'elle s'active uniquement lorsque vos définitions d'outils MCP dépassent le seuil de contexte. Si vous avez peu d'outils, ils se chargent normalement sans recherche d'outils. Cette fonctionnalité nécessite des modèles qui supportent les blocs `tool_reference` : Sonnet 4 et ultérieur, ou Opus 4 et ultérieur. Les modèles Haiku ne supportent pas la recherche d'outils.

Contrôlez le comportement de la recherche d'outils avec la variable d'environnement

`ENABLE_TOOL_SEARCH` :

Valeur	Comportement
<code>auto</code>	S'active lorsque les outils MCP dépassent 10 % du contexte (par défaut)
<code>auto:<N></code>	S'active au seuil personnalisé, où <code><N></code> est un pourcentage (par exemple, <code>auto:5</code> pour 5 %)
<code>true</code>	Toujours activé
<code>false</code>	Désactivé, tous les outils MCP chargés à l'avance

```
## Utiliser un seuil personnalisé de 5 %
ENABLE_TOOL_SEARCH=auto:5 claude

## Désactiver complètement la recherche d'outils
ENABLE_TOOL_SEARCH=false claude
```

Ou définissez la valeur dans le champ `env` de votre [settings.json](#).

Vous pouvez également désactiver l'outil MCPSearch spécifiquement en utilisant le paramètre `disallowedTools` :

```
{
  "permissions": {
    "deny": ["MCPSearch"]
  }
}
```

Utiliser les prompts MCP comme commandes

Les serveurs MCP peuvent exposer des prompts qui deviennent disponibles en tant que commandes dans Claude Code.

Exécuter les prompts MCP

Step 1: Découvrir les prompts disponibles

Tapez `/` pour voir toutes les commandes disponibles, y compris celles des serveurs MCP. Les prompts MCP apparaissent au format `/mcp__servername__promptname`.

Step 2: Exécuter un prompt sans arguments

```
/mcp__github__list_prs
```

Step 3: Exécuter un prompt avec des arguments

De nombreux prompts acceptent des arguments. Passez-les séparés par des espaces après la commande :

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue "Bug in login flow" high
```

Tip:

Conseils :

- Les prompts MCP sont découverts dynamiquement à partir des serveurs connectés
- Les arguments sont analysés en fonction des paramètres définis du prompt
- Les résultats du prompt sont injectés directement dans la conversation

- Les noms de serveur et de prompt sont normalisés (les espaces deviennent des traits de soulignement)

Configuration MCP gérée

Pour les organisations qui ont besoin d'un contrôle centralisé sur les serveurs MCP, Claude Code supporte deux options de configuration :

1. **Contrôle exclusif avec `managed-mcp.json`** : Déployer un ensemble fixe de serveurs MCP que les utilisateurs ne peuvent pas modifier ou étendre
2. **Contrôle basé sur les politiques avec listes blanches/noires** : Permettre aux utilisateurs d'ajouter leurs propres serveurs, mais restreindre lesquels sont autorisés

Ces options permettent aux administrateurs informatiques de :

- **Contrôler les serveurs MCP auxquels les employés peuvent accéder** : Déployer un ensemble standardisé de serveurs MCP approuvés dans toute l'organisation
- **Empêcher les serveurs MCP non autorisés** : Restreindre les utilisateurs d'ajouter des serveurs MCP non approuvés
- **Désactiver complètement MCP** : Supprimer complètement la fonctionnalité MCP si nécessaire

Option 1 : Contrôle exclusif avec `managed-mcp.json`

Lorsque vous déployez un fichier `managed-mcp.json`, il prend le **contrôle exclusif** de tous les serveurs MCP. Les utilisateurs ne peuvent pas ajouter, modifier ou utiliser d'autres serveurs MCP que ceux définis dans ce fichier. C'est l'approche la plus simple pour les organisations qui veulent un contrôle complet.

Les administrateurs système déploient le fichier de configuration dans un répertoire à l'échelle du système :

- macOS : `/Library/Application Support/ClaudeCode/managed-mcp.json`
- Linux et WSL : `/etc/claude-code/managed-mcp.json`
- Windows : `C:\Program Files\ClaudeCode\managed-mcp.json`

Note:

Ce sont des chemins à l'échelle du système (pas des répertoires personnels comme `~/Library/...`) qui nécessitent des privilèges d'administrateur. Ils sont conçus pour être déployés par les administrateurs informatiques.

Le fichier `managed-mcp.json` utilise le même format qu'un fichier `.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"
      }
    }
  }
}
```

Option 2 : Contrôle basé sur les politiques avec listes blanches et noires

Au lieu de prendre le contrôle exclusif, les administrateurs peuvent permettre aux utilisateurs de configurer leurs propres serveurs MCP tout en appliquant des restrictions sur les serveurs autorisés. Cette approche utilise `allowedMcpServers` et `deniedMcpServers` dans le [fichier de paramètres gérés](#).

Note:

Choisir entre les options : Utilisez l'option 1 (`managed-mcp.json`) lorsque vous souhaitez déployer un ensemble fixe de serveurs sans personnalisation utilisateur. Utilisez l'option 2 (listes blanches/noires) lorsque vous souhaitez permettre aux utilisateurs d'ajouter leurs propres serveurs dans le respect des contraintes de politique.

Options de restriction

Chaque entrée dans la liste blanche ou noire peut restreindre les serveurs de trois façons :

1. **Par nom de serveur** (`serverName`) : Correspond au nom configuré du serveur
2. **Par commande** (`serverCommand`) : Correspond à la commande exacte et aux arguments utilisés pour démarrer les serveurs stdio
3. **Par modèle d'URL** (`serverUrl`) : Correspond aux URL des serveurs distants avec support des caractères génériques

Important : Chaque entrée doit avoir exactement un de `serverName` , `serverCommand` ou `serverUrl` .

Exemple de configuration

```
{
  "allowedMcpServers": [
    // Autoriser par nom de serveur
    { "serverName": "github" },
    { "serverName": "sentry" },

    // Autoriser par commande exacte (pour les serveurs stdio)
    { "serverCommand": ["npx", "-y", "@modelcontextprotocol/server-filesystem"] },
    { "serverCommand": ["python", "/usr/local/bin/approved-server.py"] },

    // Autoriser par modèle d'URL (pour les serveurs distants)
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ],
  "deniedMcpServers": [
    // Bloquer par nom de serveur
    { "serverName": "dangerous-server" },

    // Bloquer par commande exacte (pour les serveurs stdio)
    { "serverCommand": ["npx", "-y", "unapproved-package"] },

    // Bloquer par modèle d'URL (pour les serveurs distants)
    { "serverUrl": "https://*.untrusted.com/*" }
  ]
}
```

Comment fonctionnent les restrictions basées sur les commandes

Correspondance exacte :

- Les tableaux de commandes doivent correspondre **exactement** - à la fois la commande et tous les arguments dans le bon ordre
- Exemple : `["npx", "-y", "server"]` ne correspondra PAS à `["npx", "server"]` ou `["npx", "-y", "server", "--flag"]`

Comportement du serveur stdio :

- Lorsque la liste blanche contient **n'importe quelle** entrée `serverCommand`, les serveurs stdio **doivent** correspondre à l'une de ces commandes

- Les serveurs stdio ne peuvent pas passer par le nom seul lorsque des restrictions de commande sont présentes
- Cela garantit que les administrateurs peuvent appliquer les commandes autorisées à s'exécuter

Comportement du serveur non-stdio :

- Les serveurs distants (HTTP, SSE, WebSocket) utilisent la correspondance basée sur l'URL lorsque des entrées `serverUrl` existent dans la liste blanche
- Si aucune entrée d'URL n'existe, les serveurs distants reviennent à la correspondance basée sur le nom
- Les restrictions de commande ne s'appliquent pas aux serveurs distants

Comment fonctionnent les restrictions basées sur l'URL

Les modèles d'URL supportent les caractères génériques en utilisant `*` pour correspondre à n'importe quelle séquence de caractères. Ceci est utile pour autoriser des domaines ou des sous-domaines entiers.

Exemples de caractères génériques :

- `https://mcp.company.com/*` - Autoriser tous les chemins sur un domaine spécifique
- `https://*.example.com/*` - Autoriser n'importe quel sous-domaine de example.com
- `http://localhost:*/*` - Autoriser n'importe quel port sur localhost

Comportement du serveur distant :

- Lorsque la liste blanche contient **n'importe quelle** entrée `serverUrl`, les serveurs distants **doivent** correspondre à l'un de ces modèles d'URL
- Les serveurs distants ne peuvent pas passer par le nom seul lorsque des restrictions d'URL sont présentes
- Cela garantit que les administrateurs peuvent appliquer les points de terminaison distants autorisés

```
{
  "allowedMcpServers": [
    { "serverUrl": "https://mcp.company.com/*" },
    { "serverUrl": "https://*.internal.corp/*" }
  ]
}
```

Résultat :

- Serveur HTTP à `https://mcp.company.com/api` :  Autorisé (correspond au modèle d'URL)
- Serveur HTTP à `https://api.internal.corp/mcp` :  Autorisé (correspond au sous-domaine générique)
- Serveur HTTP à `https://external.com/mcp` :  Bloqué (ne correspond à aucun modèle d'URL)
- Serveur stdio avec n'importe quelle commande :  Bloqué (aucune entrée de nom ou de commande à correspondre)

```
{
  "allowedMcpServers": [
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Résultat :

- Serveur stdio avec `["npx", "-y", "approved-package"]` :  Autorisé (correspond à la commande)
- Serveur stdio avec `["node", "server.js"]` :  Bloqué (ne correspond pas à la commande)
- Serveur HTTP nommé « my-api » :  Bloqué (aucune entrée de nom à correspondre)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverCommand": ["npx", "-y", "approved-package"] }
  ]
}
```

Résultat :

- Serveur stdio nommé « local-tool » avec `["npx", "-y", "approved-package"]` :  Autorisé (correspond à la commande)
- Serveur stdio nommé « local-tool » avec `["node", "server.js"]` :  Bloqué (les entrées de commande existent mais ne correspondent pas)

- Serveur stdio nommé « github » avec `["node", "server.js"]` : ❌ Bloqué (les serveurs stdio doivent correspondre aux commandes lorsque les entrées de commande existent)
- Serveur HTTP nommé « github » : ✅ Autorisé (correspond au nom)
- Serveur HTTP nommé « other-api » : ❌ Bloqué (le nom ne correspond pas)

```
{
  "allowedMcpServers": [
    { "serverName": "github" },
    { "serverName": "internal-tool" }
  ]
}
```

Résultat :

- Serveur stdio nommé « github » avec n'importe quelle commande : ✅ Autorisé (aucune restriction de commande)
- Serveur stdio nommé « internal-tool » avec n'importe quelle commande : ✅ Autorisé (aucune restriction de commande)
- Serveur HTTP nommé « github » : ✅ Autorisé (correspond au nom)
- N'importe quel serveur nommé « other » : ❌ Bloqué (le nom ne correspond pas)

Comportement de la liste blanche (`allowedMcpServers`)

- `undefined` (par défaut) : Aucune restriction - les utilisateurs peuvent configurer n'importe quel serveur MCP
- Tableau vide `[]` : Verrouillage complet - les utilisateurs ne peuvent configurer aucun serveur MCP
- Liste d'entrées : Les utilisateurs ne peuvent configurer que les serveurs qui correspondent par nom, commande ou modèle d'URL

Comportement de la liste noire (`deniedMcpServers`)

- `undefined` (par défaut) : Aucun serveur n'est bloqué
- Tableau vide `[]` : Aucun serveur n'est bloqué
- Liste d'entrées : Les serveurs spécifiés sont explicitement bloqués dans toutes les portées

Notes importantes

- **L'option 1 et l'option 2 peuvent être combinées** : Si `managed-mcp.json` existe, il a le contrôle exclusif et les utilisateurs ne peuvent pas ajouter de serveurs. Les listes blanches/noires s'appliquent toujours aux serveurs gérés eux-mêmes.
- **La liste noire a une précedence absolue** : Si un serveur correspond à une entrée de liste noire (par nom, commande ou URL), il sera bloqué même s'il est sur la liste blanche
- **Les restrictions basées sur le nom, la commande et l'URL fonctionnent ensemble** : un serveur passe s'il correspond à **soit** une entrée de nom, une entrée de commande, ou un modèle d'URL (sauf s'il est bloqué par la liste noire)

Note:

Lors de l'utilisation de `managed-mcp.json` : Les utilisateurs ne peuvent pas ajouter de serveurs MCP via `claude mcp add` ou les fichiers de configuration. Les paramètres `allowedMcpServers` et `deniedMcpServers` s'appliquent toujours pour filtrer les serveurs gérés qui sont réellement chargés.

Étendre Claude avec des skills

Créez, gérez et partagez des skills pour étendre les capacités de Claude dans Claude Code. Inclut les commandes personnalisées et les skills groupées.

Les skills étendent ce que Claude peut faire. Créez un fichier `SKILL.md` avec des instructions, et Claude l'ajoute à sa boîte à outils. Claude utilise les skills quand c'est pertinent, ou vous pouvez en invoquer une directement avec `/skill-name`.

Note:

Pour les commandes intégrées comme `/help` et `/compact`, voir [mode interactif](#).

Les commandes personnalisées ont été fusionnées dans les skills. Un fichier à `.claude/commands/deploy.md` et une skill à `.claude/skills/deploy/SKILL.md` créent tous les deux `/deploy` et fonctionnent de la même manière. Vos fichiers `.claude/commands/` existants continuent de fonctionner. Les skills ajoutent des fonctionnalités optionnelles : un répertoire pour les fichiers de support, un frontmatter pour [contrôler si vous ou Claude invoquez la skill](#), et la capacité pour Claude de les charger automatiquement quand c'est pertinent.

Les skills Claude Code suivent la norme ouverte [Agent Skills](#), qui fonctionne sur plusieurs outils d'IA. Claude Code étend la norme avec des fonctionnalités supplémentaires comme [le contrôle d'invocation](#), [l'exécution de subagent](#), et [l'injection de contexte dynamique](#).

Skills groupées

Les skills groupées sont livrées avec Claude Code et sont disponibles dans chaque session. Contrairement aux [commandes intégrées](#), qui exécutent une logique fixe directement, les skills groupées sont basées sur des prompts : elles donnent à Claude un manuel détaillé et le laissent orchestrer le travail en utilisant ses outils. Cela signifie que les skills groupées peuvent générer des agents parallèles, lire des fichiers et s'adapter à votre base de code.

Vous invoquez les skills groupées de la même manière que n'importe quelle autre skill : tapez `/` suivi du nom de la skill.

- `/simplify` : examine vos fichiers récemment modifiés pour les problèmes de réutilisation de code, de qualité et d'efficacité, puis les corrige. Exécutez-la après avoir implémenté une fonctionnalité ou une correction de bug pour nettoyer votre travail. Elle génère trois agents d'examen en parallèle (réutilisation de code, qualité du code,

efficacité), agrège leurs conclusions et applique les corrections. Passez du texte optionnel pour vous concentrer sur des préoccupations spécifiques : `/simplify focus on memory efficiency`.

- `/batch <instruction>` : orchestre les changements à grande échelle dans une base de code en parallèle. Fournissez une description du changement et `/batch` recherche la base de code, décompose le travail en 5 à 30 unités indépendantes, et présente un plan pour votre approbation. Une fois approuvé, il génère un agent d'arrière-plan par unité, chacun dans un [git worktree](#) isolé. Chaque agent implémente son unité, exécute les tests et ouvre une demande de tirage. Nécessite un référentiel git. Exemple : `/batch migrate src/ from Solid to React`.
- `/debug [description]` : dépanne votre session Claude Code actuelle en lisant le journal de débogage de la session. Décrivez optionnellement le problème pour concentrer l'analyse.
- `/loop [interval] <prompt>` : exécute un prompt à plusieurs reprises à un intervalle donné tant que la session reste ouverte. Claude analyse l'intervalle, planifie une tâche cron récurrente et confirme la cadence. Utile pour interroger un déploiement, surveiller une demande de tirage ou réexécuter périodiquement une autre skill. Exemple : `/loop 5m check if the deploy finished`. Voir [Exécuter des prompts selon un calendrier](#).
- `/claude-api` : charge le matériel de référence de l'API Claude pour le langage de votre projet (Python, TypeScript, Java, Go, Ruby, C#, PHP ou cURL) et la référence du SDK Agent pour Python et TypeScript. Couvre l'utilisation d'outils, le streaming, les lots, les sorties structurées et les pièges courants. S'active également automatiquement quand votre code importe `anthropic`, `@anthropic-ai/sdk` ou `claude_agent_sdk`.

Démarrage

Créez votre première skill

Cet exemple crée une skill qui enseigne à Claude comment expliquer le code en utilisant des diagrammes visuels et des analogies. Puisqu'elle utilise le frontmatter par défaut, Claude peut la charger automatiquement quand vous demandez comment quelque chose fonctionne, ou vous pouvez l'invoquer directement avec `/explain-code`.

Step 1: Créez le répertoire de la skill

Créez un répertoire pour la skill dans votre dossier de skills personnelles. Les skills personnelles sont disponibles dans tous vos projets.

```
mkdir -p ~/.claude/skills/explain-code
```

Step 2: Écrivez SKILL.md

Chaque skill a besoin d'un fichier **SKILL.md** avec deux parties : un frontmatter YAML (entre les marqueurs `---`) qui indique à Claude quand utiliser la skill, et du contenu markdown avec les instructions que Claude suit quand la skill est invoquée. Le champ `name` devient la `/slash-command`, et la `description` aide Claude à décider quand la charger automatiquement.

Créez `~/.claude/skills/explain-code/SKILL.md` :

```
---
name: explain-code
description: Explains code with visual diagrams and analogies. Use when explaining
how code works, teaching about a codebase, or when the user asks "how does this
work?"
---
```

When explaining code, always include:

- **Start with an analogy****: Compare the code to something from everyday life
- **Draw a diagram****: Use ASCII art to show the flow, structure, or relationships
- **Walk through the code****: Explain step-by-step what happens
- **Highlight a gotcha****: What's a common mistake or misconception?

Keep explanations conversational. For complex concepts, use multiple analogies.

Step 3: Testez la skill

Vous pouvez la tester de deux façons :

Laissez Claude l'invoquer automatiquement en demandant quelque chose qui correspond à la description :

```
How does this code work?
```

Ou invoquez-la directement avec le nom de la skill :

```
/explain-code src/auth/login.ts
```

De l’une ou l’autre façon, Claude devrait inclure une analogie et un diagramme ASCII dans son explication.

Où vivent les skills

L’endroit où vous stockez une skill détermine qui peut l’utiliser :

Emplacement	Chemin	S’applique à
Entreprise	Voir paramètres gérés	Tous les utilisateurs de votre organisation
Personnel	<code>~/.claude/skills/<skill-name>/SKILL.md</code>	Tous vos projets
Projet	<code>.claude/skills/<skill-name>/SKILL.md</code>	Ce projet uniquement
Plugin	<code><plugin>/skills/<skill-name>/SKILL.md</code>	Où le plugin est activé

Quand les skills partagent le même nom sur plusieurs niveaux, les emplacements de priorité plus élevée gagnent : entreprise > personnel > projet. Les skills de plugin utilisent un espace de noms `plugin-name:skill-name`, donc elles ne peuvent pas entrer en conflit avec d’autres niveaux. Si vous avez des fichiers dans `.claude/commands/`, ils fonctionnent de la même manière, mais si une skill et une commande partagent le même nom, la skill a la priorité.

Découverte automatique à partir de répertoires imbriqués

Quand vous travaillez avec des fichiers dans des sous-répertoires, Claude Code découvre automatiquement les skills à partir des répertoires `.claude/skills/` imbriqués. Par exemple, si vous modifiez un fichier dans `packages/frontend/`, Claude Code recherche également les skills dans `packages/frontend/.claude/skills/`. Cela supporte les configurations monorepo où les packages ont leurs propres skills.

Chaque skill est un répertoire avec `SKILL.md` comme point d’entrée :

```

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

```

Le **SKILL.md** contient les instructions principales et est obligatoire. Les autres fichiers sont optionnels et vous permettent de créer des skills plus puissantes : des modèles pour que Claude remplisse, des exemples de sortie montrant le format attendu, des scripts que Claude peut exécuter, ou une documentation de référence détaillée. Référez-vous à ces fichiers à partir de votre **SKILL.md** pour que Claude sache ce que chaque fichier contient et quand le charger. Voir [Ajouter des fichiers de support](#) pour plus de détails.

Note:

Les fichiers dans **.claude/commands/** fonctionnent toujours et supportent le même [front-matter](#). Les skills sont recommandées puisqu'elles supportent des fonctionnalités supplémentaires comme les fichiers de support.

Skills à partir de répertoires supplémentaires

Les skills définies dans **.claude/skills/** dans les répertoires ajoutés via **--add-dir** sont chargées automatiquement et détectées par la détection de changement en direct, vous pouvez donc les modifier pendant une session sans redémarrer.

Note:

Les fichiers **CLAUDE.md** des répertoires **--add-dir** ne sont pas chargés par défaut. Pour les charger, définissez **CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1**. Voir [Charger à partir de répertoires supplémentaires](#).

Configurez les skills

Les skills sont configurées via le frontmatter YAML en haut de **SKILL.md** et le contenu markdown qui suit.

Types de contenu de skill

Les fichiers de skill peuvent contenir n'importe quelles instructions, mais réfléchir à la façon dont vous voulez les invoquer aide à guider ce qu'il faut inclure :

Le contenu de référence ajoute des connaissances que Claude applique à votre travail actuel. Conventions, modèles, guides de style, connaissances du domaine. Ce contenu s'exécute en ligne pour que Claude puisse l'utiliser aux côtés du contexte de votre conversation.

```
---
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

Le contenu de tâche donne à Claude des instructions étape par étape pour une action spécifique, comme les déploiements, les commits ou la génération de code. Ce sont souvent des actions que vous voulez invoquer directement avec `/skill-name` plutôt que de laisser Claude décider quand les exécuter. Ajoutez `disable-model-invocation: true` pour empêcher Claude de les déclencher automatiquement.

```
---
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

Votre `SKILL.md` peut contenir n'importe quoi, mais réfléchir à la façon dont vous voulez que la skill soit invoquée (par vous, par Claude, ou les deux) et où vous voulez qu'elle s'exécute (en ligne ou dans un subagent) aide à guider ce qu'il faut inclure. Pour les skills complexes, vous pouvez également [ajouter des fichiers de support](#) pour garder la skill principale concentrée.

Référence du frontmatter

Au-delà du contenu markdown, vous pouvez configurer le comportement de la skill en utilisant les champs du frontmatter YAML entre les marqueurs `---` en haut de votre fichier `SKILL.md` :

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read, Grep
---

Your skill instructions here...
```

Tous les champs sont optionnels. Seul `description` est recommandé pour que Claude sache quand utiliser la skill.

Champ	Obligatoire	Description
<code>name</code>	Non	Nom d'affichage pour la skill. S'il est omis, utilise le nom du répertoire. Lettres minuscules, chiffres et tirets uniquement (max 64 caractères).
<code>description</code>	Recommandé	Ce que fait la skill et quand l'utiliser. Claude utilise ceci pour décider quand appliquer la skill. S'il est omis, utilise le premier paragraphe du contenu markdown.

Champ	Obligatoire	Description
<code>argument-hint</code>	Non	Indice affiché lors de l'autocomplétion pour indiquer les arguments attendus. Exemple : <code>[issue-number]</code> ou <code>[filename][format]</code> .
<code>disable-model-invocation</code>	Non	Définissez sur <code>true</code> pour empêcher Claude de charger automatiquement cette skill. Utilisez pour les workflows que vous voulez déclencher manuellement avec <code>/name</code> . Par défaut : <code>false</code> .
<code>user-invocable</code>	Non	Définissez sur <code>false</code> pour masquer du menu <code>/</code> . Utilisez pour les connaissances d'arrière-plan que les utilisateurs ne devraient pas invoquer directement. Par défaut : <code>true</code> .
<code>allowed-tools</code>	Non	Outils que Claude peut utiliser sans demander la permission quand cette skill est active.
<code>model</code>	Non	Modèle à utiliser quand cette skill est active.
<code>context</code>	Non	Définissez sur <code>fork</code> pour exécuter dans un contexte de subagent forké.
<code>agent</code>	Non	Quel type de subagent utiliser quand <code>context: fork</code> est défini.

Champ	Obligatoire	Description
hooks	Non	Hooks limités au cycle de vie de cette skill. Voir Hooks dans les skills et les agents pour le format de configuration.

Substitutions de chaîne disponibles

Les skills supportent la substitution de chaîne pour les valeurs dynamiques dans le contenu de la skill :

Variable	Description
\$ARGUMENTS	Tous les arguments passés lors de l’invocation de la skill. Si \$ARGUMENTS n’est pas présent dans le contenu, les arguments sont ajoutés comme ARGUMENTS: <value> .
\$ARGUMENTS[N]	Accédez à un argument spécifique par index basé sur 0, comme \$ARGUMENTS[0] pour le premier argument.
\$N	Raccourci pour \$ARGUMENTS[N] , comme \$0 pour le premier argument ou \$1 pour le second.
\$_ {CLAUDE_SESSION_ID }	L’ID de session actuel. Utile pour la journalisation, la création de fichiers spécifiques à la session ou la corrélation de la sortie de la skill avec les sessions.
\$_ {CLAUDE_SKILL_DIR}	Le répertoire contenant le fichier SKILL.md de la skill. Pour les skills de plugin, c’est le sous-répertoire de la skill dans le plugin, pas la racine du plugin. Utilisez ceci dans les commandes d’injection bash pour référencer les scripts ou les fichiers fournis avec la skill, indépendamment du répertoire de travail actuel.

Exemple utilisant les substitutions :


```

---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS

```

Ajouter des fichiers de support

Les skills peuvent inclure plusieurs fichiers dans leur répertoire. Cela garde `SKILL.md` concentré sur l'essentiel tout en permettant à Claude d'accéder au matériel de référence détaillé uniquement quand nécessaire. Les grandes docs de référence, les spécifications d'API ou les collections d'exemples n'ont pas besoin de se charger dans le contexte à chaque fois que la skill s'exécute.

```

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)

```

Référez les fichiers de support à partir de `SKILL.md` pour que Claude sache ce que chaque fichier contient et quand le charger :

```

### Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)

```

Tip:

Gardez `SKILL.md` sous 500 lignes. Déplacez le matériel de référence détaillé vers des fichiers séparés.

Contrôlez qui invoque une skill

Par défaut, vous et Claude pouvez invoquer n'importe quelle skill. Vous pouvez taper `/skill-name` pour l'invoquer directement, et Claude peut la charger automatiquement quand c'est pertinent pour votre conversation. Deux champs du frontmatter vous permettent de restreindre ceci :

- `disable-model-invocation: true` : Seul vous pouvez invoquer la skill. Utilisez ceci pour les workflows avec des effets secondaires ou que vous voulez contrôler le timing, comme `/commit`, `/deploy` ou `/send-slack-message`. Vous ne voulez pas que Claude décide de déployer parce que votre code semble prêt.
- `user-invocable: false` : Seul Claude peut invoquer la skill. Utilisez ceci pour les connaissances d'arrière-plan qui ne sont pas actionnables en tant que commande. Une skill `legacy-system-context` explique comment fonctionne un ancien système. Claude devrait le savoir quand c'est pertinent, mais `/legacy-system-context` n'est pas une action significative pour les utilisateurs.

Cet exemple crée une skill de déploiement que seul vous pouvez déclencher. Le champ `disable-model-invocation: true` empêche Claude de l'exécuter automatiquement :

```
---
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

Voici comment les deux champs affectent l'invocation et le chargement du contexte :

Front-matter	Vous pouvez invoquer	Claude peut invoquer	Quand chargé dans le contexte
(par défaut)	Oui	Oui	Description toujours dans le contexte, la skill complète se charge quand invoquée
<code>disable-model-invocation: true</code>	Oui	Non	Description pas dans le contexte, la skill complète se charge quand vous l'invoquez
<code>user-invocable: false</code>	Non	Oui	Description toujours dans le contexte, la skill complète se charge quand invoquée

Note:

Dans une session régulière, les descriptions de skills sont chargées dans le contexte pour que Claude sache ce qui est disponible, mais le contenu complet de la skill ne se charge que quand invoquée. [Les subagents avec des skills préchargées](#) fonctionnent différemment : le contenu complet de la skill est injecté au démarrage.

Restreindre l'accès aux outils

Utilisez le champ `allowed-tools` pour limiter les outils que Claude peut utiliser quand une skill est active. Cette skill crée un mode lecture seule où Claude peut explorer les fichiers mais pas les modifier :

```
---
name: safe-reader
description: Read files without making changes
allowed-tools: Read, Grep, Glob
---
```

Passez des arguments aux skills

Vous et Claude pouvez passer des arguments lors de l'invocation d'une skill. Les arguments sont disponibles via l'espace réservé `$ARGUMENTS`.

Cette skill corrige un problème GitHub par numéro. L'espace réservé `$ARGUMENTS` est remplacé par tout ce qui suit le nom de la 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
```

Quand vous exécutez `/fix-issue 123`, Claude reçoit 'Fix GitHub issue 123 following our coding standards...'

Si vous invoquez une skill avec des arguments mais que la skill n'inclut pas `$ARGUMENTS`, Claude Code ajoute `ARGUMENTS: <your input>` à la fin du contenu de la skill pour que Claude voie toujours ce que vous avez tapé.

Pour accéder aux arguments individuels par position, utilisez `$ARGUMENTS[N]` ou le raccourci plus court `$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.
```

Exécuter `/migrate-component SearchBar React Vue` remplace `$ARGUMENTS[0]` par `SearchBar`, `$ARGUMENTS[1]` par `React` et `$ARGUMENTS[2]` par `Vue`. La même skill utilisant le raccourci `$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.
```

Modèles avancés

Injectez du contexte dynamique

La syntaxe `! command``` exécute les commandes shell avant que le contenu de la skill soit envoyé à Claude. La sortie de la commande remplace l'espace réservé, donc Claude reçoit les données réelles, pas la commande elle-même.

Cette skill résume une demande de tirage en récupérant les données de demande de tirage en direct avec le CLI GitHub. Les commandes `! gh pr diff``` et autres s'exécutent d'abord, et leur sortie est insérée dans le 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...

```

Quand cette skill s'exécute :

1. Chaque **!** command`` s'exécute immédiatement (avant que Claude ne voie quoi que ce soit)
2. La sortie remplace l'espace réservé dans le contenu de la skill
3. Claude reçoit le prompt complètement rendu avec les données réelles de la demande de tirage

C'est du prétraitement, pas quelque chose que Claude exécute. Claude ne voit que le résultat final.

Tip:

Pour activer la [réflexion étendue](#) dans une skill, incluez le mot 'ultrathink' n'importe où dans le contenu de votre skill.

Exécutez les skills dans un subagent

Ajoutez **context: fork** à votre frontmatter quand vous voulez qu'une skill s'exécute en isolation. Le contenu de la skill devient le prompt qui pilote le subagent. Il n'aura pas accès à votre historique de conversation.

Warning:

context: fork n'a de sens que pour les skills avec des instructions explicites. Si votre skill contient des directives comme 'utilisez ces conventions d'API ' sans une tâche, le subagent reçoit les directives mais aucun prompt actionnable, et retourne sans sortie significative.

Les skills et les [subagents](#) fonctionnent ensemble dans deux directions :

Ap-proche	Prompt système	Tâche	Charge égale-ment
Skill avec context: fork	Du type d'agent (Explore , Plan , etc.)	Contenu SKILL.md	CLAUDE.md
Subagent avec champ skills	Corps markdown du subagent	Message de délégation de Claude	Skills préchargées + CLAUDE.md

Avec **context: fork**, vous écrivez la tâche dans votre skill et choisissez un type d'agent pour l'exécuter. Pour l'inverse (définir un subagent personnalisé qui utilise les skills comme matériel de référence), voir [Subagents](#).

Exemple : Skill de recherche utilisant l'agent Explore

Cette skill exécute la recherche dans un agent Explore forké. Le contenu de la skill devient la tâche, et l'agent fournit des outils en lecture seule optimisés pour l'exploration de la base de code :

```
---
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
```

Quand cette skill s'exécute :

1. Un nouveau contexte isolé est créé
2. Le subagent reçoit le contenu de la skill comme son prompt (' Research \$ARGUMENTS thoroughly... ')
3. Le champ `agent` détermine l'environnement d'exécution (modèle, outils et permissions)
4. Les résultats sont résumés et retournés à votre conversation principale

Le champ `agent` spécifie quelle configuration de subagent utiliser. Les options incluent les agents intégrés (`Explore` , `Plan` , `general-purpose`) ou n'importe quel subagent personnalisé de `.claude/agents/` . S'il est omis, utilise `general-purpose` .

Restreindre l'accès aux skills de Claude

Par défaut, Claude peut invoquer n'importe quelle skill qui n'a pas `disable-model-invocation: true` défini. Les skills qui définissent `allowed-tools` accordent à Claude l'accès à ces outils sans approbation par utilisation quand la skill est active. Vos [paramètres de permission](#) gouvernent toujours le comportement d'approbation de base pour tous les autres outils. Les commandes intégrées comme `/compact` et `/init` ne sont pas disponibles via l'outil Skill.

Trois façons de contrôler quelles skills Claude peut invoquer :

Désactivez toutes les skills en refusant l'outil Skill dans `/permissions` :

```
## Add to deny rules:
Skill
```

Autorisez ou refusez des skills spécifiques en utilisant les [règles de permission](#) :

```
## Allow only specific skills
Skill(commit)
Skill(review-pr *)

## Deny specific skills
Skill(deploy *)
```

Syntaxe de permission : `Skill(name)` pour une correspondance exacte, `Skill(name *)` pour une correspondance de préfixe avec n'importe quels arguments.

Masquez les skills individuelles en ajoutant `disable-model-invocation: true` à leur frontmatter. Cela supprime la skill du contexte de Claude entièrement.

Note:

Le champ `user-invocable` contrôle uniquement la visibilité du menu, pas l'accès à l'outil Skill. Utilisez `disable-model-invocation: true` pour bloquer l'invocation programmatique.

Partagez les skills

Les skills peuvent être distribuées à différentes portées selon votre audience :

- **Skills de projet** : Validez `.claude/skills/` dans le contrôle de version
- **Plugins** : Créez un répertoire `skills/` dans votre [plugin](#)
- **Gérées** : Déployez à l'échelle de l'organisation via les [paramètres gérés](#)

Générez une sortie visuelle

Les skills peuvent regrouper et exécuter des scripts dans n'importe quel langage, donnant à Claude des capacités au-delà de ce qui est possible dans un seul prompt. Un modèle puissant est la génération de sortie visuelle : des fichiers HTML interactifs qui s'ouvrent dans votre navigateur pour explorer les données, déboguer ou créer des rapports.

Cet exemple crée un explorateur de base de code : une vue d'arbre interactive où vous pouvez développer et réduire les répertoires, voir les tailles de fichiers en un coup d'œil et identifier les types de fichiers par couleur.

Créez le répertoire Skill :

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

Créez `~/.claude/skills/codebase-visualizer/SKILL.md`. La description indique à Claude quand activer cette Skill, et les instructions indiquent à Claude d'exécuter le script fourni :

```

---
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

```

Créez `~/.claude/skills/codebase-visualizer/scripts/visualize.py`. Ce script analyse une arborescence de répertoires et génère un fichier HTML autonome avec :

- Une **barre latérale de résumé** montrant le nombre de fichiers, le nombre de répertoires, la taille totale et le nombre de types de fichiers
- Un **graphique en barres** décomposant la base de code par type de fichier (top 8 par taille)

- Une **arborescence réductible** où vous pouvez développer et réduire les répertoires, avec des indicateurs de type de fichier codés par couleur

Le script nécessite Python mais utilise uniquement les bibliothèques intégrées, donc il n'y a pas de packages à installer :

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

from pathlib import Path
from collections import Counter

IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) → dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size,
"ext": ext})

                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
        except PermissionError:
            pass
    return result

def generate_html(data: dict, stats: dict, output: Path) → None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[0:8]
    colors = {
```

```

        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e
    ',

        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }

    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{color
s.get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )

    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
<meta charset="utf-8"><title>Codebase Explorer</title>
<style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e;
color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right:
1px solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform:
uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0;
border-bottom: 1px solid #3d3d5c; }}
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}

```

```

    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius:
4px; }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
</style>
</head><body>
  <div class="container">
    <div class="sidebar">
      <h1><img alt="Summary icon" data-bbox="211 356 228 371"/> Summary</h1>
      <div class="stat"><span>Files</span><span class="stat-
value">{stats["files"]:,}</span></div>
      <div class="stat"><span>Directories</span><span class="stat-value">{stats["d
irs"]:,}</span></div>
      <div class="stat"><span>Total size</span><span class="stat-
value">{fmt(data["size"])}</span></div>
      <div class="stat"><span>File types</span><span class="stat-value">{len(stats
["extensions"])}</span></div>
      <h2>By file type</h2>
      {lang_bars}
    </div>
    <div class="main">
      <h1><img alt="Folder icon" data-bbox="211 634 228 649"/> {data["name"]}</h1>
      <ul class="tree" id="root"></ul>
    </div>
  </div>
<script>
  const data = {json.dumps(data)};
  const colors = {json.dumps(colors)};
  function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/
1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
  function render(node, parent) {{
    if (node.children) {{
      const det = document.createElement('details');
      det.open = parent ≡ document.getElementById('root');

```

```

    det.innerHTML = `<summary><span class="folder">📁 ${{node.name}}</
span><span class="size">${{fmt(node.size)}}</span></summary>`;
    const ul = document.createElement('ul'); ul.className = 'tree';
    node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) ||
a.name.localeCompare(b.name));
    node.children.forEach(c => render(c, ul));
    det.appendChild(ul);
    const li = document.createElement('li'); li.appendChild(det);
    parent.appendChild(li);
  } else {
    const li = document.createElement('li'); li.className = 'file';
    li.innerHTML = `<span class="dot" style="background:${{colors[node.ext]}}
#6b7280'`"></span>${{node.name}}<span class="size">${{fmt(node.size)}}</span>`;
    parent.appendChild(li);
  }
}
}

data.children.forEach(c => render(c, document.getElementById('root')));
</script>
</body></html>'''
    output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes":
Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

Pour tester, ouvrez Claude Code dans n'importe quel projet et demandez « Visualize this codebase. » Claude exécute le script, génère `codebase-map.html` et l'ouvre dans votre navigateur.

Ce modèle fonctionne pour n'importe quelle sortie visuelle : graphiques de dépendances, rapports de couverture de test, documentation d'API ou visualisations de schéma de base de données. Le script fourni fait le gros du travail tandis que Claude gère l'orchestration.

Dépannage

La skill ne se déclenche pas

Si Claude n'utilise pas votre skill quand attendu :

1. Vérifiez que la description inclut les mots-clés que les utilisateurs diraient naturellement
2. Vérifiez que la skill apparaît dans « What skills are available? »
3. Essayez de reformuler votre demande pour correspondre plus étroitement à la description
4. Invoquez-la directement avec `/skill-name` si la skill est invocable par l'utilisateur

La skill se déclenche trop souvent

Si Claude utilise votre skill quand vous ne le voulez pas :

1. Rendez la description plus spécifique
2. Ajoutez `disable-model-invocation: true` si vous voulez uniquement l'invocation manuelle

Claude ne voit pas toutes mes skills

Les descriptions de skills sont chargées dans le contexte pour que Claude sache ce qui est disponible. Si vous avez beaucoup de skills, elles peuvent dépasser le budget de caractères. Le budget s'adapte dynamiquement à 2 % de la fenêtre de contexte, avec un repli de 16 000 caractères. Exécutez `/context` pour vérifier un avertissement concernant les skills exclues.

Pour remplacer la limite, définissez la variable d'environnement

`SLASH_COMMAND_TOOL_CHAR_BUDGET` .

Ressources connexes

- [Subagents](#) : déléguez les tâches à des agents spécialisés
- [Plugins](#) : empaquetez et distribuez les skills avec d'autres extensions
- [Hooks](#) : automatisez les workflows autour des événements d'outils
- [Memory](#) : gérez les fichiers CLAUDE.md pour le contexte persistant
- [Mode interactif](#) : commandes intégrées et raccourcis
- [Permissions](#) : contrôlez l'accès aux outils et aux skills

Créer des plugins

Créez des plugins personnalisés pour étendre Claude Code avec des skills, des agents, des hooks et des serveurs MCP.

Les plugins vous permettent d'étendre Claude Code avec des fonctionnalités personnalisées qui peuvent être partagées entre les projets et les équipes. Ce guide couvre la création de vos propres plugins avec des skills, des agents, des hooks et des serveurs MCP.

Vous cherchez à installer des plugins existants ? Consultez [Découvrir et installer des plugins](#). Pour les spécifications techniques complètes, consultez [Référence des plugins](#).

Quand utiliser les plugins par rapport à la configuration autonome

Claude Code prend en charge deux façons d'ajouter des skills, des agents et des hooks personnalisés :

Approche	Noms des skills	Idéal pour
Autonome (répertoire <code>.claude/</code>)	<code>/hello</code>	Flux de travail personnels, personnalisations spécifiques au projet, expériences rapides
Plugins (répertoires avec <code>.claude-plugin/plugin.json</code>)	<code>/plugin-name:hello</code>	Partage avec les coéquipiers, distribution à la communauté, versions publiées, réutilisable entre les projets

Utilisez la configuration autonome quand :

- Vous personnalisez Claude Code pour un seul projet
- La configuration est personnelle et n'a pas besoin d'être partagée
- Vous expérimentez avec des skills ou des hooks avant de les empaqueter
- Vous voulez des noms de skills courts comme `/hello` ou `/deploy`

Utilisez les plugins quand :

- Vous voulez partager des fonctionnalités avec votre équipe ou la communauté

- Vous avez besoin des mêmes skills/agents sur plusieurs projets
- Vous voulez le contrôle de version et les mises à jour faciles pour vos extensions
- Vous distribuez via une marketplace
- Vous êtes d'accord avec les skills avec espace de noms comme `/my-plugin:hello` (l'espace de noms prévient les conflits entre les plugins)

Tip:

Commencez par la configuration autonome dans `.claude/` pour une itération rapide, puis [convertissez en plugin](#) quand vous êtes prêt à partager.

Démarrage rapide

Ce démarrage rapide vous guide dans la création d'un plugin avec un skill personnalisé. Vous allez créer un manifeste (le fichier de configuration qui définit votre plugin), ajouter un skill et le tester localement en utilisant le drapeau `--plugin-dir`.

Prérequis

- Claude Code [installé et authentifié](#)
- Claude Code version 1.0.33 ou ultérieure (exécutez `claude --version` pour vérifier)

Note:

Si vous ne voyez pas la commande `/plugin`, mettez à jour Claude Code vers la dernière version. Consultez [Dépannage](#) pour les instructions de mise à niveau.

Créez votre premier plugin

Step 1: Créez le répertoire du plugin

Chaque plugin se trouve dans son propre répertoire contenant un manifeste et vos skills, agents ou hooks. Créez-en un maintenant :

```
mkdir my-first-plugin
```

Step 2: Créez le manifeste du plugin

Le fichier manifeste à `.claude-plugin/plugin.json` définit l'identité de votre plugin : son nom, sa description et sa version. Claude Code utilise ces métadonnées pour afficher votre plugin dans le gestionnaire de plugins.

Créez le répertoire `.claude-plugin` à l'intérieur de votre dossier de plugin :

```
mkdir my-first-plugin/.claude-plugin
```

Ensuite, créez `my-first-plugin/.claude-plugin/plugin.json` avec ce contenu :

```
{
  "name": "my-first-plugin",
  "description": "A greeting plugin to learn the basics",
  "version": "1.0.0",
  "author": {
    "name": "Your Name"
  }
}
```

Champ	Objectif
<code>name</code>	Identifiant unique et espace de noms du skill. Les skills sont préfixés avec ceci (par exemple, <code>/my-first-plugin:hello</code>).
<code>description</code>	Affiché dans le gestionnaire de plugins lors de la navigation ou de l'installation de plugins.
<code>version</code>	Suivez les versions en utilisant le versioning sémantique .
<code>author</code>	Optionnel. Utile pour l'attribution.

Pour les champs supplémentaires comme `homepage`, `repository` et `license`, consultez le [schéma manifeste complet](#).

Step 3: Ajoutez un skill

Les skills se trouvent dans le répertoire `skills/`. Chaque skill est un dossier contenant un fichier `SKILL.md`. Le nom du dossier devient le nom du skill, préfixé par l'espace de noms du plugin (`hello/` dans un plugin nommé `my-first-plugin` crée `/my-first-plugin:hello`).

Créez un répertoire de skill dans votre dossier de plugin :

```
mkdir -p my-first-plugin/skills/hello
```

Ensuite, créez `my-first-plugin/skills/hello/SKILL.md` avec ce contenu :

```
---
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: Testez votre plugin

Exécutez Claude Code avec le drapeau `--plugin-dir` pour charger votre plugin :

```
claude --plugin-dir ./my-first-plugin
```

Une fois Claude Code démarré, essayez votre nouveau skill :

```
/my-first-plugin:hello
```

Vous verrez Claude répondre avec un salut. Exécutez `/help` pour voir votre skill listé sous l'espace de noms du plugin.

Note:

Pourquoi l'espace de noms ? Les skills des plugins sont toujours avec espace de noms (comme `/greet:hello`) pour prévenir les conflits quand plusieurs plugins ont des skills avec le même nom.

Pour changer le préfixe d'espace de noms, mettez à jour le champ `name` dans `plugin.json`.

Step 5: Ajoutez des arguments au skill

Rendez votre skill dynamique en acceptant l'entrée de l'utilisateur. L'espace réservé `$ARGUMENTS` capture tout texte que l'utilisateur fournit après le nom du skill.

Mettez à jour votre fichier `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.
```

Exécutez `/reload-plugins` pour récupérer les modifications, puis essayez le skill avec votre nom :

```
/my-first-plugin:hello Alex
```

Claude vous saluera par votre nom. Pour plus d'informations sur la transmission d'arguments aux skills, consultez [Skills](#).

Vous avez créé et testé avec succès un plugin avec ces composants clés :

- **Manifeste du plugin** (`.claude-plugin/plugin.json`) : décrit les métadonnées de votre plugin
- **Répertoire des skills** (`skills/`) : contient vos skills personnalisés
- **Arguments du skill** (`$ARGUMENTS`) : capture l'entrée de l'utilisateur pour un comportement dynamique

Tip:

Le drapeau `--plugin-dir` est utile pour le développement et les tests. Quand vous êtes prêt à partager votre plugin avec d'autres, consultez [Créer et distribuer une marketplace de plugins](#).

Aperçu de la structure du plugin

Vous avez créé un plugin avec un skill, mais les plugins peuvent inclure beaucoup plus : des agents personnalisés, des hooks, des serveurs MCP et des serveurs LSP.

Warning:

Erreur courante : Ne mettez pas `commands/` , `agents/` , `skills/` ou `hooks/` à l'intérieur du répertoire `.claude-plugin/` . Seul `plugin.json` va à l'intérieur de `.claude-plugin/` . Tous les autres répertoires doivent être au niveau racine du plugin.

Répertoire	Emplacement	Objectif
<code>.claude-plugin/</code>	Racine du plugin	Contient le manifeste <code>plugin.json</code> (optionnel si les composants utilisent les emplacements par défaut)
<code>commands/</code>	Racine du plugin	Skills en tant que fichiers Markdown
<code>agents/</code>	Racine du plugin	Définitions d'agents personnalisés
<code>skills/</code>	Racine du plugin	Agent Skills avec fichiers <code>SKILL.md</code>
<code>hooks/</code>	Racine du plugin	Gestionnaires d'événements dans <code>hooks.json</code>
<code>.mcp.json</code>	Racine du plugin	Configurations du serveur MCP
<code>.lsp.json</code>	Racine du plugin	Configurations du serveur LSP pour l'intelligence du code
<code>settings.json</code>	Racine du plugin	Paramètres par défaut appliqués quand le plugin est activé

Note:

Prochaines étapes : Prêt à ajouter plus de fonctionnalités ? Allez à [Développer des plug-ins plus complexes](#) pour ajouter des agents, des hooks, des serveurs MCP et des serveurs LSP. Pour les spécifications techniques complètes de tous les composants du plugin, consultez [Référence des plug-ins](#).

Développer des plugins plus complexes

Une fois que vous êtes à l'aise avec les plugins de base, vous pouvez créer des extensions plus sophistiquées.

Ajoutez des Skills à votre plugin

Les plugins peuvent inclure des [Agent Skills](#) pour étendre les capacités de Claude. Les skills sont invoqués par le modèle : Claude les utilise automatiquement en fonction du contexte de la tâche.

Ajoutez un répertoire `skills/` à la racine de votre plugin avec des dossiers de Skill contenant des fichiers `SKILL.md` :

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json
└── skills/
    └── code-review/
        └── SKILL.md
```

Chaque `SKILL.md` a besoin d'un frontmatter avec les champs `name` et `description`, suivi d'instructions :

```
---
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
```

Après l'installation du plugin, exécutez `/reload-plugins` pour charger les Skills. Pour des conseils complets sur la création de Skills incluant la divulgation progressive et les restrictions d'outils, consultez [Agent Skills](#).

Ajoutez des serveurs LSP à votre plugin

Tip:

Pour les langages courants comme TypeScript, Python et Rust, installez les plugins LSP pré-construits à partir de la marketplace officielle. Créez des plugins LSP personnalisés uniquement quand vous avez besoin de support pour des langages non encore couverts.

Les plugins LSP (Language Server Protocol) donnent à Claude l'intelligence du code en temps réel. Si vous avez besoin de supporter un langage qui n'a pas de plugin LSP officiel, vous pouvez en créer un en ajoutant un fichier `.lsp.json` à votre plugin :

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

Les utilisateurs qui installent votre plugin doivent avoir le binaire du serveur de langage installé sur leur machine.

Pour les options de configuration LSP complètes, consultez [Serveurs LSP](#).

Livrez les paramètres par défaut avec votre plugin

Les plugins peuvent inclure un fichier `settings.json` à la racine du plugin pour appliquer la configuration par défaut quand le plugin est activé. Actuellement, seule la clé `agent` est supportée.

Définir `agent` active l'un des [agents personnalisés](#) du plugin en tant que thread principal, en appliquant son invite système, ses restrictions d'outils et son modèle. Cela permet à un plugin de changer le comportement par défaut de Claude Code quand il est activé.


```
{
  "agent": "security-reviewer"
}
```

Cet exemple active l'agent `security-reviewer` défini dans le répertoire `agents/` du plugin. Les paramètres de `settings.json` ont priorité sur les `settings` déclarés dans `plugin.json`. Les clés inconnues sont silencieusement ignorées.

Organisez les plugins complexes

Pour les plugins avec de nombreux composants, organisez votre structure de répertoires par fonctionnalité. Pour les dispositions de répertoires complètes et les modèles d'organisation, consultez [Structure du répertoire du plugin](#).

Testez vos plugins localement

Utilisez le drapeau `--plugin-dir` pour tester les plugins pendant le développement. Cela charge votre plugin directement sans nécessiter d'installation.

```
claude --plugin-dir ./my-plugin
```

À mesure que vous apportez des modifications à votre plugin, exécutez `/reload-plugins` pour récupérer les mises à jour sans redémarrer. Les modifications de la configuration du serveur LSP nécessitent toujours un redémarrage complet. Testez vos composants de plugin :

- Essayez vos skills avec `/plugin-name:skill-name`
- Vérifiez que les agents apparaissent dans `/agents`
- Vérifiez que les hooks fonctionnent comme prévu

Tip:

Vous pouvez charger plusieurs plugins à la fois en spécifiant le drapeau plusieurs fois :

```
claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
```

Débuguez les problèmes de plugin

Si votre plugin ne fonctionne pas comme prévu :

1. **Vérifiez la structure** : Assurez-vous que vos répertoires sont à la racine du plugin, pas à l'intérieur de `.claude-plugin/`
2. **Testez les composants individuellement** : Vérifiez chaque commande, agent et hook séparément
3. **Utilisez les outils de validation et de débogage** : Consultez [Outils de débogage et de développement](#) pour les commandes CLI et les techniques de dépannage

Partagez vos plugins

Quand votre plugin est prêt à être partagé :

1. **Ajoutez de la documentation** : Incluez un `README.md` avec les instructions d'installation et d'utilisation
2. **Versionnez votre plugin** : Utilisez le [versioning sémantique](#) dans votre `plugin.json`
3. **Créez ou utilisez une marketplace** : Distribuez via des [marketplaces de plugins](#) pour l'installation
4. **Testez avec d'autres** : Faites tester le plugin par les membres de l'équipe avant une distribution plus large

Une fois que votre plugin est dans une marketplace, d'autres peuvent l'installer en utilisant les instructions dans [Découvrir et installer des plugins](#).

Soumettez votre plugin à la marketplace officielle

Pour soumettre un plugin à la marketplace officielle d'Anthropic, utilisez l'un des formulaires de soumission dans l'application :

- **Claude.ai** : claude.ai/settings/plugins/submit
- **Console** : platform.claude.com/plugins/submit

Note:

Pour les spécifications techniques complètes, les techniques de débogage et les stratégies de distribution, consultez [Référence des plugins](#).

Convertir les configurations existantes en plugins

Si vous avez déjà des skills ou des hooks dans votre répertoire `.claude/`, vous pouvez les convertir en plugin pour un partage et une distribution plus faciles.

Étapes de migration

Step 1: Créez la structure du plugin

Créez un nouveau répertoire de plugin :

```
mkdir -p my-plugin/.claude-plugin
```

Créez le fichier manifeste à `my-plugin/.claude-plugin/plugin.json` :

```
{
  "name": "my-plugin",
  "description": "Migrated from standalone configuration",
  "version": "1.0.0"
}
```

Step 2: Copiez vos fichiers existants

Copiez vos configurations existantes dans le répertoire du 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: Migrez les hooks

Si vous avez des hooks dans vos paramètres, créez un répertoire de hooks :

```
mkdir my-plugin/hooks
```

Créez `my-plugin/hooks/hooks.json` avec votre configuration de hooks. Copiez l'objet `hooks` de votre `.claude/settings.json` ou `settings.local.json`, car le format est le même. La commande reçoit l'entrée du hook en tant que JSON sur stdin, donc utilisez `jq` pour extraire le chemin du fichier :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command":
          "jq -r '.tool_input.file_path' | xargs npm run lint:fix" }]
      }
    ]
  }
}
```

Step 4: Testez votre plugin migré

Chargez votre plugin pour vérifier que tout fonctionne :

```
claude --plugin-dir ./my-plugin
```

Testez chaque composant : exécutez vos commandes, vérifiez que les agents apparaissent dans `/agents` et vérifiez que les hooks se déclenchent correctement.

Ce qui change lors de la migration

Autonome (<code>.claude/</code>)	Plugin
Disponible uniquement dans un projet	Peut être partagé via des marketplaces
Fichiers dans <code>.claude/commands/</code>	Fichiers dans <code>plugin-name/commands/</code>
Hooks dans <code>settings.json</code>	Hooks dans <code>hooks/hooks.json</code>
Doit être copié manuellement pour partager	Installer avec <code>/plugin install</code>

Note:

Après la migration, vous pouvez supprimer les fichiers originaux de `.claude/` pour éviter les doublons. La version du plugin aura la priorité quand elle est chargée.

Prochaines étapes

Maintenant que vous comprenez le système de plugins de Claude Code, voici les chemins suggérés pour différents objectifs :

Pour les utilisateurs de plugins

- [Découvrir et installer des plugins](#) : parcourir les marketplaces et installer des plugins
- [Configurer les marketplaces d'équipe](#) : configurer les plugins au niveau du référentiel pour votre équipe

Pour les développeurs de plugins

- [Créer et distribuer une marketplace](#) : emballer et partager vos plugins
- [Référence des plugins](#) : spécifications techniques complètes
- Approfondissez les composants spécifiques du plugin :
 - [Skills](#) : détails du développement des skills
 - [Subagents](#) : configuration et capacités des agents
 - [Hooks](#) : gestion des événements et automatisation
 - [MCP](#) : intégration d'outils externes

Référence des plugins

Référence technique complète du système de plugins Claude Code, incluant les schémas, les commandes CLI et les spécifications des composants.

Tip:

Vous cherchez à installer des plugins ? Consultez [Découvrir et installer des plugins](#). Pour créer des plugins, consultez [Plugins](#). Pour distribuer des plugins, consultez [Marketplaces de plugins](#).

Cette référence fournit les spécifications techniques complètes du système de plugins Claude Code, incluant les schémas de composants, les commandes CLI et les outils de développement.

Un **plugin** est un répertoire autonome de composants qui étend Claude Code avec des fonctionnalités personnalisées. Les composants de plugin incluent les skills, les agents, les hooks, les serveurs MCP et les serveurs LSP.

Référence des composants de plugin

Skills

Les plugins ajoutent des skills à Claude Code, créant des raccourcis `/name` que vous ou Claude pouvez invoquer.

Emplacement : répertoire `skills/` ou `commands/` à la racine du plugin

Format de fichier : Les skills sont des répertoires avec `SKILL.md` ; les commandes sont des fichiers markdown simples

Structure des skills :

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (optionnel)
│   └── scripts/ (optionnel)
└── code-reviewer/
    └── SKILL.md
```

Comportement d'intégration :

- Les skills et les commandes sont découverts automatiquement lors de l'installation du plugin
- Claude peut les invoquer automatiquement en fonction du contexte de la tâche
- Les skills peuvent inclure des fichiers de support à côté de SKILL.md

Pour plus de détails, consultez [Skills](#).

Agents

Les plugins peuvent fournir des subagents spécialisés pour des tâches spécifiques que Claude peut invoquer automatiquement si approprié.

Emplacement : répertoire `agents/` à la racine du plugin

Format de fichier : Fichiers markdown décrivant les capacités de l'agent

Structure de l'agent :

```
---
name: agent-name
description: Ce dans quoi cet agent se spécialise et quand Claude devrait
l'invoquer
---
```

Invite système détaillée pour l'agent décrivant son rôle, son expertise et son comportement.

Points d'intégration :

- Les agents apparaissent dans l'interface `/agents`
- Claude peut invoquer les agents automatiquement en fonction du contexte de la tâche

- Les agents peuvent être invoqués manuellement par les utilisateurs
- Les agents de plugin fonctionnent aux côtés des agents Claude intégrés

Pour plus de détails, consultez [Subagents](#).

Hooks

Les plugins peuvent fournir des gestionnaires d'événements qui répondent automatiquement aux événements de Claude Code.

Emplacement : `hooks/hooks.json` à la racine du plugin, ou en ligne dans `plugin.json`

Format : Configuration JSON avec des correspondances d'événements et des actions

Configuration des hooks :

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

Événements disponibles :

- `PreToolUse` : Avant que Claude utilise un outil
- `PostToolUse` : Après que Claude utilise avec succès un outil
- `PostToolUseFailure` : Après l'échec de l'exécution d'un outil par Claude
- `PermissionRequest` : Quand une boîte de dialogue de permission est affichée
- `UserPromptSubmit` : Quand l'utilisateur soumet une invite
- `Notification` : Quand Claude Code envoie des notifications
- `Stop` : Quand Claude tente d'arrêter

- `SubagentStart` : Quand un subagent est démarré
- `SubagentStop` : Quand un subagent tente d'arrêter
- `SessionStart` : Au début des sessions
- `SessionEnd` : À la fin des sessions
- `TeammateIdle` : Quand un coéquipier d'une équipe d'agents est sur le point de devenir inactif
- `TaskCompleted` : Quand une tâche est marquée comme complétée
- `PreCompact` : Avant que l'historique de conversation soit compacté

Types de hooks :

- `command` : Exécuter des commandes shell ou des scripts
- `prompt` : Évaluer une invite avec un LLM (utilise l'espace réservé `$ARGUMENTS` pour le contexte)
- `agent` : Exécuter un vérificateur agentic avec des outils pour les tâches de vérification complexes

Serveurs MCP

Les plugins peuvent regrouper des serveurs Model Context Protocol (MCP) pour connecter Claude Code avec des outils et services externes.

Emplacement : `.mcp.json` à la racine du plugin, ou en ligne dans `plugin.json`

Format : Configuration standard du serveur MCP

Configuration du serveur 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}"
    }
  }
}
```

Comportement d'intégration :

- Les serveurs MCP de plugin démarrent automatiquement quand le plugin est activé
- Les serveurs apparaissent comme des outils MCP standard dans la boîte à outils de Claude
- Les capacités du serveur s'intègrent de manière transparente avec les outils existants de Claude
- Les serveurs de plugin peuvent être configurés indépendamment des serveurs MCP de l'utilisateur

Serveurs LSP

Tip:

Vous cherchez à utiliser des plugins LSP ? Installez-les depuis la marketplace officielle : recherchez « lsp » dans l'onglet Découvrir de [/plugin](#) . Cette section documente comment créer des plugins LSP pour les langages non couverts par la marketplace officielle.

Les plugins peuvent fournir des serveurs [Language Server Protocol](#) (LSP) pour donner à Claude une intelligence de code en temps réel lors du travail sur votre base de code.

L'intégration LSP fournit :

- **Diagnostics instantanés** : Claude voit les erreurs et les avertissements immédiatement après chaque modification
- **Navigation de code** : aller à la définition, trouver les références et les informations au survol
- **Sensibilisation au langage** : informations de type et documentation pour les symboles de code

Emplacement : `.lsp.json` à la racine du plugin, ou en ligne dans `plugin.json`

Format : Configuration JSON mappant les noms des serveurs de langage à leurs configurations

Format du fichier `.lsp.json` :

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

En ligne dans `plugin.json` :

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

Champs obligatoires :

Champ	Description
<code>command</code>	Le binaire LSP à exécuter (doit être dans PATH)
<code>extensionToLanguage</code>	Mappe les extensions de fichier aux identifiants de langage

Champs optionnels :

Champ	Description
<code>args</code>	Arguments de ligne de commande pour le serveur LSP
<code>transport</code>	Transport de communication : <code>stdio</code> (par défaut) ou <code>socket</code>
<code>env</code>	Variables d'environnement à définir au démarrage du serveur
<code>initializationOptions</code>	Options transmises au serveur lors de l'initialisation
<code>settings</code>	Paramètres transmis via <code>workspace/didChangeConfiguration</code>
<code>workspaceFolder</code>	Chemin du dossier de l'espace de travail pour le serveur
<code>startupTimeout</code>	Temps maximum d'attente du démarrage du serveur (millisecon- des)
<code>shutdownTimeout</code>	Temps maximum d'attente de l'arrêt gracieux (millisecondes)
<code>restartOnCrash</code>	S'il faut redémarrer automatiquement le serveur en cas de plantage

Champ	Description
<code>maxRestarts</code>	Nombre maximum de tentatives de redémarrage avant d'abandonner

Warning:

Vous devez installer le binaire du serveur de langage séparément. Les plugins LSP configurent comment Claude Code se connecte à un serveur de langage, mais ils n'incluent pas le serveur lui-même. Si vous voyez `Executable not found in $PATH` dans l'onglet Erreurs de `/plugin`, installez le binaire requis pour votre langage.

Plugins LSP disponibles :

Plugin	Serveur de langage	Commande d'installation
<code>pyright-lsp</code>	Pyright (Python)	<code>pip install pyright</code> ou <code>npm install -g pyright</code>
<code>typescript-lsp</code>	TypeScript Language Server	<code>npm install -g typescript-language-server typescript</code>
<code>rust-lsp</code>	rust-analyzer	Voir l'installation de rust-analyzer

Installez d'abord le serveur de langage, puis installez le plugin depuis la marketplace.

Portées d'installation des plugins

Quand vous installez un plugin, vous choisissez une **portée** qui détermine où le plugin est disponible et qui d'autre peut l'utiliser :

Portée	Fichier de paramètres	Cas d'usage
<code>user</code>	<code>~/.claude/settings.json</code>	Plugins personnels disponibles dans tous les projets (par défaut)
<code>project</code>	<code>.claude/settings.json</code>	Plugins d'équipe partagés via le contrôle de version

Portée	Fichier de paramètres	Cas d'usage
local	.claude/settings.local.json	Plugins spécifiques au projet, ignorés par git
managed	Paramètres gérés	Plugins gérés (lecture seule, mise à jour uniquement)

Les plugins utilisent le même système de portée que les autres configurations de Claude Code. Pour les instructions d'installation et les drapeaux de portée, consultez [Installer des plugins](#). Pour une explication complète des portées, consultez [Portées de configuration](#).

Schéma du manifeste du plugin

Le fichier `.claude-plugin/plugin.json` définit les métadonnées et la configuration de votre plugin. Cette section documente tous les champs et options pris en charge.

Le manifeste est optionnel. S'il est omis, Claude Code découvre automatiquement les composants dans les [emplacements par défaut](#) et dérive le nom du plugin du nom du répertoire. Utilisez un manifeste quand vous devez fournir des métadonnées ou des chemins de composants personnalisés.

Schéma complet

```
{
  "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"
}
```

Champs obligatoires

Si vous incluez un manifeste, `name` est le seul champ obligatoire.

Champ	Type	Description	Exemple
<code>name</code>	string	Identifiant unique (kebab-case, pas d'es- paces)	<code>"deployment- tools"</code>

Ce nom est utilisé pour l'espace de noms des composants. Par exemple, dans l'interface utilisateur, l'agent `agent-creator` pour le plugin avec le nom `plugin-dev` apparaîtra comme `plugin-dev:agent-creator`.

Champs de métadonnées

Champ	Type	Description	Exemple
version	string	Version sémantique. Si elle est également définie dans l'entrée de la marketplace, plugin.json a la priorité. Vous n'avez besoin de la définir que dans un seul endroit.	"2.1.0"
description	string	Explication brève de l'objectif du plugin	"Deployment automation tools"
author	object	Informations sur l'auteur	{ "name": "Dev Team", "email": "dev@company.com" }
homepage	string	URL de documentation	"https://docs.example.com"
repository	string	URL du code source	"https://github.com/user/plugin"
license	string	Identifiant de licence	"MIT", "Apache-2.0"
keywords	array	Balises de découverte	["deployment", "ci-cd"]

Champs de chemin de composant

Champ	Type	Description	Exemple
commands	string array	Fichiers/répertoires de commandes supplémentaires	". /custom/cmd.md" ou [". /cmd1.md"]

Champ	Type	Description	Exemple
<code>agents</code>	string array	Fichiers d'agents supplémentaires	<code>"./custom/agents/reviewer.md"</code>
<code>skills</code>	string array	Répertoires de skills supplémentaires	<code>"./custom/skills/"</code>
<code>hooks</code>	string array object	Chemins de configuration des hooks ou configuration en ligne	<code>"./my-extra-hooks.json"</code>
<code>mcpServers</code>	string array object	Chemins de configuration MCP ou configuration en ligne	<code>"./my-extra-mcp-config.json"</code>
<code>outputStyles</code>	string array	Fichiers/répertoires de styles de sortie supplémentaires	<code>"./styles/"</code>
<code>lspServers</code>	string array object	Configurations Language Server Protocol pour l'intelligence de code (aller à la définition, trouver les références, etc.)	<code>"./.lsp.json"</code>

Règles de comportement des chemins

Important : Les chemins personnalisés complètent les répertoires par défaut - ils ne les remplacent pas.

- Si `commands/` existe, il est chargé en plus des chemins de commandes personnalisés
- Tous les chemins doivent être relatifs à la racine du plugin et commencer par `./`
- Les commandes des chemins personnalisés utilisent les mêmes règles de nommage et d'espace de noms
- Plusieurs chemins peuvent être spécifiés sous forme de tableaux pour plus de flexibilité

Exemples de chemins :

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

Variables d'environnement

`${CLAUDE_PLUGIN_ROOT}` : Contient le chemin absolu de votre répertoire de plugin. Utilisez ceci dans les hooks, les serveurs MCP et les scripts pour assurer les chemins corrects indépendamment de l'emplacement d'installation.

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

Mise en cache des plugins et résolution des fichiers

Les plugins sont spécifiés de deux façons :

- Via `claude --plugin-dir`, pour la durée d'une session.
- Via une marketplace, installés pour les sessions futures.

À des fins de sécurité et de vérification, Claude Code copie les plugins de *marketplace* dans le **cache de plugins** local de l'utilisateur (`~/.claude/plugins/cache`) plutôt que de les utiliser sur place. Comprendre ce comportement est important lors du développement de plugins qui référencent des fichiers externes.

Limitations de traversée de répertoires

Les plugins installés ne peuvent pas référencer des fichiers en dehors de leur répertoire. Les chemins qui traversent en dehors de la racine du plugin (comme `../shared-utils`) ne fonctionneront pas après l'installation car ces fichiers externes ne sont pas copiés dans le cache.

Travail avec les dépendances externes

Si votre plugin doit accéder à des fichiers en dehors de son répertoire, vous pouvez créer des liens symboliques vers des fichiers externes dans votre répertoire de plugin. Les liens symboliques sont honorés lors du processus de copie :

```
## À l'intérieur de votre répertoire de plugin
ln -s /path/to/shared-utils ./shared-utils
```

Le contenu lié symboliquement sera copié dans le cache de plugins. Cela offre de la flexibilité tout en maintenant les avantages de sécurité du système de mise en cache.

Structure du répertoire des plugins

Disposition standard des plugins

Un plugin complet suit cette structure :

```

enterprise-plugin/
├── .claude-plugin/          # Répertoire de métadonnées (optionnel)
│   └── plugin.json         # manifeste du plugin
├── commands/               # Emplacement de commande par défaut
│   ├── status.md
│   └── logs.md
├── agents/                 # Emplacement d'agent par défaut
│   ├── security-reviewer.md
│   ├── performance-tester.md
│   └── compliance-checker.md
├── skills/                 # Skills d'agent
│   ├── code-reviewer/
│   │   └── SKILL.md
│   └── pdf-processor/
│       ├── SKILL.md
│       └── scripts/
├── hooks/                  # Configurations des hooks
│   ├── hooks.json          # Configuration principale des hooks
│   └── security-hooks.json # Hooks supplémentaires
├── settings.json           # Paramètres par défaut pour le plugin
├── .mcp.json               # Définitions du serveur MCP
├── .lsp.json               # Configurations du serveur LSP
├── scripts/                # Scripts de hooks et d'utilitaires
│   ├── security-scan.sh
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                 # Fichier de licence
└── CHANGELOG.md            # Historique des versions

```

Warning:

Le répertoire `.claude-plugin/` contient le fichier `plugin.json`. Tous les autres répertoires (commands/, agents/, skills/, hooks/) doivent être à la racine du plugin, pas à l'intérieur de `.claude-plugin/`.

Référence des emplacements de fichiers

Composant	Emplacement par défaut	Objectif
Manifeste	<code>.claude-plugin/plugin.json</code>	Métadonnées et configuration du plugin (optionnel)
Commandes	<code>commands/</code>	Fichiers Markdown de skill (hérité ; utilisez <code>skills/</code> pour les nouveaux skills)
Agents	<code>agents/</code>	Fichiers Markdown de subagent
Skills	<code>skills/</code>	Skills avec structure <code><name>/SKILL.md</code>
Hooks	<code>hooks/hooks.json</code>	Configuration des hooks
Serveurs MCP	<code>.mcp.json</code>	Définitions du serveur MCP
Serveurs LSP	<code>.lsp.json</code>	Configurations du serveur de langage
Paramètres	<code>settings.json</code>	Configuration par défaut appliquée quand le plugin est activé. Seuls les paramètres <code>agent</code> sont actuellement pris en charge

Référence des commandes CLI

Claude Code fournit des commandes CLI pour la gestion des plugins non interactive, utile pour les scripts et l’automatisation.

plugin install

Installez un plugin à partir des marketplaces disponibles.

```
claude plugin install <plugin> [options]
```

Arguments :

- `<plugin>` : Nom du plugin ou `plugin-name@marketplace-name` pour une marketplace spécifique

Options :

Option	Description	Par défaut
<code>-s, --scope <scope></code>	Portée d'installation : <code>user</code> , <code>project</code> , ou <code>local</code>	<code>user</code>
<code>-h, --help</code>	Afficher l'aide pour la commande	

La portée détermine quel fichier de paramètres le plugin installé est ajouté. Par exemple, `--scope project` écrit dans `enabledPlugins` dans `.claude/settings.json`, rendant le plugin disponible à tous ceux qui clonent le référentiel du projet.

Exemples :

```
## Installer dans la portée utilisateur (par défaut)
claude plugin install formatter@my-marketplace

## Installer dans la portée du projet (partagé avec l'équipe)
claude plugin install formatter@my-marketplace --scope project

## Installer dans la portée locale (ignorée par git)
claude plugin install formatter@my-marketplace --scope local
```

plugin uninstall

Supprimez un plugin installé.

```
claude plugin uninstall <plugin> [options]
```

Arguments :

- `<plugin>` : Nom du plugin ou `plugin-name@marketplace-name`

Options :

Option	Description	Par défaut
<code>-s, --scope <scope></code>	Désinstaller de la portée : <code>user</code> , <code>project</code> , ou <code>local</code>	<code>user</code>
<code>-h, --help</code>	Afficher l'aide pour la commande	

Alias : `remove` , `rm`

plugin enable

Activez un plugin désactivé.

```
claude plugin enable <plugin> [options]
```

Arguments :

- `<plugin>` : Nom du plugin ou `plugin-name@marketplace-name`

Options :

Option	Description	Par défaut
<code>-s, --scope <scope></code>	Portée à activer : <code>user</code> , <code>project</code> , ou <code>local</code>	<code>user</code>
<code>-h, --help</code>	Afficher l'aide pour la commande	

plugin disable

Désactivez un plugin sans le désinstaller.

```
claude plugin disable <plugin> [options]
```

Arguments :

- `<plugin>` : Nom du plugin ou `plugin-name@marketplace-name`

Options :

Option	Description	Par défaut
<code>-s, --scope <scope></code>	Portée à désactiver : <code>user</code> , <code>project</code> , ou <code>local</code>	<code>user</code>

Option	Description	Par défaut
<code>-h, --help</code>	Afficher l'aide pour la commande	

plugin update

Mettez à jour un plugin vers la dernière version.

```
claude plugin update <plugin> [options]
```

Arguments :

- `<plugin>` : Nom du plugin ou `plugin-name@marketplace-name`

Options :

Option	Description	Par défaut
<code>-s, --scope <scope></code>	Portée à mettre à jour : <code>user</code> , <code>project</code> , <code>local</code> , ou <code>managed</code>	<code>user</code>
<code>-h, --help</code>	Afficher l'aide pour la commande	

Outils de débogage et de développement

Commandes de débogage

Utilisez `claude --debug` (ou `/debug` dans le TUI) pour voir les détails du chargement des plugins :

Cela affiche :

- Quels plugins sont en cours de chargement
- Toute erreur dans les manifestes de plugins
- Enregistrement des commandes, agents et hooks
- Initialisation du serveur MCP

Problèmes courants

Problème	Cause	Solution
Plugin ne se charge pas	<code>plugin.json</code> invalide	Validez la syntaxe JSON avec <code>claude plugin validate</code> ou <code>/plugin validate</code>
Les commandes n'apparaissent pas	Structure de répertoire incorrecte	Assurez-vous que <code>commands/</code> est à la racine, pas dans <code>.claude-plugin/</code>
Les hooks ne se déclenchent pas	Le script n'est pas exécutable	Exécutez <code>chmod +x script.sh</code>
Le serveur MCP échoue	<code>\${CLAUDE_PLUGIN_ROOT}</code> manquant	Utilisez la variable pour tous les chemins de plugin
Erreurs de chemin	Chemins absolus utilisés	Tous les chemins doivent être relatifs et commencer par <code>./</code>
LSP <code>Executable not found in \$PATH</code>	Serveur de langage non installé	Installez le binaire (par exemple, <code>npm install -g typescript-language-server typescript</code>)

Exemples de messages d'erreur

Erreurs de validation du manifeste :

- `Invalid JSON syntax: Unexpected token }` in JSON at position 142 : vérifiez les virgules manquantes, les virgules supplémentaires ou les chaînes non citées
- `Plugin has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Required` : un champ obligatoire est manquant
- `Plugin has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...` : erreur de syntaxe JSON

Erreurs de chargement du plugin :

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.` : le chemin de commande existe mais ne contient aucun fichier de commande valide
- `Plugin directory not found at path: ./plugins/my-plugin. Check that the marketplace entry has the correct path.` : le chemin `source` dans `marketplace.json` pointe vers un répertoire inexistant
- `Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.` : supprimez les définitions de composants en double ou supprimez `strict: false` dans l'entrée de la marketplace

Dépannage des hooks

Le script du hook ne s'exécute pas :

1. Vérifiez que le script est exécutable : `chmod +x ./scripts/your-script.sh`
2. Vérifiez la ligne shebang : La première ligne doit être `#!/bin/bash` ou `#!/usr/bin/env bash`
3. Vérifiez que le chemin utilise `${CLAUDE_PLUGIN_ROOT}` : `"command": "${CLAUDE_PLUGIN_ROOT}/scripts/your-script.sh"`
4. Testez le script manuellement : `./scripts/your-script.sh`

Le hook ne se déclenche pas sur les événements attendus :

1. Vérifiez que le nom de l'événement est correct (sensible à la casse) : `PostToolUse`, pas `postToolUse`
2. Vérifiez que le motif de correspondance correspond à vos outils : `"matcher": "Write|Edit"` pour les opérations de fichier
3. Confirmez que le type de hook est valide : `command`, `prompt`, ou `agent`

Dépannage du serveur MCP

Le serveur ne démarre pas :

1. Vérifiez que la commande existe et est exécutable
2. Vérifiez que tous les chemins utilisent la variable `${CLAUDE_PLUGIN_ROOT}`
3. Vérifiez les journaux du serveur MCP : `claude --debug` affiche les erreurs d'initialisation
4. Testez le serveur manuellement en dehors de Claude Code

Les outils du serveur n'apparaissent pas :

1. Assurez-vous que le serveur est correctement configuré dans `.mcp.json` ou `plugin.json`
2. Vérifiez que le serveur implémente correctement le protocole MCP
3. Vérifiez les délais d'expiration de la connexion dans la sortie de débogage

Erreurs de structure de répertoire

Symptômes : Le plugin se charge mais les composants (commandes, agents, hooks) sont manquants.

Structure correcte : Les composants doivent être à la racine du plugin, pas à l'intérieur de `.claude-plugin/`. Seul `plugin.json` appartient à `.claude-plugin/`.

```
my-plugin/
├── .claude-plugin/
│   └── plugin.json    ← Seul le manifeste ici
├── commands/         ← Au niveau racine
├── agents/           ← Au niveau racine
└── hooks/            ← Au niveau racine
```

Si vos composants sont à l'intérieur de `.claude-plugin/`, déplacez-les à la racine du plugin.

Liste de contrôle de débogage :

1. Exécutez `claude --debug` et recherchez les messages « loading plugin »
2. Vérifiez que chaque répertoire de composants est listé dans la sortie de débogage
3. Vérifiez que les permissions de fichier permettent de lire les fichiers du plugin

Référence de distribution et de versioning

Gestion des versions

Suivez le versioning sémantique pour les versions de plugin :

```
{
  "name": "my-plugin",
  "version": "2.1.0"
}
```

Format de version : **MAJOR.MINOR.PATCH**

- **MAJOR** : Changements cassants (changements d'API incompatibles)
- **MINOR** : Nouvelles fonctionnalités (ajouts rétro-compatibles)
- **PATCH** : Corrections de bugs (corrections rétro-compatibles)

Meilleures pratiques :

- Commencez à **1.0.0** pour votre première version stable
- Mettez à jour la version dans **plugin.json** avant de distribuer les modifications
- Documentez les modifications dans un fichier **CHANGELOG.md**
- Utilisez des versions de pré-version comme **2.0.0-beta.1** pour les tests

Warning:

Claude Code utilise la version pour déterminer s'il faut mettre à jour votre plugin. Si vous modifiez le code de votre plugin mais ne mettez pas à jour la version dans **plugin.json**, les utilisateurs existants de votre plugin ne verront pas vos modifications en raison de la mise en cache.

Si votre plugin se trouve dans un répertoire de [marketplace](#), vous pouvez gérer la version via **marketplace.json** à la place et omettre le champ **version** de **plugin.json**.

Voir aussi

- [Plugins](#) - Tutoriels et utilisation pratique
- [Marketplaces de plugins](#) - Création et gestion des marketplaces
- [Skills](#) - Détails du développement des skills
- [Subagents](#) - Configuration et capacités des agents
- [Hooks](#) - Gestion des événements et automatisation
- [MCP](#) - Intégration des outils externes
- [Paramètres](#) - Options de configuration pour les plugins

Créer et distribuer une place de marché de plugins

Créez et hébergez des places de marché de plugins pour distribuer les extensions Claude Code dans vos équipes et communautés.

Une **place de marché de plugins** est un catalogue qui vous permet de distribuer des plugins à d'autres. Les places de marché offrent une découverte centralisée, un suivi des versions, des mises à jour automatiques et la prise en charge de plusieurs types de sources (dépôts git, chemins locaux, etc.). Ce guide vous montre comment créer votre propre place de marché pour partager des plugins avec votre équipe ou votre communauté.

Vous cherchez à installer des plugins à partir d'une place de marché existante ? Consultez [Découvrir et installer des plugins préconfigurés](#).

Aperçu

La création et la distribution d'une place de marché impliquent :

1. **Créer des plugins** : créez un ou plusieurs plugins avec des commandes, des agents, des hooks, des serveurs MCP ou des serveurs LSP. Ce guide suppose que vous avez déjà des plugins à distribuer ; consultez [Créer des plugins](#) pour plus de détails sur la création de plugins.
2. **Créer un fichier de place de marché** : définissez un `marketplace.json` qui répertorie vos plugins et où les trouver (voir [Créer le fichier de place de marché](#)).
3. **Héberger la place de marché** : poussez vers GitHub, GitLab ou un autre hôte git (voir [Héberger et distribuer les places de marché](#)).
4. **Partager avec les utilisateurs** : les utilisateurs ajoutent votre place de marché avec `/plugin marketplace add` et installent des plugins individuels (voir [Découvrir et installer des plugins](#)).

Une fois votre place de marché en ligne, vous pouvez la mettre à jour en poussant les modifications vers votre dépôt. Les utilisateurs actualisent leur copie locale avec `/plugin marketplace update`.

Procédure pas à pas : créer une place de marché locale

Cet exemple crée une place de marché avec un plugin : une compétence `/quality-review` pour les révisions de code. Vous allez créer la structure de répertoires, ajouter une compétence, créer le manifeste du plugin et le catalogue de la place de marché, puis l'installer et la tester.

Step 1: Créer la structure de répertoires

```
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: Créer la compétence

Créez un fichier `SKILL.md` qui définit ce que fait la compétence `/quality-review`.

```
---
description: Review code for bugs, security, and performance
disable-model-invocation: true
---

Review the code I've selected or the recent changes for:

- Potential bugs or edge cases
- Security concerns
- Performance issues
- Readability improvements

Be concise and actionable.
```

Step 3: Créer le manifeste du plugin

Créez un fichier `plugin.json` qui décrit le plugin. Le manifeste se trouve dans le répertoire `.claude-plugin/`.

```
{
  "name": "quality-review-plugin",
  "description": "Adds a /quality-review skill for quick code reviews",
  "version": "1.0.0"
}
```

Step 4: Créer le fichier de place de marché

Créez le catalogue de la place de marché qui répertorie votre plugin.

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "Adds a /quality-review skill for quick code reviews"
    }
  ]
}
```

Step 5: Ajouter et installer

Ajoutez la place de marché et installez le plugin.

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

Step 6: Essayer

Sélectionnez du code dans votre éditeur et exécutez votre nouvelle commande.

```
/review
```

Pour en savoir plus sur ce que les plugins peuvent faire, notamment les hooks, les agents, les serveurs MCP et les serveurs LSP, consultez [Plugins](#).

Note:

Comment les plugins sont installés : Lorsque les utilisateurs installent un plugin, Claude Code copie le répertoire du plugin vers un emplacement de cache. Cela signifie que les plugins ne peuvent pas référencer des fichiers en dehors de leur répertoire en utilisant des chemins comme `../shared-utils`, car ces fichiers ne seront pas copiés.

Si vous devez partager des fichiers entre les plugins, utilisez des symlinks (qui sont suivis lors de la copie). Consultez [Plugin caching and file resolution](#) pour plus de détails.

Créer le fichier de place de marché

Créez `.claude-plugin/marketplace.json` à la racine de votre dépôt. Ce fichier définit le nom de votre place de marché, les informations du propriétaire et une liste de plugins avec leurs sources.

Chaque entrée de plugin a besoin au minimum d'un `name` et d'une `source` (où la récupérer). Consultez le [schéma complet](#) ci-dessous pour tous les champs disponibles.


```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "Automatic code formatting on save",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "Deployment automation tools"
    }
  ]
}
```

Schéma de la place de marché

Champs obligatoires

Champ	Type	Description	Exemple
<code>name</code>	string	Identifiant de la place de marché (kebab-case, sans espaces). C'est un élément public : les utilisateurs le voient lors de l'installation de plugins (par exemple, <code>/plugin install my-tool@your-marketplace</code>).	<code>"acme-tools"</code>
<code>owner</code>	object	Informations du responsable de la place de marché (voir les champs ci-dessous)	
<code>plugins</code>	array	Liste des plugins disponibles	Voir ci-dessous

Note:

Noms réservés : Les noms de place de marché suivants sont réservés à l'usage officiel d'Anthropic et ne peuvent pas être utilisés par les places de marché tierces : `claude-code-marketplace`, `claude-code-plugins`, `claude-plugins-official`, `anthropic-marketplace`, `anthropic-plugins`, `agent-skills`, `life-sciences`. Les noms qui usurpent l'identité de places de marché officielles (comme `official-claude-plugins` ou `anthropic-tools-v2`) sont également bloqués.

Champs du propriétaire

Champ	Type	Obligatoire	Description
<code>name</code>	string	Oui	Nom du responsable ou de l'équipe

Champ	Type	Obligatoire	Description
<code>email</code>	string	Non	Adresse e-mail de contact du responsable

Métadonnées optionnelles

Champ	Type	Description
<code>metadata.description</code>	string	Brève description de la place de marché
<code>metadata.version</code>	string	Version de la place de marché
<code>metadata.pluginRoot</code>	string	Répertoire de base ajouté aux chemins de source de plugin relatifs (par exemple, <code>"/plugins"</code> vous permet d'écrire <code>"source": "formatter"</code> au lieu de <code>"source": "./plugins/formatter"</code>)

Entrées de plugin

Chaque entrée de plugin dans le tableau `plugins` décrit un plugin et où le trouver. Vous pouvez inclure n'importe quel champ du [schéma du manifeste du plugin](#) (comme `description`, `version`, `author`, `commands`, `hooks`, etc.), plus ces champs spécifiques à la place de marché : `source`, `category`, `tags` et `strict`.

Champs obligatoires

Champ	Type	Description
<code>name</code>	string	Identifiant du plugin (kebab-case, sans espaces). C'est un élément public : les utilisateurs le voient lors de l'installation (par exemple, <code>/plugin install my-plugin@marketplace</code>).

Champ	Type	Description
<code>source</code>	string object	Où récupérer le plugin (voir Sources de plugin ci-dessous)

Champs de plugin optionnels

Champs de métadonnées standard :

Champ	Type	Description
<code>description</code>	string	Brève description du plugin
<code>version</code>	string	Version du plugin
<code>author</code>	object	Informations sur l'auteur du plugin (<code>name</code> obligatoire, <code>email</code> optionnel)
<code>homepage</code>	string	URL de la page d'accueil ou de la documentation du plugin
<code>repository</code>	string	URL du dépôt du code source
<code>license</code>	string	Identifiant de licence SPDX (par exemple, MIT, Apache-2.0)
<code>keywords</code>	array	Balises pour la découverte et la catégorisation des plugins
<code>category</code>	string	Catégorie du plugin pour l'organisation
<code>tags</code>	array	Balises pour la recherche
<code>strict</code>	boolean	Contrôle si <code>plugin.json</code> est l'autorité pour les définitions de composants (par défaut : true). Voir Mode strict ci-dessous.

Champs de configuration des composants :

Champ	Type	Description
<code>commands</code>	string array	Chemins personnalisés vers les fichiers ou répertoires de commandes
<code>agents</code>	string array	Chemins personnalisés vers les fichiers d'agents
<code>hooks</code>	string object	Configuration personnalisée des hooks ou chemin vers le fichier des hooks
<code>mcpServers</code>	string object	Configurations du serveur MCP ou chemin vers la configuration MCP
<code>lspServers</code>	string object	Configurations du serveur LSP ou chemin vers la configuration LSP

Sources de plugin

Les sources de plugin indiquent à Claude Code où récupérer chaque plugin individuel répertorié dans votre place de marché. Elles sont définies dans le champ `source` de chaque entrée de plugin dans `marketplace.json`.

Une fois qu'un plugin est cloné ou copié sur la machine locale, il est copié dans le cache de plugin local versionné à `~/.claude/plugins/cache`.

Source	Type	Champs	Notes
Chemin relatif	<code>string</code> (par exemple <code>"./my-plugin"</code>)	—	Répertoire local dans le dépôt de la place de marché. Doit commencer par <code>./</code>
<code>github</code>	object	<code>repo</code> , <code>ref?</code> , <code>sha?</code>	
<code>url</code>	object	<code>url</code> (doit se terminer par <code>.git</code>) , <code>ref?</code> , <code>sha?</code>	Source d'URL Git

Source	Type	Champs	Notes
<code>git-subdir</code>	object	<code>url</code> , <code>path</code> , <code>ref?</code> , <code>sha?</code>	Sous-répertoire dans un dépôt git. Clone partiellement pour minimiser la bande passante pour les monodépôts
<code>npm</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Installé via <code>npm install</code>
<code>pip</code>	object	<code>package</code> , <code>version?</code> , <code>registry?</code>	Installé via pip

Note:

Sources de place de marché vs sources de plugin : Ce sont des concepts différents qui contrôlent des choses différentes.

- **Source de place de marché** — où récupérer le catalogue `marketplace.json` lui-même. Défini lorsque les utilisateurs exécutent `/plugin marketplace add` ou dans les paramètres `extraKnownMarketplaces`. Prend en charge `ref` (branche/tag) mais pas `sha`.
- **Source de plugin** — où récupérer un plugin individuel répertorié dans la place de marché. Défini dans le champ `source` de chaque entrée de plugin dans `marketplace.json`. Prend en charge à la fois `ref` (branche/tag) et `sha` (commit exact).

Par exemple, une place de marché hébergée à `acme-corp/plugin-catalog` (source de place de marché) peut répertorier un plugin récupéré à partir de `acme-corp/code-formatter` (source de plugin). La source de place de marché et la source de plugin pointent vers des dépôts différents et sont épinglées indépendamment.

Chemins relatifs

Pour les plugins dans le même dépôt :

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

Note:

Les chemins relatifs ne fonctionnent que lorsque les utilisateurs ajoutent votre place de marché via Git (GitHub, GitLab ou URL git). Si les utilisateurs ajoutent votre place de marché via une URL directe vers le fichier `marketplace.json`, les chemins relatifs ne se résoudre pas correctement. Pour la distribution basée sur les URL, utilisez plutôt les sources GitHub, npm ou URL git. Consultez [Dépannage](#) pour plus de détails.

Dépôts GitHub

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

Vous pouvez épingler à une branche, un tag ou un commit spécifique :

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

Champ	Type	Description
<code>repo</code>	string	Obligatoire. Dépôt GitHub au format <code>owner/repo</code>
<code>ref</code>	string	Optionnel. Branche ou tag Git (par défaut la branche par défaut du dépôt)
<code>sha</code>	string	Optionnel. SHA de commit git complet de 40 caractères pour épingler à une version exacte

Dépôts Git

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

Vous pouvez épingler à une branche, un tag ou un commit spécifique :

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```


Champ	Type	Description
<code>url</code>	string	Obligatoire. URL complète du dépôt git (doit se terminer par <code>.git</code>)
<code>ref</code>	string	Optionnel. Branche ou tag Git (par défaut la branche par défaut du dépôt)
<code>sha</code>	string	Optionnel. SHA de commit git complet de 40 caractères pour épingler à une version exacte

Sous-répertoires Git

Utilisez `git-subdir` pour pointer vers un plugin qui se trouve dans un sous-répertoire d'un dépôt git. Claude Code utilise un clone partiel et clairsemé pour récupérer uniquement le sous-répertoire, minimisant la bande passante pour les grands monodépôts.

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

Vous pouvez épingler à une branche, un tag ou un commit spécifique :

```
{
  "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"
  }
}
```

Le champ `url` accepte également un raccourci GitHub (`owner/repo`) ou des URL SSH (`git@github.com:owner/repo.git`).

Champ	Type	Description
<code>url</code>	string	Obligatoire. URL du dépôt Git, raccourci GitHub <code>owner/repo</code> ou URL SSH
<code>path</code>	string	Obligatoire. Chemin du sous-répertoire dans le dépôt contenant le plugin (par exemple, <code>"tools/claude-plugin"</code>)
<code>ref</code>	string	Optionnel. Branche ou tag Git (par défaut la branche par défaut du dépôt)
<code>sha</code>	string	Optionnel. SHA de commit git complet de 40 caractères pour épingler à une version exacte

Paquets npm

Les plugins distribués en tant que paquets npm sont installés à l'aide de `npm install` . Cela fonctionne avec n'importe quel paquet du registre npm public ou d'un registre privé que votre équipe héberge.

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

Pour épingler à une version spécifique, ajoutez le champ `version` :

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

Pour installer à partir d'un registre privé ou interne, ajoutez le champ `registry` :

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

Champ	Type	Description
<code>package</code>	string	Obligatoire. Nom du paquet ou paquet scopé (par exemple, <code>@org/plugin</code>)

Champ	Type	Description
version	string	Optionnel. Version ou plage de version (par exemple, 2.1.0 , ^2.0.0 , ~1.5.0)
registry	string	Optionnel. URL du registre npm personnalisé. Par défaut le registre npm du système (généralement npm-js.org)

Entrées de plugin avancées

Cet exemple montre une entrée de plugin utilisant de nombreux champs optionnels, notamment des chemins personnalisés pour les commandes, les agents, les hooks et les serveurs MCP :

```

{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "Enterprise workflow automation tools",
  "version": "2.1.0",
  "author": {
    "name": "Enterprise Team",
    "email": "enterprise@example.com"
  },
  "homepage": "https://docs.example.com/plugins/enterprise-tools",
  "repository": "https://github.com/company/enterprise-plugin",
  "license": "MIT",
  "keywords": ["enterprise", "workflow", "automation"],
  "category": "productivity",
  "commands": [
    "./commands/core/",
    "./commands/enterprise/",
    "./commands/experimental/preview.md"
  ],
  "agents": ["/agents/security-reviewer.md", "/agents/compliance-checker.md"],
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/validate.sh"
          }
        ]
      }
    ]
  },
  "mcpServers": {
    "enterprise-db": {

```

```

    "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
    "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"]
  }
},
"strict": false
}

```

Points clés à noter :

- **commands et agents** : Vous pouvez spécifier plusieurs répertoires ou fichiers individuels. Les chemins sont relatifs à la racine du plugin.
- **\${CLAUDE_PLUGIN_ROOT}** : Utilisez cette variable dans les hooks et les configurations du serveur MCP pour référencer les fichiers dans le répertoire d'installation du plugin. C'est nécessaire car les plugins sont copiés vers un emplacement de cache lors de l'installation.
- **strict: false** : Puisque ceci est défini sur false, le plugin n'a pas besoin de son propre `plugin.json`. L'entrée de la place de marché définit tout. Voir [Mode strict](#) ci-dessous.

Mode strict

Le champ `strict` contrôle si `plugin.json` est l'autorité pour les définitions de composants (commandes, agents, hooks, compétences, serveurs MCP, styles de sortie).

Valeur	Comportement
<code>true</code> (par défaut)	<code>plugin.json</code> est l'autorité. L'entrée de la place de marché peut la compléter avec des composants supplémentaires, et les deux sources sont fusionnées.
<code>false</code>	L'entrée de la place de marché est la définition complète. Si le plugin a également un <code>plugin.json</code> qui déclare des composants, c'est un conflit et le plugin ne se charge pas.

Quand utiliser chaque mode :

- **strict: true** : le plugin a son propre `plugin.json` et gère ses propres composants. L'entrée de la place de marché peut ajouter des commandes ou des hooks supplémentaires par-dessus. C'est la valeur par défaut et fonctionne pour la plupart des plugins.
- **strict: false** : l'opérateur de la place de marché veut le contrôle total. Le dépôt du plugin fournit des fichiers bruts, et l'entrée de la place de marché définit lesquels de

ces fichiers sont exposés en tant que commandes, agents, hooks, etc. Utile lorsque la place de marché restructure ou sélectionne les composants d'un plugin différemment de ce que l'auteur du plugin avait prévu.

Héberger et distribuer les places de marché

Héberger sur GitHub (recommandé)

GitHub offre la méthode de distribution la plus facile :

- 1. **Créer un dépôt** : Configurez un nouveau dépôt pour votre place de marché
- 2. **Ajouter le fichier de place de marché** : Créez `.claude-plugin/marketplace.json` avec vos définitions de plugins
- 3. **Partager avec les équipes** : Les utilisateurs ajoutent votre place de marché avec `/plugin marketplace add owner/repo`

Avantages : Contrôle de version intégré, suivi des problèmes et fonctionnalités de collaboration d'équipe.

Héberger sur d'autres services git

N'importe quel service d'hébergement git fonctionne, comme GitLab, Bitbucket et les serveurs auto-hébergés. Les utilisateurs ajoutent avec l'URL complète du dépôt :

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Dépôts privés

Claude Code prend en charge l'installation de plugins à partir de dépôts privés. Pour l'installation manuelle et les mises à jour, Claude Code utilise vos assistants de credentials git existants. Si `git clone` fonctionne pour un dépôt privé dans votre terminal, cela fonctionne aussi dans Claude Code. Les assistants de credentials courants incluent `gh auth login` pour GitHub, Keychain macOS et `git-credential-store`.

Les mises à jour automatiques en arrière-plan s'exécutent au démarrage sans assistants de credentials, car les invites interactives bloqueraient le démarrage de Claude Code. Pour activer les mises à jour automatiques pour les places de marché privées, définissez le jeton d'authentification approprié dans votre environnement :

Fournisseur	Variables d'environnement	Notes
GitHub	<code>GITHUB_TOKEN</code> ou <code>GH_TOKEN</code>	Jeton d'accès personnel ou jeton GitHub App

Fournisseur	Variables d'environnement	Notes
GitLab	<code>GITLAB_TOKEN</code> ou <code>GL_TOKEN</code>	Jeton d'accès personnel ou jeton de projet
Bitbucket	<code>BITBUCKET_TOKEN</code>	Mot de passe d'application ou jeton d'accès au dépôt

Définissez le jeton dans votre configuration de shell (par exemple, `.bashrc`, `.zshrc`) ou passez-le lors de l'exécution de Claude Code :

```
export GITHUB_TOKEN=ghp_XXXXXXXXXXXXXXXXXXXX
```

Note:

Pour les environnements CI/CD, configurez le jeton en tant que variable d'environnement secrète. GitHub Actions fournit automatiquement `GITHUB_TOKEN` pour les dépôts de la même organisation.

Tester localement avant la distribution

Testez votre place de marché localement avant de la partager :

```
/plugin marketplace add ./my-local-marketplace
/plugin install test-plugin@my-local-marketplace
```

Pour la gamme complète de commandes add (GitHub, URL Git, chemins locaux, URL distantes), consultez [Ajouter des places de marché](#).

Exiger des places de marché pour votre équipe

Vous pouvez configurer votre dépôt pour que les membres de l'équipe soient automatiquement invités à installer votre place de marché lorsqu'ils font confiance au dossier du projet. Ajoutez votre place de marché à `.claude/settings.json` :


```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Vous pouvez également spécifier quels plugins doivent être activés par défaut :

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

Pour les options de configuration complètes, consultez [Paramètres des plugins](#).

Restrictions des places de marché gérées

Pour les organisations nécessitant un contrôle strict sur les sources de plugins, les administrateurs peuvent restreindre les places de marché de plugins que les utilisateurs sont autorisés à ajouter en utilisant le paramètre `strictKnownMarketplaces` dans les paramètres gérés.

Lorsque `strictKnownMarketplaces` est configuré dans les paramètres gérés, le comportement de restriction dépend de la valeur :

Valeur	Comportement
Non défini (par défaut)	Aucune restriction. Les utilisateurs peuvent ajouter n'importe quelle place de marché
Tableau vide <code>[]</code>	Verrouillage complet. Les utilisateurs ne peuvent pas ajouter de nouvelles places de marché

Valeur	Comportement
Liste de sources	Les utilisateurs ne peuvent ajouter que les places de marché qui correspondent exactement à la liste d'autorisation

Configurations courantes

Désactiver tous les ajouts de place de marché :

```
{  
  "strictKnownMarketplaces": []  
}
```

Autoriser uniquement les places de marché spécifiques :

```
{  
  "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"  
    }  
  ]  
}
```

Autoriser toutes les places de marché d'un serveur git interne en utilisant la correspondance de motif regex sur l'hôte :

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

Autoriser les places de marché basées sur le système de fichiers à partir d'un répertoire spécifique en utilisant la correspondance de motif regex sur le chemin :

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

Utilisez `".*"` comme `pathPattern` pour autoriser n'importe quel chemin du système de fichiers tout en contrôlant les sources réseau avec `hostPattern`.

Comment fonctionnent les restrictions

Les restrictions sont validées tôt dans le processus d'installation du plugin, avant toute demande réseau ou opération du système de fichiers. Cela empêche les tentatives d'accès non autorisé à la place de marché.

La liste d'autorisation utilise la correspondance exacte pour la plupart des types de sources. Pour qu'une place de marché soit autorisée, tous les champs spécifiés doivent correspondre exactement :

- Pour les sources GitHub : `repo` est obligatoire, et `ref` ou `path` doivent également correspondre s'ils sont spécifiés dans la liste d'autorisation
- Pour les sources URL : l'URL complète doit correspondre exactement
- Pour les sources `hostPattern` : l'hôte de la place de marché est comparé au motif regex

- Pour les sources `pathPattern` : le chemin du système de fichiers de la place de marché est comparé au motif regex

Parce que `strictKnownMarketplaces` est défini dans les [paramètres gérés](#), les configurations individuelles des utilisateurs et des projets ne peuvent pas contourner ces restrictions.

Pour les détails de configuration complets, y compris tous les types de sources pris en charge et la comparaison avec `extraKnownMarketplaces`, consultez la [référence strictKnownMarketplaces](#).

Résolution des versions et canaux de publication

Les versions des plugins déterminent les chemins du cache et la détection des mises à jour. Vous pouvez spécifier la version dans le manifeste du plugin (`plugin.json`) ou dans l'entrée de la place de marché (`marketplace.json`).

Warning:

Lorsque c'est possible, évitez de définir la version aux deux endroits. Le manifeste du plugin gagne toujours silencieusement, ce qui peut faire que la version de la place de marché soit ignorée. Pour les plugins avec chemins relatifs, définissez la version dans l'entrée de la place de marché. Pour toutes les autres sources de plugins, définissez-la dans le manifeste du plugin.

Configurer les canaux de publication

Pour prendre en charge les canaux de publication « stable » et « latest » pour vos plugins, vous pouvez configurer deux places de marché qui pointent vers différentes refs ou SHAs du même dépôt. Vous pouvez ensuite assigner les deux places de marché à différents groupes d'utilisateurs via les [paramètres gérés](#).

Warning:

Le `plugin.json` du plugin doit déclarer une `version` différente à chaque ref ou commit épinglé. Si deux refs ou commits ont la même version de manifeste, Claude Code les traite comme identiques et ignore la mise à jour.

Exemple

```
{
  "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"
      }
    }
  ]
}
```

Assigner les canaux aux groupes d'utilisateurs

Assignez chaque place de marché au groupe d'utilisateurs approprié via les paramètres gérés. Par exemple, le groupe stable reçoit :

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

Le groupe early-access reçoit **latest-tools** à la place :

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

Validation et test

Testez votre place de marché avant de la partager.

Validez la syntaxe JSON de votre place de marché :

```
claude plugin validate .
```

Ou depuis Claude Code :

```
/plugin validate .
```

Ajoutez la place de marché pour le test :

```
/plugin marketplace add ./path/to/marketplace
```

Installez un plugin de test pour vérifier que tout fonctionne :

```
/plugin install test-plugin@marketplace-name
```

Pour les flux de travail complets de test de plugins, consultez [Tester vos plugins localement](#). Pour le dépannage technique, consultez [Référence des plugins](#).

Dépannage

La place de marché ne se charge pas

Symptômes : Impossible d'ajouter la place de marché ou de voir les plugins qu'elle contient

Solutions :

- Vérifiez que l'URL de la place de marché est accessible
- Vérifiez que `.claude-plugin/marketplace.json` existe au chemin spécifié
- Assurez-vous que la syntaxe JSON est valide en utilisant `claude plugin validate` ou `/plugin validate`
- Pour les dépôts privés, confirmez que vous avez les permissions d'accès

Erreurs de validation de la place de marché

Exécutez `claude plugin validate .` ou `/plugin validate .` à partir de votre répertoire de place de marché pour vérifier les problèmes. Erreurs courantes :

Erreur	Cause	Solution
<code>File not found: .claude-plugin/marketplace.json</code>	Manifeste manquant	Créez <code>.claude-plugin/marketplace.json</code> avec les champs obligatoires
<code>Invalid JSON syntax: Unexpected token...</code>	Erreur de syntaxe JSON	Vérifiez les virgules manquantes, les virgules supplémentaires ou les chaînes non citées

Erreur	Cause	Solution
<code>Duplicate plugin name "x" found in marketplace</code>	Deux plugins partagent le même nom	Donnez à chaque plugin une valeur <code>name</code> unique
<code>plugins[0].source: Path traversal not allowed</code>	Le chemin source contient <code>..</code>	Utilisez des chemins relatifs à la racine de la place de marché sans <code>..</code>

Avertissements (non bloquants) :

- `Marketplace has no plugins defined` : ajoutez au moins un plugin au tableau `plugins`
- `No marketplace description provided` : ajoutez `metadata.description` pour aider les utilisateurs à comprendre votre place de marché

Échecs d'installation de plugins

Symptômes : La place de marché apparaît mais l'installation du plugin échoue

Solutions :

- Vérifiez que les URL sources des plugins sont accessibles
- Vérifiez que les répertoires des plugins contiennent les fichiers requis
- Pour les sources GitHub, assurez-vous que les dépôts sont publics ou que vous avez accès
- Testez manuellement les sources de plugins en les clonant/téléchargeant

L'authentification du dépôt privé échoue

Symptômes : Erreurs d'authentification lors de l'installation de plugins à partir de dépôts privés

Solutions :

Pour l'installation manuelle et les mises à jour :

- Vérifiez que vous êtes authentifié auprès de votre fournisseur git (par exemple, exécutez `gh auth status` pour GitHub)
- Vérifiez que votre assistant de credentials est configuré correctement : `git config --global credential.helper`
- Essayez de cloner le dépôt manuellement pour vérifier que vos credentials fonctionnent

Pour les mises à jour automatiques en arrière-plan :

- Définissez le jeton approprié dans votre environnement : `echo $GITHUB_TOKEN`
- Vérifiez que le jeton a les permissions requises (accès en lecture au dépôt)
- Pour GitHub, assurez-vous que le jeton a la portée `repo` pour les dépôts privés
- Pour GitLab, assurez-vous que le jeton a au moins la portée `read_repository`
- Vérifiez que le jeton n'a pas expiré

Les opérations Git expirent

Symptômes : L'installation du plugin ou les mises à jour de la place de marché échouent avec une erreur de délai d'expiration comme « Git clone timed out after 120s » ou « Git pull timed out after 120s ».

Cause : Claude Code utilise un délai d'expiration de 120 secondes pour toutes les opérations git, y compris le clonage des dépôts de plugins et l'extraction des mises à jour de la place de marché. Les grands dépôts ou les connexions réseau lentes peuvent dépasser cette limite.

Solution : Augmentez le délai d'expiration en utilisant la variable d'environnement `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS`. La valeur est en millisecondes :

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000 # 5 minutes
```

Les plugins avec chemins relatifs échouent dans les places de marché basées sur les URL

Symptômes : Vous avez ajouté une place de marché via URL (comme `https://example.com/marketplace.json`), mais les plugins avec des sources de chemin relatif comme `"/plugins/my-plugin"` échouent à installer avec des erreurs « path not found ».

Cause : Les places de marché basées sur les URL téléchargent uniquement le fichier `marketplace.json` lui-même. Elles ne téléchargent pas les fichiers de plugins du serveur. Les chemins relatifs dans l'entrée de la place de marché référencent des fichiers sur le serveur distant qui n'ont pas été téléchargés.

Solutions :

- **Utiliser des sources externes** : Changez les entrées de plugins pour utiliser les sources GitHub, npm ou URL git au lieu des chemins relatifs :

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Utiliser une place de marché basée sur Git** : Hébergez votre place de marché dans un dépôt Git et ajoutez-la avec l'URL git. Les places de marché basées sur Git clonent le dépôt entier, ce qui rend les chemins relatifs fonctionnels.

Fichiers non trouvés après l'installation

Symptômes : Le plugin s'installe mais les références aux fichiers échouent, en particulier les fichiers en dehors du répertoire du plugin

Cause : Les plugins sont copiés vers un répertoire de cache plutôt que d'être utilisés sur place. Les chemins qui référencent des fichiers en dehors du répertoire du plugin (comme `../shared-utils`) ne fonctionneront pas car ces fichiers ne sont pas copiés.

Solutions : Consultez [Plugin caching and file resolution](#) pour les solutions de contournement, y compris les symlinks et la restructuration des répertoires.

Pour des outils de débogage supplémentaires et des problèmes courants, consultez [Debugging and development tools](#).

Voir aussi

- [Découvrir et installer des plugins préconfigurés](#) - Installation de plugins à partir de places de marché existantes
- [Plugins](#) - Création de vos propres plugins
- [Référence des plugins](#) - Spécifications techniques complètes et schémas
- [Paramètres des plugins](#) - Options de configuration des plugins
- [Référence strictKnownMarketplaces](#) - Restrictions des places de marché gérées

Découvrir et installer des plugins prédéfinis via les marketplaces

Trouvez et installez des plugins depuis les marketplaces pour étendre Claude Code avec de nouvelles commandes, agents et capacités.

Les plugins étendent Claude Code avec des skills, des agents, des hooks et des MCP servers. Les marketplaces de plugins sont des catalogues qui vous aident à découvrir et installer ces extensions sans les construire vous-même.

Vous cherchez à créer et distribuer votre propre marketplace ? Consultez [Créer et distribuer une marketplace de plugins](#).

Comment fonctionnent les marketplaces

Une marketplace est un catalogue de plugins que quelqu'un d'autre a créé et partagé. L'utilisation d'une marketplace est un processus en deux étapes :

Step 1: Ajouter la marketplace

Cela enregistre le catalogue avec Claude Code pour que vous puissiez parcourir ce qui est disponible. Aucun plugin n'est installé pour le moment.

Step 2: Installer des plugins individuels

Parcourez le catalogue et installez les plugins que vous souhaitez.

Pensez-y comme ajouter un app store : ajouter le store vous donne accès pour parcourir sa collection, mais vous choisissez toujours quelles applications télécharger individuellement.

Marketplace officielle Anthropic

La marketplace officielle Anthropic (`claude-plugins-official`) est automatiquement disponible quand vous démarrez Claude Code. Exécutez `/plugin` et allez à l'onglet **Discover** pour parcourir ce qui est disponible.

Pour installer un plugin depuis la marketplace officielle :

```
/plugin install plugin-name@claude-plugins-official
```

Note:

La marketplace officielle est maintenue par Anthropic. Pour soumettre un plugin à la marketplace officielle, utilisez l'un des formulaires de soumission intégrés à l'application :

- **Claude.ai** : claude.ai/settings/plugins/submit
- **Console** : platform.claude.com/plugins/submit

Pour distribuer des plugins indépendamment, [créez votre propre marketplace](#) et partagez-la avec les utilisateurs.

La marketplace officielle inclut plusieurs catégories de plugins :

Code intelligence

Les plugins de code intelligence activent l'outil LSP intégré de Claude Code, donnant à Claude la capacité de sauter aux définitions, trouver les références et voir les erreurs de type immédiatement après les modifications. Ces plugins configurent les connexions [Language Server Protocol](#), la même technologie qui alimente la code intelligence de VS Code.

Ces plugins nécessitent que le binaire du serveur de langage soit installé sur votre système. Si vous avez déjà un serveur de langage installé, Claude peut vous inviter à installer le plugin correspondant quand vous ouvrez un projet.

Langage	Plugin	Binaire requis
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

Language	Plugin	Binaire requis
Swift	<code>swift-lsp</code>	<code>sourcekit-lsp</code>
TypeScript	<code>typescript-lsp</code>	<code>typescript-language-server</code>

Vous pouvez également [créer votre propre plugin LSP](#) pour d'autres langages.

Note:

Si vous voyez `Executable not found in $PATH` dans l'onglet `/plugin` Errors après avoir installé un plugin, installez le binaire requis du tableau ci-dessus.

Ce que Claude gagne des plugins de code intelligence

Une fois qu'un plugin de code intelligence est installé et que son binaire de serveur de langage est disponible, Claude gagne deux capacités :

- **Diagnostics automatiques** : après chaque modification de fichier que Claude effectue, le serveur de langage analyse les modifications et signale les erreurs et avertissements automatiquement. Claude voit les erreurs de type, les imports manquants et les problèmes de syntaxe sans avoir besoin d'exécuter un compilateur ou un linter. Si Claude introduit une erreur, il la remarque et la corrige dans le même tour. Cela ne nécessite aucune configuration au-delà de l'installation du plugin. Vous pouvez voir les diagnostics en ligne en appuyant sur **Ctrl+O** quand l'indicateur « diagnostics found » apparaît.
- **Navigation de code** : Claude peut utiliser le serveur de langage pour sauter aux définitions, trouver les références, obtenir les informations de type au survol, lister les symboles, trouver les implémentations et tracer les hiérarchies d'appels. Ces opérations donnent à Claude une navigation plus précise que la recherche basée sur grep, bien que la disponibilité puisse varier selon le langage et l'environnement.

Si vous rencontrez des problèmes, consultez [Dépannage de la code intelligence](#).

Intégrations externes

Ces plugins regroupent des [MCP servers](#) préconfigurés pour que vous puissiez connecter Claude à des services externes sans configuration manuelle :

- **Contrôle de source** : `github`, `gitlab`
- **Gestion de projet** : `atlassian` (Jira/Confluence), `asana`, `linear`, `notion`
- **Design** : `figma`

- **Infrastructure** : `vercel`, `firebase`, `supabase`
- **Communication** : `slack`
- **Monitoring** : `sentry`

Workflows de développement

Plugins qui ajoutent des commandes et des agents pour les tâches de développement courantes :

- **commit-commands** : Workflows de commit Git incluant commit, push et création de PR
- **pr-review-toolkit** : Agents spécialisés pour examiner les pull requests
- **agent-sdk-dev** : Outils pour construire avec le Claude Agent SDK
- **plugin-dev** : Toolkit pour créer vos propres plugins

Styles de sortie

Personnalisez comment Claude répond :

- **explanatory-output-style** : Insights éducatifs sur les choix d'implémentation
- **learning-output-style** : Mode d'apprentissage interactif pour la construction de compétences

Essayez : ajouter la marketplace de démonstration

Anthropic maintient également une [marketplace de plugins de démonstration](#) (`claude-code-plugins`) avec des plugins d'exemple qui montrent ce qui est possible avec le système de plugins. Contrairement à la marketplace officielle, vous devez ajouter celle-ci manuellement.

Step 1: Ajouter la marketplace

Depuis Claude Code, exécutez la commande `plugin marketplace add` pour la marketplace `anthropics/claude-code` :

```
/plugin marketplace add anthropics/claude-code
```

Cela télécharge le catalogue de la marketplace et rend ses plugins disponibles pour vous.

Step 2: Parcourir les plugins disponibles

Exécutez `/plugin` pour ouvrir le gestionnaire de plugins. Cela ouvre une interface à onglets avec quatre onglets que vous pouvez parcourir en utilisant **Tab** (ou **Shift+Tab** pour aller en arrière) :

- **Discover** : parcourez les plugins disponibles de toutes vos marketplaces
- **Installed** : visualisez et gérez vos plugins installés
- **Marketplaces** : ajoutez, supprimez ou mettez à jour vos marketplaces ajoutées
- **Errors** : visualisez les erreurs de chargement de plugins

Allez à l'onglet **Discover** pour voir les plugins de la marketplace que vous venez d'ajouter.

Step 3: Installer un plugin

Sélectionnez un plugin pour voir ses détails, puis choisissez une portée d'installation :

- **User scope** : installez pour vous-même dans tous les projets
- **Project scope** : installez pour tous les collaborateurs sur ce référentiel
- **Local scope** : installez pour vous-même dans ce référentiel uniquement

Par exemple, sélectionnez **commit-commands** (un plugin qui ajoute des commandes de workflow git) et installez-le à votre portée utilisateur.

Vous pouvez également installer directement depuis la ligne de commande :

```
/plugin install commit-commands@anthropics-claude-code
```

Consultez [Configuration scopes](#) pour en savoir plus sur les portées.

Step 4: Utiliser votre nouveau plugin

Après l'installation, les commandes du plugin sont immédiatement disponibles. Les commandes de plugin sont espacées par le nom du plugin, donc **commit-commands** fournit des commandes comme `/commit-commands:commit`.

Essayez en effectuant une modification à un fichier et en exécutant :

```
/commit-commands:commit
```

Cela prépare vos modifications, génère un message de commit et crée le commit.

Chaque plugin fonctionne différemment. Consultez la description du plugin dans l'onglet **Discover** ou sa page d'accueil pour apprendre quelles commandes et capacités il fournit.

Le reste de ce guide couvre tous les moyens d'ajouter des marketplaces, installer des plugins et gérer votre configuration.

Ajouter des marketplaces

Utilisez la commande `/plugin marketplace add` pour ajouter des marketplaces de différentes sources.

Tip:

Raccourcis : Vous pouvez utiliser `/plugin market` au lieu de `/plugin marketplace`, et `rm` au lieu de `remove`.

- **Référentiels GitHub** : format `owner/repo` (par exemple, `anthropics/claude-code`)
- **URLs Git** : n'importe quelle URL de référentiel git (GitLab, Bitbucket, auto-hébergé)
- **Chemins locaux** : répertoires ou chemins directs vers les fichiers `marketplace.json`
- **URLs distantes** : URLs directs vers les fichiers `marketplace.json` hébergés

Ajouter depuis GitHub

Ajoutez un référentiel GitHub qui contient un fichier `.claude-plugin/marketplace.json` en utilisant le format `owner/repo` —où `owner` est le nom d'utilisateur ou l'organisation GitHub et `repo` est le nom du référentiel.

Par exemple, `anthropics/claude-code` fait référence au référentiel `claude-code` appartenant à `anthropics` :

```
/plugin marketplace add anthropics/claude-code
```

Ajouter depuis d'autres hôtes Git

Ajoutez n'importe quel référentiel git en fournissant l'URL complète. Cela fonctionne avec n'importe quel hôte Git, y compris GitLab, Bitbucket et les serveurs auto-hébergés :

Utilisation de HTTPS :

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

Utilisation de SSH :


```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

Pour ajouter une branche ou un tag spécifique, ajoutez **#** suivi de la ref :

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

Ajouter depuis des chemins locaux

Ajoutez un répertoire local qui contient un fichier **.claude-plugin/marketplace.json** :

```
/plugin marketplace add ./my-marketplace
```

Vous pouvez également ajouter un chemin direct vers un fichier **marketplace.json** :

```
/plugin marketplace add ./path/to/marketplace.json
```

Ajouter depuis des URLs distantes

Ajoutez un fichier **marketplace.json** distant via URL :

```
/plugin marketplace add https://example.com/marketplace.json
```

Note:

Les marketplaces basées sur URL ont certaines limitations par rapport aux marketplaces basées sur Git. Si vous rencontrez des erreurs « path not found » lors de l'installation de plugins, consultez [Dépannage](#).

Installer des plugins

Une fois que vous avez ajouté des marketplaces, vous pouvez installer des plugins directement (installe à la portée utilisateur par défaut) :

```
/plugin install plugin-name@marketplace-name
```

Pour choisir une [portée d'installation](#) différente, utilisez l'interface interactive : exécutez `/plugin`, allez à l'onglet **Discover** et appuyez sur **Enter** sur un plugin. Vous verrez les options pour :

- **User scope** (par défaut) : installez pour vous-même dans tous les projets
- **Project scope** : installez pour tous les collaborateurs sur ce référentiel (ajoute à `.claude/settings.json`)
- **Local scope** : installez pour vous-même dans ce référentiel uniquement (non partagé avec les collaborateurs)

Vous pouvez également voir des plugins avec la portée **managed**—ceux-ci sont installés par les administrateurs via [managed settings](#) et ne peuvent pas être modifiés.

Exécutez `/plugin` et allez à l'onglet **Installed** pour voir vos plugins groupés par portée.

Warning:

Assurez-vous de faire confiance à un plugin avant de l'installer. Anthropic ne contrôle pas quels MCP servers, fichiers ou autres logiciels sont inclus dans les plugins et ne peut pas vérifier qu'ils fonctionnent comme prévu. Consultez la page d'accueil de chaque plugin pour plus d'informations.

Gérer les plugins installés

Exécutez `/plugin` et allez à l'onglet **Installed** pour visualiser, activer, désactiver ou désinstaller vos plugins. Tapez pour filtrer la liste par nom ou description du plugin.

Vous pouvez également gérer les plugins avec des commandes directes.

Désactiver un plugin sans le désinstaller :

```
/plugin disable plugin-name@marketplace-name
```

Réactiver un plugin désactivé :

```
/plugin enable plugin-name@marketplace-name
```

Supprimer complètement un plugin :

```
/plugin uninstall plugin-name@marketplace-name
```

L'option `--scope` vous permet de cibler une portée spécifique avec les commandes CLI :

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

Appliquer les modifications de plugin sans redémarrer

Quand vous installez, activez ou désactivez des plugins pendant une session, certaines modifications (comme les nouvelles commandes et hooks) prennent effet immédiatement. D'autres, y compris les mises à jour du serveur LSP, nécessitent un redémarrage.

Pour activer toutes les modifications de plugin en attente sans redémarrer, exécutez :

```
/reload-plugins
```

Claude Code recharge tous les plugins actifs et signale ce qui a été chargé. Si des serveurs LSP ont été ajoutés ou mis à jour, il vous le fera savoir, ceux-ci nécessitent un redémarrage pour prendre effet.

Gérer les marketplaces

Vous pouvez gérer les marketplaces via l'interface interactive `/plugin` ou avec des commandes CLI.

Utiliser l'interface interactive

Exécutez `/plugin` et allez à l'onglet **Marketplaces** pour :

- Visualiser toutes vos marketplaces ajoutées avec leurs sources et statut
- Ajouter de nouvelles marketplaces
- Mettre à jour les listes de marketplace pour récupérer les derniers plugins
- Supprimer les marketplaces dont vous n'avez plus besoin

Utiliser les commandes CLI

Vous pouvez également gérer les marketplaces avec des commandes directes.

Lister toutes les marketplaces configurées :

```
/plugin marketplace list
```

Actualiser les listes de plugins d'une marketplace :

```
/plugin marketplace update marketplace-name
```

Supprimer une marketplace :

```
/plugin marketplace remove marketplace-name
```

Warning:

La suppression d'une marketplace désinstallera tous les plugins que vous avez installés à partir de celle-ci.

Configurer les mises à jour automatiques

Claude Code peut automatiquement mettre à jour les marketplaces et leurs plugins installés au démarrage. Quand la mise à jour automatique est activée pour une marketplace, Claude Code actualise les données de la marketplace et met à jour les plugins installés vers leurs dernières versions. Si des plugins ont été mis à jour, vous verrez une notification vous invitant à exécuter `/reload-plugins`.

Basculez la mise à jour automatique pour les marketplaces individuelles via l'interface utilisateur :

1. Exécutez `/plugin` pour ouvrir le gestionnaire de plugins
2. Sélectionnez **Marketplaces**
3. Choisissez une marketplace dans la liste
4. Sélectionnez **Enable auto-update** ou **Disable auto-update**

Les marketplaces officielles Anthropic ont la mise à jour automatique activée par défaut. Les marketplaces tierces et de développement local ont la mise à jour automatique désactivée par défaut.

Pour désactiver complètement toutes les mises à jour automatiques pour Claude Code et tous les plugins, définissez la variable d'environnement `DISABLE_AUTOUPDATER`. Consultez [Auto updates](#) pour plus de détails.

Pour garder les mises à jour automatiques des plugins activées tout en désactivant les mises à jour automatiques de Claude Code, définissez `FORCE_AUTOUPDATE_PLUGINS=true` avec `DISABLE_AUTOUPDATER` :

```
export DISABLE_AUTOUPDATER=true
export FORCE_AUTOUPDATE_PLUGINS=true
```

Cela est utile quand vous voulez gérer les mises à jour de Claude Code manuellement mais recevoir toujours les mises à jour automatiques des plugins.

Configurer les marketplaces d'équipe

Les administrateurs d'équipe peuvent configurer l'installation automatique de marketplace pour les projets en ajoutant la configuration de marketplace à `.claude/settings.json`. Quand les membres de l'équipe font confiance au dossier du référentiel, Claude Code les invite à installer ces marketplaces et plugins.

Ajoutez `extraKnownMarketplaces` au `.claude/settings.json` de votre projet :

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

Pour les options de configuration complètes incluant `extraKnownMarketplaces` et `enabledPlugins`, consultez [Plugin settings](#).

Sécurité

Les plugins et les marketplaces sont des composants hautement fiables qui peuvent exécuter du code arbitraire sur votre machine avec vos privilèges utilisateur. Installez uniquement les plugins et ajoutez les marketplaces à partir de sources auxquelles vous faites confiance. Les organisations peuvent restreindre quelles marketplaces les utilisateurs sont autorisés à ajouter en utilisant [managed marketplace restrictions](#).

Dépannage

Commande /plugin non reconnue

Si vous voyez « unknown command » ou la commande `/plugin` n'apparaît pas :

1. **Vérifiez votre version** : Exécutez `claude --version`. Les plugins nécessitent la version 1.0.33 ou ultérieure.
2. **Mettez à jour Claude Code** :
 - **Homebrew** : `brew upgrade claude-code`
 - **npm** : `npm update -g @anthropic-ai/claude-code`
 - **Native installer** : Réexécutez la commande d'installation depuis [Setup](#)
3. **Redémarrez Claude Code** : Après la mise à jour, redémarrez votre terminal et exécutez `claude` à nouveau.

Problèmes courants

- **Marketplace ne se charge pas** : Vérifiez que l'URL est accessible et que `.claude-plugin/marketplace.json` existe au chemin
- **Échecs d'installation de plugin** : Vérifiez que les URLs sources du plugin sont accessibles et que les référentiels sont publics (ou vous avez accès)
- **Fichiers non trouvés après l'installation** : Les plugins sont copiés dans un cache, donc les chemins référençant des fichiers en dehors du répertoire du plugin ne fonctionneront pas
- **Les skills du plugin n'apparaissent pas** : Effacez le cache avec `rm -rf ~/.claude/plugins/cache`, redémarrez Claude Code et réinstallez le plugin.

Pour un dépannage détaillé avec des solutions, consultez [Dépannage](#) dans le guide de la marketplace. Pour les outils de débogage, consultez [Debugging and development tools](#).

Problèmes de code intelligence

- **Le serveur de langage ne démarre pas** : vérifiez que le binaire est installé et disponible dans votre `$PATH`. Consultez l'onglet `/plugin` Errors pour plus de détails.

- **Utilisation élevée de la mémoire** : les serveurs de langage comme `rust-analyzer` et `pyright` peuvent consommer une mémoire importante sur les grands projets. Si vous rencontrez des problèmes de mémoire, désactivez le plugin avec `/plugin disable <plugin-name>` et fiez-vous aux outils de recherche intégrés de Claude à la place.
- **Diagnostics faux positifs dans les monorepos** : les serveurs de langage peuvent signaler des erreurs d'import non résolues pour les packages internes si l'espace de travail n'est pas configuré correctement. Ceux-ci n'affectent pas la capacité de Claude à modifier le code.

Prochaines étapes

- **Construisez vos propres plugins** : Consultez [Plugins](#) pour créer des skills, des agents et des hooks
- **Créez une marketplace** : Consultez [Créer une marketplace de plugins](#) pour distribuer des plugins à votre équipe ou communauté
- **Référence technique** : Consultez [Plugins reference](#) pour les spécifications complètes

Part 6: Advanced & Automation

Créer des sous-agents personnalisés

Créez et utilisez des sous-agents IA spécialisés dans Claude Code pour des workflows spécifiques à des tâches et une meilleure gestion du contexte.

Les sous-agents sont des assistants IA spécialisés qui gèrent des types de tâches spécifiques. Chaque sous-agent s'exécute dans sa propre fenêtre de contexte avec une invite système personnalisée, un accès à des outils spécifiques et des permissions indépendantes. Lorsque Claude rencontre une tâche qui correspond à la description d'un sous-agent, il délègue à ce sous-agent, qui fonctionne indépendamment et retourne les résultats.

Note:

Si vous avez besoin de plusieurs agents travaillant en parallèle et communiquant entre eux, consultez plutôt [les équipes d'agents](#). Les sous-agents fonctionnent au sein d'une seule session ; les équipes d'agents coordonnent les sessions séparées.

Les sous-agents vous aident à :

- **Préserver le contexte** en gardant l'exploration et l'implémentation en dehors de votre conversation principale
- **Appliquer des contraintes** en limitant les outils qu'un sous-agent peut utiliser
- **Réutiliser les configurations** dans les projets avec des sous-agents au niveau utilisateur
- **Spécialiser le comportement** avec des invites système ciblées pour des domaines spécifiques
- **Contrôler les coûts** en acheminant les tâches vers des modèles plus rapides et moins chers comme Haiku

Claude utilise la description de chaque sous-agent pour décider quand déléguer les tâches. Lorsque vous créez un sous-agent, écrivez une description claire pour que Claude sache quand l'utiliser.

Claude Code inclut plusieurs sous-agents intégrés comme **Explore**, **Plan** et **general-purpose**. Vous pouvez également créer des sous-agents personnalisés pour gérer des tâches spécifiques. Cette page couvre les [sous-agents intégrés](#), [comment créer les vôtres](#), [les options de configuration complètes](#), [les modèles de travail avec les sous-agents](#) et les [exemples de sous-agents](#).

Sous-agents intégrés

Claude Code inclut des sous-agents intégrés que Claude utilise automatiquement le cas échéant. Chacun hérite des permissions de la conversation parent avec des restrictions d'outils supplémentaires.

Explore

Un agent rapide et en lecture seule optimisé pour la recherche et l'analyse de bases de code.

- **Modèle** : Haiku (rapide, faible latence)
- **Outils** : Outils en lecture seule (accès refusé aux outils Write et Edit)
- **Objectif** : Découverte de fichiers, recherche de code, exploration de base de code

Claude délègue à Explore lorsqu'il doit rechercher ou comprendre une base de code sans apporter de modifications. Cela garde les résultats d'exploration en dehors du contexte de votre conversation principale.

Lors de l'invocation d'Explore, Claude spécifie un niveau de minutie : **quick** pour les recherches ciblées, **medium** pour l'exploration équilibrée, ou **very thorough** pour l'analyse complète.

Plan

Un agent de recherche utilisé pendant le [mode plan](#) pour rassembler le contexte avant de présenter un plan.

- **Modèle** : Hérité de la conversation principale
- **Outils** : Outils en lecture seule (accès refusé aux outils Write et Edit)
- **Objectif** : Recherche de base de code pour la planification

Lorsque vous êtes en mode plan et que Claude doit comprendre votre base de code, il délègue la recherche au sous-agent Plan. Cela empêche l'imbrication infinie (les sous-agents ne peuvent pas générer d'autres sous-agents) tout en rassemblant le contexte nécessaire.

General-purpose

Un agent capable pour les tâches complexes et multi-étapes qui nécessitent à la fois l'exploration et l'action.

- **Modèle** : Hérité de la conversation principale
- **Outils** : Tous les outils
- **Objectif** : Recherche complexe, opérations multi-étapes, modifications de code

Claude délègue à general-purpose lorsque la tâche nécessite à la fois l'exploration et la modification, un raisonnement complexe pour interpréter les résultats, ou plusieurs étapes dépendantes.

Other

Claude Code inclut des agents d'assistance supplémentaires pour des tâches spécifiques. Ceux-ci sont généralement invoqués automatiquement, vous n'avez donc pas besoin de les utiliser directement.

Agent	Modèle	Quand Claude l'utilise
Bash	Hérité	Exécution de commandes de terminal dans un contexte séparé
statusline-setup	Sonnet	Lorsque vous exécutez <code>/statusline</code> pour configurer votre ligne d'état
Claude Code Guide	Haiku	Lorsque vous posez des questions sur les fonctionnalités de Claude Code

Au-delà de ces sous-agents intégrés, vous pouvez créer les vôtres avec des invites personnalisées, des restrictions d'outils, des modes de permission, des hooks et des skills. Les sections suivantes montrent comment commencer et personnaliser les sous-agents.

Démarrage rapide : créer votre premier sous-agent

Les sous-agents sont définis dans des fichiers Markdown avec du frontmatter YAML. Vous pouvez les [créer manuellement](#) ou utiliser la commande `/agents`.

Cette procédure pas à pas vous guide dans la création d'un sous-agent au niveau utilisateur avec la commande `/agent`. Le sous-agent examine le code et suggère des améliorations pour la base de code.

Step 1: Ouvrir l'interface des sous-agents

Dans Claude Code, exécutez :

```
/agents
```

Step 2: Créer un nouvel agent au niveau utilisateur

Sélectionnez **Create new agent**, puis choisissez **User-level**. Cela enregistre le sous-agent dans `~/ .claude/agents/` pour qu'il soit disponible dans tous vos projets.

Step 3: Générer avec Claude

Sélectionnez **Generate with Claude**. Lorsque vous y êtes invité, décrivez le sous-agent :

```
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 génère l'invite système et la configuration. Appuyez sur **e** pour l'ouvrir dans votre éditeur si vous souhaitez la personnaliser.

Step 4: Sélectionner les outils

Pour un examinateur en lecture seule, désélectionnez tout sauf **Read-only tools**. Si vous gardez tous les outils sélectionnés, le sous-agent hérite de tous les outils disponibles pour la conversation principale.

Step 5: Sélectionner le modèle

Choisissez le modèle que le sous-agent utilise. Pour cet exemple d'agent, sélectionnez **Sonnet**, qui équilibre la capacité et la vitesse pour analyser les modèles de code.

Step 6: Choisir une couleur

Choisissez une couleur de fond pour le sous-agent. Cela vous aide à identifier quel sous-agent s'exécute dans l'interface utilisateur.

Step 7: Enregistrer et essayer

Enregistrez le sous-agent. Il est disponible immédiatement (aucun redémarrage nécessaire). Essayez-le :

Use the code-improver agent to suggest improvements in this project

Claude délègue à votre nouveau sous-agent, qui analyse la base de code et retourne les suggestions d'amélioration.

Vous avez maintenant un sous-agent que vous pouvez utiliser dans n'importe quel projet sur votre machine pour analyser les bases de code et suggérer des améliorations.

Vous pouvez également créer des sous-agents manuellement en tant que fichiers Markdown, les définir via des drapeaux CLI ou les distribuer via des plugins. Les sections suivantes couvrent toutes les options de configuration.

Configurer les sous-agents

Utiliser la commande `/agents`

La commande `/agents` fournit une interface interactive pour gérer les sous-agents. Exécutez `/agents` pour :

- Afficher tous les sous-agents disponibles (intégrés, utilisateur, projet et plugin)
- Créer de nouveaux sous-agents avec une configuration guidée ou une génération Claude
- Modifier la configuration des sous-agents existants et l'accès aux outils
- Supprimer les sous-agents personnalisés
- Voir quels sous-agents sont actifs lorsque des doublons existent

C'est la méthode recommandée pour créer et gérer les sous-agents. Pour la création manuelle ou l'automatisation, vous pouvez également ajouter directement des fichiers de sous-agent.

Pour lister tous les sous-agents configurés à partir de la ligne de commande sans démarrer une session interactive, exécutez `claude agents`. Cela affiche les agents groupés par source et indique lesquels sont remplacés par des définitions de priorité plus élevée.

Choisir la portée du sous-agent

Les sous-agents sont des fichiers Markdown avec du frontmatter YAML. Stockez-les dans différents emplacements selon la portée. Lorsque plusieurs sous-agents partagent le même nom, l'emplacement de priorité plus élevée gagne.

Emplacement	Portée	Priorité	Comment créer
Drapeau CLI <code>--agents</code>	Session actuelle	1 (la plus élevée)	Passer JSON lors du lancement de Claude Code
<code>.claude/agents/</code>	Projet actuel	2	Interactif ou manuel
<code>~/.claude/agents/</code>	Tous vos projets	3	Interactif ou manuel
Répertoire <code>agents/</code> du plugin	Où le plugin est activé	4 (la plus basse)	Installé avec plugins

Les sous-agents de projet (`.claude/agents/`) sont idéaux pour les sous-agents spécifiques à une base de code. Enregistrez-les dans le contrôle de version pour que votre équipe puisse les utiliser et les améliorer de manière collaborative.

Les sous-agents utilisateur (`~/.claude/agents/`) sont des sous-agents personnels disponibles dans tous vos projets.

Les sous-agents définis par CLI sont passés en JSON lors du lancement de Claude Code. Ils n'existent que pour cette session et ne sont pas enregistrés sur le disque, ce qui les rend utiles pour les tests rapides ou les scripts d'automatisation :

```
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"
  }
}'
```

Le drapeau `--agents` accepte JSON avec les mêmes champs de [frontmatter](#) que les sous-agents basés sur fichier : `description`, `prompt`, `tools`, `disallowedTools`, `model`, `permissionMode`, `mcpServers`, `hooks`, `maxTurns`, `skills` et `memory`. Utilisez `prompt` pour l'invite système, équivalent au corps markdown dans les sous-agents basés sur fichier. Consultez la [référence CLI](#) pour le format JSON complet.

Les sous-agents de plugin proviennent des [plugins](#) que vous avez installés. Ils apparaissent dans `/agents` aux côtés de vos sous-agents personnalisés. Consultez la [référence des composants de plugin](#) pour plus de détails sur la création de sous-agents de plugin.

Écrire des fichiers de sous-agent

Les fichiers de sous-agent utilisent du frontmatter YAML pour la configuration, suivi de l'invite système en Markdown :

Note:

Les sous-agents sont chargés au démarrage de la session. Si vous créez un sous-agent en ajoutant manuellement un fichier, redémarrez votre session ou utilisez `/agents` pour le charger immédiatement.

```
---
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.
```

Le frontmatter définit les métadonnées et la configuration du sous-agent. Le corps devient l'invite système qui guide le comportement du sous-agent. Les sous-agents reçoivent uniquement cette invite système (plus les détails d'environnement de base comme le répertoire de travail), pas l'invite système complète de Claude Code.

Champs de frontmatter pris en charge

Les champs suivants peuvent être utilisés dans le frontmatter YAML. Seuls `name` et `description` sont obligatoires.

Champ	Obligatoire	Description
<code>name</code>	Oui	Identifiant unique utilisant des lettres minuscules et des tirets
<code>description</code>	Oui	Quand Claude doit déléguer à ce sous-agent
<code>tools</code>	Non	Outils que le sous-agent peut utiliser. Hérite de tous les outils s'il est omis
<code>disallowedTools</code>	Non	Outils à refuser, supprimés de la liste héritée ou spécifiée
<code>model</code>	Non	Modèle à utiliser : <code>sonnet</code> , <code>opus</code> , <code>haiku</code> ou <code>inherit</code> . Par défaut <code>inherit</code>
<code>permissionMode</code>	Non	Mode de permission : <code>default</code> , <code>acceptEdits</code> , <code>dontAsk</code> , <code>bypassPermissions</code> ou <code>plan</code>
<code>maxTurns</code>	Non	Nombre maximum de tours d'agent avant que le sous-agent s'arrête
<code>skills</code>	Non	Skills à charger dans le contexte du sous-agent au démarrage. Le contenu complet de la skill est injecté, pas seulement mis à disposition pour l'invocation. Les sous-agents n'héritent pas des skills de la conversation parent

Champ	Obligatoire	Description
mcpServers	Non	Serveurs MCP disponibles pour ce sous-agent. Chaque entrée est soit un nom de serveur référén-çant un serveur déjà confi-guré (par exemple, "slack") soit une défini-tion en ligne avec le nom du serveur comme clé et une configuration com-plète du serveur MCP comme valeur
hooks	Non	Hooks de cycle de vie limi-tés à ce sous-agent
memory	Non	Portée de la mémoire per-sistante : user , project ou local . Ac-tive l'apprentissage entre sessions
background	Non	Définir sur true pour toujours exécuter ce sous-agent en tant que tâche d'arrière-plan. Par défaut : false
isolation	Non	Définir sur worktree pour exécuter le sous-agent dans un git worktree temporaire, ce qui lui donne une copie isolée du référentiel. Le worktree est automatiquement nettoyé si le sous-agent n'apporte aucune modification

Choisir un modèle

Le champ model contrôle quel [modèle IA](#) le sous-agent utilise :

- **Alias de modèle** : Utilisez l'un des alias disponibles : sonnet , opus ou haiku

- **inherit** : Utiliser le même modèle que la conversation principale
- **Omis** : S'il n'est pas spécifié, par défaut **inherit** (utilise le même modèle que la conversation principale)

Contrôler les capacités du sous-agent

Vous pouvez contrôler ce que les sous-agents peuvent faire via l'accès aux outils, les modes de permission et les règles conditionnelles.

Outils disponibles

Les sous-agents peuvent utiliser n'importe lequel des [outils internes](#) de Claude Code. Par défaut, les sous-agents héritent de tous les outils de la conversation principale, y compris les outils MCP.

Pour restreindre les outils, utilisez le champ **tools** (liste blanche) ou le champ **disallowedTools** (liste noire) :

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
disallowedTools: Write, Edit
---
```

Restreindre les sous-agents qui peuvent être générés

Lorsqu'un agent s'exécute en tant que thread principal avec **claude --agent**, il peut générer des sous-agents à l'aide de l'outil Agent. Pour restreindre les types de sous-agents qu'il peut générer, utilisez la syntaxe **Agent(agent_type)** dans le champ **tools**.

Note:

Dans la version 2.1.63, l'outil Task a été renommé en Agent. Les références **Task(...)** existantes dans les paramètres et les définitions d'agent fonctionnent toujours comme des alias.

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

C'est une liste blanche : seuls les sous-agents `worker` et `researcher` peuvent être générés. Si l'agent essaie de générer un autre type, la demande échoue et l'agent ne voit que les types autorisés dans son invite. Pour bloquer des agents spécifiques tout en autorisant d'autres, utilisez plutôt `permissions.deny`.

Pour autoriser la génération de n'importe quel sous-agent sans restrictions, utilisez `Agent` sans parenthèses :

```
tools: Agent, Read, Bash
```

Si `Agent` est complètement omis de la liste `tools`, l'agent ne peut générer aucun sous-agent. Cette restriction s'applique uniquement aux agents s'exécutant en tant que thread principal avec `claude --agent`. Les sous-agents ne peuvent pas générer d'autres sous-agents, donc `Agent(agent_type)` n'a aucun effet dans les définitions de sous-agent.

Modes de permission

Le champ `permissionMode` contrôle comment le sous-agent gère les invites de permission. Les sous-agents héritent du contexte de permission de la conversation principale mais peuvent remplacer le mode.

Mode	Comportement
<code>default</code>	Vérification de permission standard avec invites
<code>acceptEdits</code>	Acceptation automatique des modifications de fichiers
<code>dontAsk</code>	Refus automatique des invites de permission (les outils explicitement autorisés fonctionnent toujours)
<code>bypassPermissions</code>	Ignorer toutes les vérifications de permission
<code>plan</code>	Mode plan (exploration en lecture seule)

Warning:

Utilisez `bypassPermissions` avec prudence. Il ignore toutes les vérifications de permission, permettant au sous-agent d'exécuter n'importe quelle opération sans approbation.

Si le parent utilise `bypassPermissions`, cela prend la priorité et ne peut pas être remplacé.

Précharger les skills dans les sous-agents

Utilisez le champ `skills` pour injecter le contenu de la skill dans le contexte du sous-agent au démarrage. Cela donne au sous-agent des connaissances de domaine sans qu'il ait besoin de découvrir et charger les skills pendant l'exécution.

```
---
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.
```

Le contenu complet de chaque skill est injecté dans le contexte du sous-agent, pas seulement mis à disposition pour l'invocation. Les sous-agents n'héritent pas des skills de la conversation parent ; vous devez les lister explicitement.

Note:

C'est l'inverse de [l'exécution d'une skill dans un sous-agent](#). Avec `skills` dans un sous-agent, le sous-agent contrôle l'invite système et charge le contenu de la skill. Avec `context: fork` dans une skill, le contenu de la skill est injecté dans l'agent que vous spécifiez. Les deux utilisent le même système sous-jacent.

Activer la mémoire persistante

Le champ `memory` donne au sous-agent un répertoire persistant qui survit aux conversations. Le sous-agent utilise ce répertoire pour accumuler des connaissances au fil du temps, comme les modèles de base de code, les insights de débogage et les décisions architecturales.

```
---
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.

Choisissez une portée en fonction de la largeur avec laquelle la mémoire doit s’appliquer :

Portée	Emplacement	Utiliser quand
user	<code>~/.claude/agent-memory/<name-of-agent>/</code>	le sous-agent doit se souvenir des apprentissages dans tous les projets
project	<code>.claude/agent-memory/<name-of-agent>/</code>	les connaissances du sous-agent sont spécifiques au projet et partageables via le contrôle de version
local	<code>.claude/agent-memory-local/<name-of-agent>/</code>	les connaissances du sous-agent sont spécifiques au projet mais ne doivent pas être enregistrées dans le contrôle de version

Lorsque la mémoire est activée :

- L’invite système du sous-agent inclut des instructions pour lire et écrire dans le répertoire de mémoire.
- L’invite système du sous-agent inclut également les 200 premières lignes de `MEMORY.md` dans le répertoire de mémoire, avec des instructions pour organiser `MEMORY.md` s’il dépasse 200 lignes.
- Les outils Read, Write et Edit sont automatiquement activés pour que le sous-agent puisse gérer ses fichiers de mémoire.

Conseils de mémoire persistante

- `user` est la portée par défaut recommandée. Utilisez `project` ou `local` lorsque les connaissances du sous-agent ne sont pertinentes que pour une base de code spécifique.
- Demandez au sous-agent de consulter sa mémoire avant de commencer le travail :
« Examinez cette PR et consultez votre mémoire pour les modèles que vous avez vus auparavant. »
- Demandez au sous-agent de mettre à jour sa mémoire après avoir terminé une tâche :
« Maintenant que vous avez terminé, enregistrez ce que vous avez appris dans votre mémoire. » Au fil du temps, cela crée une base de connaissances qui rend le sous-agent plus efficace.
- Incluez les instructions de mémoire directement dans le fichier markdown du sous-agent pour qu'il maintienne proactivement sa propre base de connaissances :

```
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.
```

Règles conditionnelles avec hooks

Pour un contrôle plus dynamique de l'utilisation des outils, utilisez les hooks `PreToolUse` pour valider les opérations avant leur exécution. C'est utile lorsque vous devez autoriser certaines opérations d'un outil tout en bloquant d'autres.

Cet exemple crée un sous-agent qui n'autorise que les requêtes de base de données en lecture seule. Le hook `PreToolUse` exécute le script spécifié dans `command` avant chaque commande 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 [passe l'entrée du hook en JSON](#) via stdin aux commandes du hook. Le script de validation lit ce JSON, extrait la commande Bash et [quitte avec le code 2](#) pour bloquer les opérations d'écriture :

```

#!/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

```

Consultez [Hook input](#) pour le schéma d'entrée complet et [les codes de sortie](#) pour savoir comment les codes de sortie affectent le comportement.

Désactiver les sous-agents spécifiques

Vous pouvez empêcher Claude d'utiliser des sous-agents spécifiques en les ajoutant au tableau `deny` dans vos [paramètres](#). Utilisez le format `Agent(subagent-name)` où `subagent-name` correspond au champ `name` du sous-agent.

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

Cela fonctionne pour les sous-agents intégrés et personnalisés. Vous pouvez également utiliser le drapeau CLI `--disallowedTools` :

```
claude --disallowedTools "Agent(Explore)"
```

Consultez la [documentation Permissions](#) pour plus de détails sur les règles de permission.

Définir les hooks pour les sous-agents

Les sous-agents peuvent définir des [hooks](#) qui s'exécutent pendant le cycle de vie du sous-agent. Il y a deux façons de configurer les hooks :

1. **Dans le frontmatter du sous-agent** : Définir les hooks qui s'exécutent uniquement pendant que ce sous-agent est actif
2. **Dans `settings.json`** : Définir les hooks qui s'exécutent dans la session principale lorsque les sous-agents démarrent ou s'arrêtent

Hooks dans le frontmatter du sous-agent

Définissez les hooks directement dans le fichier markdown du sous-agent. Ces hooks s'exécutent uniquement pendant que ce sous-agent spécifique est actif et sont nettoyés lorsqu'il se termine.

Tous les [événements de hook](#) sont pris en charge. Les événements les plus courants pour les sous-agents sont :

Événement	Entrée du matcher	Quand il se déclenche
PreToolUse	Nom de l'outil	Avant que le sous-agent utilise un outil
PostToolUse	Nom de l'outil	Après que le sous-agent utilise un outil
Stop	(aucun)	Quand le sous-agent se termine (converti en SubagentStop à l'exécution)

Cet exemple valide les commandes Bash avec le hook `PreToolUse` et exécute un linter après les modifications de fichiers avec `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"
---
```

Les hooks `Stop` dans le frontmatter sont automatiquement convertis en événements `SubagentStop`.

Hooks au niveau du projet pour les événements de sous-agent

Configurez les hooks dans `settings.json` qui répondent aux événements du cycle de vie du sous-agent dans la session principale.

Événement	Entrée du matcher	Quand il se déclenche
<code>SubagentStart</code>	Nom du type d'agent	Quand un sous-agent commence l'exécution
<code>SubagentStop</code>	Nom du type d'agent	Quand un sous-agent se termine

Les deux événements prennent en charge les matchers pour cibler des types d'agents spécifiques par nom. Cet exemple exécute un script de configuration uniquement lorsque le sous-agent `db-agent` démarre et un script de nettoyage lorsque n'importe quel sous-agent s'arrête :

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

Consultez [Hooks](#) pour le format de configuration complet des hooks.

Travailler avec les sous-agents

Comprendre la délégation automatique

Claude délègue automatiquement les tâches en fonction de la description de la tâche dans votre demande, du champ `description` dans les configurations de sous-agent et du contexte actuel. Pour encourager la délégation proactive, incluez des phrases comme « use proactively » dans le champ description de votre sous-agent.

Vous pouvez également demander un sous-agent spécifique explicitement :

```
Use the test-runner subagent to fix failing tests
Have the code-reviewer subagent look at my recent changes
```

Exécuter les sous-agents au premier plan ou en arrière-plan

Les sous-agents peuvent s'exécuter au premier plan (bloquant) ou en arrière-plan (concurrent) :

- **Les sous-agents au premier plan** bloquent la conversation principale jusqu'à la fin. Les invites de permission et les questions de clarification (comme `AskUserQuestion`) vous sont transmises.
- **Les sous-agents en arrière-plan** s'exécutent simultanément pendant que vous continuez à travailler. Avant de lancer, Claude Code vous demande les permissions d'outils dont le sous-agent aura besoin, en s'assurant qu'il a les approbations nécessaires à l'avance. Une fois en cours d'exécution, le sous-agent hérite de ces permissions et refuse automatiquement tout ce qui n'est pas pré-approuvé. Si un sous-agent en arrière-plan doit poser des questions de clarification, cet appel d'outil échoue mais le sous-agent continue.

Si un sous-agent en arrière-plan échoue en raison de permissions manquantes, vous pouvez le [reprendre](#) au premier plan pour réessayer avec des invites interactives.

Claude décide si les sous-agents s'exécutent au premier plan ou en arrière-plan en fonction de la tâche. Vous pouvez également :

- Demander à Claude de « run this in the background »
- Appuyer sur **Ctrl+B** pour mettre une tâche en arrière-plan

Pour désactiver toute la fonctionnalité de tâche en arrière-plan, définissez la variable d'environnement `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` sur `1`. Consultez [Variables d'environnement](#).

Modèles courants

Isoler les opérations à haut volume

L'une des utilisations les plus efficaces des sous-agents est l'isolation des opérations qui produisent de grandes quantités de résultats. L'exécution de tests, la récupération de documentation ou le traitement de fichiers journaux peuvent consommer un contexte important. En déléguant ces tâches à un sous-agent, la sortie détaillée reste dans le contexte du sous-agent tandis que seul le résumé pertinent revient à votre conversation principale.

```
Use a subagent to run the test suite and report only the failing tests with their error messages
```

Exécuter la recherche en parallèle

Pour les investigations indépendantes, générez plusieurs sous-agents pour travailler simultanément :

```
Research the authentication, database, and API modules in parallel using separate subagents
```

Chaque sous-agent explore son domaine indépendamment, puis Claude synthétise les résultats. Cela fonctionne mieux lorsque les chemins de recherche ne dépendent pas les uns des autres.

Warning:

Lorsque les sous-agents se terminent, leurs résultats reviennent à votre conversation principale. L'exécution de nombreux sous-agents qui retournent chacun des résultats détaillés peut consommer un contexte important.

Pour les tâches qui nécessitent un parallélisme soutenu ou qui dépassent votre fenêtre de contexte, les [équipes d'agents](#) donnent à chaque travailleur son propre contexte indépendant.

Chaîner les sous-agents

Pour les workflows multi-étapes, demandez à Claude d'utiliser les sous-agents en séquence. Chaque sous-agent termine sa tâche et retourne les résultats à Claude, qui transmet ensuite le contexte pertinent au sous-agent suivant.

Use the code-reviewer subagent to find performance issues, then use the optimizer subagent to fix them

Choisir entre les sous-agents et la conversation principale

Utilisez la **conversation principale** quand :

- La tâche nécessite des allers-retours fréquents ou un raffinement itératif
- Plusieurs phases partagent un contexte important (planification → implémentation → test)
- Vous apportez une modification rapide et ciblée
- La latence est importante. Les sous-agents commencent à zéro et peuvent avoir besoin de temps pour rassembler le contexte

Utilisez les **sous-agents** quand :

- La tâche produit une sortie détaillée dont vous n'avez pas besoin dans votre contexte principal
- Vous souhaitez appliquer des restrictions d'outils ou des permissions spécifiques
- Le travail est autonome et peut retourner un résumé

Envisagez plutôt les [Skills](#) lorsque vous souhaitez des invites ou des workflows réutilisables qui s'exécutent dans le contexte de la conversation principale plutôt que dans un contexte de sous-agent isolé.

Pour une question rapide sur quelque chose déjà dans votre conversation, utilisez [/btw](#) au lieu d'un sous-agent. Il voit votre contexte complet mais n'a pas d'accès aux outils, et la réponse est rejetée plutôt que d'être ajoutée à l'historique.

Note:

Les sous-agents ne peuvent pas générer d'autres sous-agents. Si votre workflow nécessite une délégation imbriquée, utilisez [Skills](#) ou [chaînez les sous-agents](#) à partir de la conversation principale.

Gérer le contexte du sous-agent

Reprendre les sous-agents

Chaque invocation de sous-agent crée une nouvelle instance avec un contexte frais. Pour continuer le travail d'un sous-agent existant au lieu de recommencer, demandez à Claude de le reprendre.

Les sous-agents repris conservent leur historique de conversation complet, y compris tous les appels d'outils précédents, les résultats et le raisonnement. Le sous-agent reprend exactement où il s'était arrêté plutôt que de recommencer à zéro.

Lorsqu'un sous-agent se termine, Claude reçoit son ID d'agent. Pour reprendre un sous-agent, demandez à Claude de continuer le travail précédent :

```
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]
```

Vous pouvez également demander à Claude l'ID d'agent si vous souhaitez le référencer explicitement, ou trouver les ID dans les fichiers de transcription à `~/.claude/projects/{project}/{sessionId}/subagents/`. Chaque transcription est stockée sous la forme `agent-{agentId}.jsonl`.

Les transcriptions de sous-agent persistent indépendamment de la conversation principale :

- **Compaction de la conversation principale** : Lorsque la conversation principale se compacte, les transcriptions de sous-agent ne sont pas affectées. Elles sont stockées dans des fichiers séparés.
- **Persistance de session** : Les transcriptions de sous-agent persistent au sein de leur session. Vous pouvez [reprendre un sous-agent](#) après avoir redémarré Claude Code en reprenant la même session.
- **Nettoyage automatique** : Les transcriptions sont nettoyées en fonction du paramètre `cleanupPeriodDays` (par défaut : 30 jours).

Auto-compaction

Les sous-agents prennent en charge la compaction automatique en utilisant la même logique que la conversation principale. Par défaut, la compaction automatique se déclenche à environ 95 % de capacité. Pour déclencher la compaction plus tôt, définissez `CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` sur un pourcentage inférieur (par exemple, `50`). Consultez [variables d'environnement](#) pour plus de détails.

Les événements de compaction sont enregistrés dans les fichiers de transcription de sous-agent :

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
  }
}
```

La valeur `preTokens` indique combien de tokens ont été utilisés avant la compaction.

Exemples de sous-agents

Ces exemples démontrent des modèles efficaces pour construire des sous-agents. Utilisez-les comme points de départ ou générez une version personnalisée avec Claude.

Tip:

Meilleures pratiques :

- **Concevoir des sous-agents ciblés** : chaque sous-agent doit exceller dans une tâche spécifique
- **Écrire des descriptions détaillées** : Claude utilise la description pour décider quand déléguer
- **Limiter l'accès aux outils** : accordez uniquement les permissions nécessaires pour la sécurité et la concentration
- **Enregistrer dans le contrôle de version** : partagez les sous-agents de projet avec votre équipe

Examineur de code

Un sous-agent en lecture seule qui examine le code sans le modifier. Cet exemple montre comment concevoir un sous-agent ciblé avec un accès limité aux outils (pas d'outils Edit ou Write) et une invite détaillée qui spécifie exactement ce qu'il faut rechercher et comment formater la sortie.

```
---  
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.

Débogueur

Un sous-agent qui peut à la fois analyser et corriger les problèmes. Contrairement à l'examineur de code, celui-ci inclut Edit car la correction des bugs nécessite de modifier le code. L'invite fournit un workflow clair du diagnostic à la vérification.

```
---  
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 sous-agent spécifique au domaine pour le travail d'analyse de données. Cet exemple montre comment créer des sous-agents pour des workflows spécialisés en dehors des tâches de codage typiques. Il définit explicitement `model: sonnet` pour une analyse plus capable.


```
---
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.

Valideur de requête de base de données

Un sous-agent qui permet l'accès Bash mais valide les commandes pour n'autoriser que les requêtes SQL en lecture seule. Cet exemple montre comment utiliser les hooks

`PreToolUse` pour la validation conditionnelle lorsque vous avez besoin d'un contrôle plus fin que le champ `tools` ne le permet.

```

---
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 [passe l'entrée du hook en JSON](#) via stdin aux commandes du hook. Le script de validation lit ce JSON, extrait la commande en cours d'exécution et la vérifie par rapport à une liste d'opérations d'écriture SQL. Si une opération d'écriture est détectée, le script [quitte avec le code 2](#) pour bloquer l'exécution et retourne un message d'erreur à Claude via stderr.

Créez le script de validation n'importe où dans votre projet. Le chemin doit correspondre au champ `command` dans votre configuration de 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
```

Rendez le script exécutable :

```
chmod +x ./scripts/validate-readonly-query.sh
```

Le hook reçoit JSON via stdin avec la commande Bash dans `tool_input.command`. Le code de sortie 2 bloque l'opération et renvoie le message d'erreur à Claude. Consultez [Hooks](#) pour plus de détails sur les codes de sortie et [Hook input](#) pour le schéma d'entrée complet.

Étapes suivantes

Maintenant que vous comprenez les sous-agents, explorez ces fonctionnalités connexes :

- [Distribuer les sous-agents avec les plugins](#) pour partager les sous-agents entre les équipes ou les projets
- [Exécuter Claude Code par programmation](#) avec le SDK Agent pour CI/CD et l'automatisation

- [Utiliser les serveurs MCP](#) pour donner aux sous-agents l'accès aux outils et données externes

Orchestrer des équipes de sessions Claude Code

Coordonnez plusieurs instances Claude Code travaillant ensemble en tant qu'équipe, avec des tâches partagées, la messagerie inter-agents et la gestion centralisée.

Warning:

Les équipes d'agents sont expérimentales et désactivées par défaut. Activez-les en ajoutant `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` à votre [settings.json](#) ou à votre environnement. Les équipes d'agents ont des [limitations connues](#) concernant la reprise de session, la coordination des tâches et le comportement d'arrêt.

Les équipes d'agents vous permettent de coordonner plusieurs instances Claude Code travaillant ensemble. Une session agit comme chef d'équipe, coordonnant le travail, assignant des tâches et synthétisant les résultats. Les coéquipiers travaillent indépendamment, chacun dans sa propre fenêtre de contexte, et communiquent directement les uns avec les autres.

Contrairement aux [subagents](#), qui s'exécutent au sein d'une seule session et ne peuvent que rendre compte à l'agent principal, vous pouvez également interagir directement avec les coéquipiers individuels sans passer par le chef.

Note:

Les équipes d'agents nécessitent Claude Code v2.1.32 ou ultérieur. Vérifiez votre version avec `claude --version`.

Cette page couvre :

- [Quand utiliser les équipes d'agents](#), y compris les meilleurs cas d'usage et comment ils se comparent aux subagents
- [Démarrer une équipe](#)
- [Contrôler les coéquipiers](#), y compris les modes d'affichage, l'assignation de tâches et la délégation
- [Meilleures pratiques pour le travail parallèle](#)

Quand utiliser les équipes d'agents

Les équipes d'agents sont les plus efficaces pour les tâches où l'exploration parallèle ajoute une réelle valeur. Consultez les [exemples de cas d'usage](#) pour des scénarios complets. Les cas d'usage les plus solides sont :

- **Recherche et examen** : plusieurs coéquipiers peuvent enquêter sur différents aspects d'un problème simultanément, puis partager et contester les conclusions les uns des autres
- **Nouveaux modules ou fonctionnalités** : les coéquipiers peuvent chacun posséder une partie distincte sans se marcher dessus
- **Débogage avec hypothèses concurrentes** : les coéquipiers testent différentes théories en parallèle et convergent vers la réponse plus rapidement
- **Coordination inter-couches** : les modifications qui s'étendent sur le frontend, le backend et les tests, chacun possédé par un coéquipier différent

Les équipes d'agents ajoutent une surcharge de coordination et utilisent considérablement plus de tokens qu'une seule session. Elles fonctionnent mieux lorsque les coéquipiers peuvent opérer indépendamment. Pour les tâches séquentielles, les modifications du même fichier ou le travail avec de nombreuses dépendances, une seule session ou les [sub-agents](#) sont plus efficaces.

Comparer avec les subagents

Les équipes d'agents et les [subagents](#) vous permettent tous deux de paralléliser le travail, mais ils fonctionnent différemment. Choisissez en fonction de la nécessité pour vos travailleurs de communiquer les uns avec les autres :

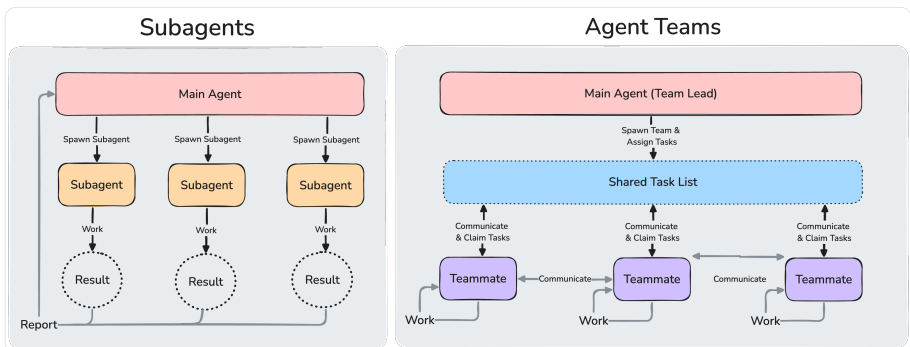


Diagramme comparant les architectures des subagents et des équipes d'agents. Les subagents sont générés par l'agent principal, font du travail et rendent compte des résultats. Les équipes d'agents se coordonnent via une liste de tâches partagée, avec les coéquipiers communiquant directement les uns avec les autres.

	Subagents	Équipes d'agents
Contexte	Fenêtre de contexte propre ; les résultats reviennent à l'appelant	Fenêtre de contexte propre ; complètement indépendant
Communication	Rendre compte uniquement à l'agent principal	Les coéquipiers se mesagent directement
Coordination	L'agent principal gère tout le travail	Liste de tâches partagée avec auto-coordination
Meilleur pour	Les tâches ciblées où seul le résultat compte	Le travail complexe nécessitant discussion et collaboration
Coût en tokens	Inférieur : les résultats sont résumés au contexte principal	Supérieur : chaque coéquipier est une instance Claude distincte

Utilisez les subagents lorsque vous avez besoin de travailleurs rapides et ciblés qui rendent compte. Utilisez les équipes d'agents lorsque les coéquipiers doivent partager les conclusions, se contester mutuellement et se coordonner de manière autonome.

Activer les équipes d'agents

Les équipes d'agents sont désactivées par défaut. Activez-les en définissant la variable d'environnement `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` sur `1`, soit dans votre environnement shell, soit via [settings.json](#) :

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS": "1"
  }
}
```

Démarrer votre première équipe d'agents

Après avoir activé les équipes d'agents, demandez à Claude de créer une équipe d'agents et décrivez la tâche et la structure d'équipe que vous souhaitez en langage naturel. Claude crée l'équipe, génère les coéquipiers et coordonne le travail en fonction de votre prompt.

Cet exemple fonctionne bien car les trois rôles sont indépendants et peuvent explorer le problème sans attendre les uns les autres :

```
Je conçois un outil CLI qui aide les développeurs à suivre les commentaires TODO
dans
leur base de code. Créez une équipe d'agents pour explorer cela sous différents
angles : un
coéquipier sur l'UX, un sur l'architecture technique, un jouant l'avocat du
diable.
```

À partir de là, Claude crée une équipe avec une [liste de tâches partagée](#), génère les coéquipiers pour chaque perspective, les fait explorer le problème, synthétise les conclusions et tente de [nettoyer l'équipe](#) une fois terminée.

Le terminal du chef liste tous les coéquipiers et sur quoi ils travaillent. Utilisez Maj+Bas pour parcourir les coéquipiers et leur envoyer un message directement. Après le dernier coéquipier, Maj+Bas revient au chef.

Si vous souhaitez que chaque coéquipier soit dans son propre volet divisé, consultez [Choisir un mode d'affichage](#).

Contrôler votre équipe d'agents

Dites au chef ce que vous voulez en langage naturel. Il gère la coordination d'équipe, l'assignation de tâches et la délégation en fonction de vos instructions.

Choisir un mode d'affichage

Les équipes d'agents supportent deux modes d'affichage :

- **In-process** : tous les coéquipiers s'exécutent dans votre terminal principal. Utilisez Maj+Bas pour parcourir les coéquipiers et tapez pour leur envoyer un message directement. Fonctionne dans n'importe quel terminal, aucune configuration supplémentaire requise.
- **Volets divisés** : chaque coéquipier obtient son propre volet. Vous pouvez voir la sortie de tout le monde à la fois et cliquer dans un volet pour interagir directement. Nécessite tmux ou iTerm2.

Note:

`tmux` a des limitations connues sur certains systèmes d'exploitation et fonctionne traditionnellement mieux sur macOS. L'utilisation de `tmux -CC` dans iTerm2 est le point d'entrée suggéré dans `tmux`.

La valeur par défaut est `"auto"`, qui utilise les volets divisés si vous êtes déjà en train de s'exécuter dans une session `tmux`, et in-process sinon. Le paramètre `"tmux"` active le mode volets divisés et détecte automatiquement s'il faut utiliser `tmux` ou iTerm2 en fonction de votre terminal. Pour remplacer, définissez `teammateMode` dans votre [set-tings.json](#) :

```
{
  "teammateMode": "in-process"
}
```

Pour forcer le mode in-process pour une seule session, passez-le en tant que drapeau :

```
claude --teammate-mode in-process
```

Le mode volets divisés nécessite soit [tmux](#) soit iTerm2 avec le CLI [it2](#). Pour installer manuellement :

- **tmux** : installez via le gestionnaire de paquets de votre système. Consultez le [wiki tmux](#) pour les instructions spécifiques à la plateforme.
- **iTerm2** : installez le CLI [it2](#), puis activez l'API Python dans **iTerm2** → Paramètres → Général → Magie → Activer l'API Python.

Spécifier les coéquipiers et les modèles

Claude décide du nombre de coéquipiers à générer en fonction de votre tâche, ou vous pouvez spécifier exactement ce que vous voulez :

Créez une équipe avec 4 coéquipiers pour refactoriser ces modules en parallèle. Utilisez Sonnet pour chaque coéquipier.

Exiger l'approbation du plan pour les coéquipiers

Pour les tâches complexes ou risquées, vous pouvez exiger que les coéquipiers planifient avant de mettre en œuvre. Le coéquipier travaille en mode plan en lecture seule jusqu'à ce que le chef approuve son approche :

Générez un coéquipier architecte pour refactoriser le module d'authentification. Exigez l'approbation du plan avant qu'il ne fasse des modifications.

Lorsqu'un coéquipier termine la planification, il envoie une demande d'approbation du plan au chef. Le chef examine le plan et l'approuve ou le rejette avec des commentaires. S'il est rejeté, le coéquipier reste en mode plan, révisé en fonction des commentaires et resoumis. Une fois approuvé, le coéquipier quitte le mode plan et commence la mise en œuvre.

Le chef prend les décisions d'approbation de manière autonome. Pour influencer le jugement du chef, donnez-lui des critères dans votre prompt, tels que « n'approuvez que les plans qui incluent la couverture de test » ou « rejetez les plans qui modifient le schéma de base de données ».

Parler directement aux coéquipiers

Chaque coéquipier est une session Claude Code complète et indépendante. Vous pouvez envoyer un message à n'importe quel coéquipier directement pour donner des instructions supplémentaires, poser des questions de suivi ou rediriger son approche.

- **Mode in-process** : utilisez Maj+Bas pour parcourir les coéquipiers, puis tapez pour leur envoyer un message. Appuyez sur Entrée pour afficher la session d'un coéquipier, puis Échap pour interrompre son tour actuel. Appuyez sur Ctrl+T pour basculer la liste des tâches.
- **Mode volets divisés** : cliquez dans le volet d'un coéquipier pour interagir directement avec sa session. Chaque coéquipier a une vue complète de son propre terminal.

Assigner et revendiquer des tâches

La liste de tâches partagée coordonne le travail dans l'équipe. Le chef crée des tâches et les coéquipiers les accomplissent. Les tâches ont trois états : en attente, en cours et terminées. Les tâches peuvent également dépendre d'autres tâches : une tâche en attente avec des dépendances non résolues ne peut pas être revendiquée jusqu'à ce que ces dépendances soient complétées.

Le chef peut assigner des tâches explicitement, ou les coéquipiers peuvent les revendiquer eux-mêmes :

- **Le chef assigne** : dites au chef quelle tâche donner à quel coéquipier
- **Auto-revendication** : après avoir terminé une tâche, un coéquipier choisit la prochaine tâche non assignée et non bloquée de sa propre initiative

La revendication de tâche utilise le verrouillage de fichiers pour prévenir les conditions de course lorsque plusieurs coéquipiers tentent de revendiquer la même tâche simultanément.

Arrêter les coéquipiers

Pour terminer gracieusement la session d'un coéquipier :

```
Demandez au coéquipier chercheur d'arrêter
```

Le chef envoie une demande d'arrêt. Le coéquipier peut approuver, quittant gracieusement, ou rejeter avec une explication.

Nettoyer l'équipe

Lorsque vous avez terminé, demandez au chef de nettoyer :

```
Nettoyez l'équipe
```

Cela supprime les ressources d'équipe partagées. Lorsque le chef exécute le nettoyage, il vérifie les coéquipiers actifs et échoue s'il y en a encore en cours d'exécution, alors arrêtez-les d'abord.

Warning:

Utilisez toujours le chef pour nettoyer. Les coéquipiers ne doivent pas exécuter le nettoyage car leur contexte d'équipe peut ne pas se résoudre correctement, laissant potentiellement les ressources dans un état incohérent.

Appliquer des portes de qualité avec des hooks

Utilisez les [hooks](#) pour appliquer des règles lorsque les coéquipiers terminent le travail ou que les tâches se complètent :

- `TeammateIdle` : s'exécute lorsqu'un coéquipier est sur le point de devenir inactif. Quittez avec le code 2 pour envoyer des commentaires et garder le coéquipier au travail.
- `TaskCompleted` : s'exécute lorsqu'une tâche est marquée comme complète. Quittez avec le code 2 pour empêcher la complétion et envoyer des commentaires.

Comment fonctionnent les équipes d’agents

Cette section couvre l’architecture et la mécanique derrière les équipes d’agents. Si vous souhaitez commencer à les utiliser, consultez [Contrôler votre équipe d’agents](#) ci-dessus.

Comment Claude démarre les équipes d’agents

Il y a deux façons de démarrer les équipes d’agents :

- **Vous demandez une équipe** : donnez à Claude une tâche qui bénéficie du travail parallèle et demandez explicitement une équipe d’agents. Claude en crée une en fonction de vos instructions.
- **Claude propose une équipe** : si Claude détermine que votre tâche bénéficierait du travail parallèle, il peut suggérer de créer une équipe. Vous confirmez avant qu’il ne procède.

Dans les deux cas, vous restez maître. Claude ne créera pas d’équipe sans votre approbation.

Architecture

Une équipe d’agents se compose de :

Compo- sant	Rôle
Chef d’équipe	La session Claude Code principale qui crée l’équipe, génère les coéquipiers et coordonne le travail
Coéquipiers	Des instances Claude Code distinctes qui travaillent chacune sur des tâches assignées
Liste de tâches	Liste partagée d’éléments de travail que les coéquipiers revendiquent et complètent
Boîte aux lettres	Système de messagerie pour la communication entre agents

Consultez [Choisir un mode d’affichage](#) pour les options de configuration d’affichage. Les messages des coéquipiers arrivent au chef automatiquement.

Le système gère automatiquement les dépendances de tâches. Lorsqu’un coéquipier complète une tâche dont d’autres tâches dépendent, les tâches bloquées se débloquent sans intervention manuelle.

Les équipes et les tâches sont stockées localement :

- **Configuration d'équipe** : `~/.claude/teams/{team-name}/config.json`
- **Liste de tâches** : `~/.claude/tasks/{team-name}/`

La configuration d'équipe contient un tableau `members` avec le nom de chaque coéquipier, l'ID d'agent et le type d'agent. Les coéquipiers peuvent lire ce fichier pour découvrir les autres membres de l'équipe.

Permissions

Les coéquipiers commencent avec les paramètres de permission du chef. Si le chef s'exécute avec `--dangerously-skip-permissions`, tous les coéquipiers le font aussi. Après la génération, vous pouvez modifier les modes de coéquipiers individuels, mais vous ne pouvez pas définir les modes par coéquipier au moment de la génération.

Contexte et communication

Chaque coéquipier a sa propre fenêtre de contexte. Lorsqu'il est généré, un coéquipier charge le même contexte de projet qu'une session régulière : CLAUDE.md, MCP servers et skills. Il reçoit également le prompt de génération du chef. L'historique de conversation du chef ne se transporte pas.

Comment les coéquipiers partagent les informations :

- **Livraison automatique de messages** : lorsque les coéquipiers envoient des messages, ils sont livrés automatiquement aux destinataires. Le chef n'a pas besoin d'interroger les mises à jour.
- **Notifications d'inactivité** : lorsqu'un coéquipier termine et s'arrête, il notifie automatiquement le chef.
- **Liste de tâches partagée** : tous les agents peuvent voir l'état des tâches et revendiquer le travail disponible.

Messagerie des coéquipiers :

- **message** : envoyer un message à un coéquipier spécifique
- **broadcast** : envoyer à tous les coéquipiers simultanément. À utiliser avec parcimonie, car les coûts augmentent avec la taille de l'équipe.

Utilisation des tokens

Les équipes d'agents utilisent considérablement plus de tokens qu'une seule session. Chaque coéquipier a sa propre fenêtre de contexte, et l'utilisation des tokens augmente avec le nombre de coéquipiers actifs. Pour la recherche, l'examen et le travail sur les

nouvelles fonctionnalités, les tokens supplémentaires en valent généralement la peine. Pour les tâches de routine, une seule session est plus rentable. Consultez les [coûts des tokens des équipes d'agents](#) pour les conseils d'utilisation.

Exemples de cas d'usage

Ces exemples montrent comment les équipes d'agents gèrent les tâches où l'exploration parallèle ajoute de la valeur.

Exécuter un examen de code parallèle

Un seul examinateur tend à graviter vers un type de problème à la fois. Diviser les critères d'examen en domaines indépendants signifie que la sécurité, l'impact sur les performances et la couverture de test reçoivent tous une attention approfondie simultanément. Le prompt assigne à chaque coéquipier une lentille distincte pour qu'ils ne se chevauchent pas :

```
Créez une équipe d'agents pour examiner la PR #142. Générez trois examinateurs :  
- Un axé sur les implications de sécurité  
- Un vérifiant l'impact sur les performances  
- Un validant la couverture de test  
Demandez-leur d'examiner et de signaler les conclusions.
```

Chaque examinateur travaille à partir de la même PR mais applique un filtre différent. Le chef synthétise les conclusions de tous les trois après qu'ils aient terminé.

Enquêter avec des hypothèses concurrentes

Lorsque la cause première est peu claire, un seul agent tend à trouver une explication plausible et s'arrête. Le prompt combat cela en rendant les coéquipiers explicitement adversaires : le travail de chacun n'est pas seulement d'enquêter sur sa propre théorie mais de contester les autres.

Les utilisateurs signalent que l'application se ferme après un message au lieu de rester connectée.

Générez 5 coéquipiers agents pour enquêter sur différentes hypothèses. Demandez-leur de se parler pour essayer de réfuter les théories les uns des autres, comme un débat scientifique. Mettez à jour le document des conclusions avec le consensus qui émerge.

La structure du débat est le mécanisme clé ici. L'enquête séquentielle souffre de l'ancrage : une fois qu'une théorie est explorée, l'enquête ultérieure est biaisée vers elle.

Avec plusieurs enquêteurs indépendants essayant activement de réfuter les uns les autres, la théorie qui survit est beaucoup plus susceptible d'être la cause première réelle.

Meilleures pratiques

Donner aux coéquipiers suffisamment de contexte

Les coéquipiers chargent automatiquement le contexte du projet, y compris CLAUDE.md, MCP servers et skills, mais ils n'héritent pas de l'historique de conversation du chef. Consultez [Contexte et communication](#) pour les détails. Incluez les détails spécifiques à la tâche dans le prompt de génération :

Générez un coéquipier examinateur de sécurité avec le prompt : « Examinez le module d'authentification à src/auth/ pour les vulnérabilités de sécurité. Concentrez-vous sur la gestion des tokens, la gestion des sessions et la validation des entrées. L'application utilise des tokens JWT stockés dans des cookies httpOnly. Signalez tout problème avec les évaluations de gravité. »

Choisir une taille d'équipe appropriée

Il n'y a pas de limite stricte au nombre de coéquipiers, mais des contraintes pratiques s'appliquent :

- **Les coûts des tokens augmentent linéairement** : chaque coéquipier a sa propre fenêtre de contexte et consomme des tokens indépendamment. Consultez les [coûts des tokens des équipes d'agents](#) pour les détails.

- **La surcharge de coordination augmente** : plus de coéquipiers signifie plus de communication, de coordination de tâches et de risques de conflits
- **Rendements décroissants** : au-delà d'un certain point, les coéquipiers supplémentaires n'accroissent pas le travail proportionnellement

Commencez avec 3 à 5 coéquipiers pour la plupart des flux de travail. Cela équilibre le travail parallèle avec une coordination gérable. Les exemples de ce guide utilisent 3 à 5 coéquipiers car cette plage fonctionne bien dans différents types de tâches.

Avoir 5 à 6 [tâches](#) par coéquipier garde tout le monde productif sans changement de contexte excessif. Si vous avez 15 tâches indépendantes, 3 coéquipiers est un bon point de départ.

Augmentez l'échelle uniquement lorsque le travail bénéficie véritablement d'avoir des coéquipiers travaillant simultanément. Trois coéquipiers ciblés surpassent souvent cinq dispersés.

Dimensionner les tâches de manière appropriée

- **Trop petites** : la surcharge de coordination dépasse le bénéfice
- **Trop grandes** : les coéquipiers travaillent trop longtemps sans points de contrôle, augmentant le risque d'effort gaspillé
- **Juste bien** : des unités autonomes qui produisent un livrable clair, comme une fonction, un fichier de test ou un examen

Tip:

Le chef divise le travail en tâches et les assigne aux coéquipiers automatiquement. S'il ne crée pas assez de tâches, demandez-lui de diviser le travail en morceaux plus petits. Avoir 5 à 6 tâches par coéquipier garde tout le monde productif et permet au chef de réassigner le travail si quelqu'un est bloqué.

Attendre que les coéquipiers terminent

Parfois, le chef commence à mettre en œuvre des tâches lui-même au lieu d'attendre les coéquipiers. Si vous remarquez cela :

Attendez que vos coéquipiers complètent leurs tâches avant de procéder

Commencer par la recherche et l'examen

Si vous êtes nouveau aux équipes d'agents, commencez par des tâches qui ont des limites claires et ne nécessitent pas d'écrire du code : examiner une PR, rechercher une bibliothèque ou enquêter sur un bug. Ces tâches montrent la valeur de l'exploration parallèle sans les défis de coordination qui accompagnent la mise en œuvre parallèle.

Éviter les conflits de fichiers

Deux coéquipiers éditant le même fichier entraîne des écrasements. Divisez le travail pour que chaque coéquipier possède un ensemble de fichiers différent.

Surveiller et diriger

Vérifiez la progression des coéquipiers, redirigez les approches qui ne fonctionnent pas et synthétisez les conclusions au fur et à mesure qu'elles arrivent. Laisser une équipe s'exécuter sans surveillance pendant trop longtemps augmente le risque d'effort gaspillé.

Dépannage

Les coéquipiers n'apparaissent pas

Si les coéquipiers n'apparaissent pas après avoir demandé à Claude de créer une équipe :

- En mode in-process, les coéquipiers peuvent déjà être en cours d'exécution mais non visibles. Appuyez sur Maj+Bas pour parcourir les coéquipiers actifs.
- Vérifiez que la tâche que vous avez donnée à Claude était suffisamment complexe pour justifier une équipe. Claude décide s'il faut générer des coéquipiers en fonction de la tâche.
- Si vous avez explicitement demandé des volets divisés, assurez-vous que tmux est installé et disponible dans votre PATH :

```
which tmux
```

- Pour iTerm2, vérifiez que le CLI `it2` est installé et que l'API Python est activée dans les préférences d'iTerm2.

Trop de demandes de permission

Les demandes de permission des coéquipiers remontent au chef, ce qui peut créer des frictions. Pré-approuvez les opérations courantes dans vos [paramètres de permission](#) avant de générer les coéquipiers pour réduire les interruptions.

Les coéquipiers s'arrêtent sur les erreurs

Les coéquipiers peuvent s'arrêter après avoir rencontré des erreurs au lieu de se rétablir. Vérifiez leur sortie en utilisant Maj+Bas en mode in-process ou en cliquant sur le volet en mode divisé, puis :

- Donnez-leur des instructions supplémentaires directement
- Générez un coéquipier de remplacement pour continuer le travail

Le chef s'arrête avant que le travail ne soit terminé

Le chef peut décider que l'équipe est terminée avant que toutes les tâches ne soient réellement complètes. Si cela se produit, dites-lui de continuer. Vous pouvez également dire au chef d'attendre que les coéquipiers terminent avant de procéder s'il commence à faire du travail au lieu de déléguer.

Sessions tmux orphelines

Si une session tmux persiste après la fin de l'équipe, elle peut ne pas avoir été complètement nettoyée. Listez les sessions et tuez celle créée par l'équipe :

```
tmux ls
tmux kill-session -t <session-name>
```

Limitations

Les équipes d'agents sont expérimentales. Les limitations actuelles à connaître :

- **Pas de reprise de session avec les coéquipiers in-process** : `/resume` et `/rewind` ne restaurent pas les coéquipiers in-process. Après la reprise d'une session, le chef peut tenter de envoyer un message aux coéquipiers qui n'existent plus. Si cela se produit, dites au chef de générer de nouveaux coéquipiers.
- **L'état des tâches peut être en retard** : les coéquipiers échouent parfois à marquer les tâches comme complètes, ce qui bloque les tâches dépendantes. Si une tâche semble bloquée, vérifiez si le travail est réellement terminé et mettez à jour l'état de la tâche manuellement ou dites au chef de pousser le coéquipier.
- **L'arrêt peut être lent** : les coéquipiers terminent leur demande actuelle ou appel d'outil avant de s'arrêter, ce qui peut prendre du temps.
- **Une équipe par session** : un chef ne peut gérer qu'une seule équipe à la fois. Nettoyez l'équipe actuelle avant de démarrer une nouvelle.

- **Pas d'équipes imbriquées** : les coéquipiers ne peuvent pas générer leurs propres équipes ou coéquipiers. Seul le chef peut gérer l'équipe.
- **Le chef est fixe** : la session qui crée l'équipe est le chef pour sa durée de vie. Vous ne pouvez pas promouvoir un coéquipier en chef ou transférer le leadership.
- **Permissions définies au moment de la génération** : tous les coéquipiers commencent avec le mode de permission du chef. Vous pouvez modifier les modes de coéquipiers individuels après la génération, mais vous ne pouvez pas définir les modes par coéquipier au moment de la génération.
- **Les volets divisés nécessitent tmux ou iTerm2** : le mode in-process par défaut fonctionne dans n'importe quel terminal. Le mode volets divisés n'est pas supporté dans le terminal intégré de VS Code, Windows Terminal ou Ghostty.

Tip:

CLAUDE.md fonctionne normalement : les coéquipiers lisent les fichiers **CLAUDE.md** de leur répertoire de travail. Utilisez ceci pour fournir des conseils spécifiques au projet à tous les coéquipiers.

Prochaines étapes

Explorez les approches connexes pour le travail parallèle et la délégation :

- **Délégation légère** : les [subagents](#) génèrent des agents auxiliaires pour la recherche ou la vérification au sein de votre session, mieux pour les tâches qui n'ont pas besoin de coordination inter-agents
- **Sessions parallèles manuelles** : les [Git worktrees](#) vous permettent d'exécuter plusieurs sessions Claude Code vous-même sans coordination d'équipe automatisée
- **Comparer les approches** : consultez la comparaison [subagent vs équipe d'agents](#) pour une répartition côte à côte

Exécuter Claude Code par programmation

Utilisez l'Agent SDK pour exécuter Claude Code par programmation depuis la CLI, Python ou TypeScript.

L'[Agent SDK](#) vous donne accès aux mêmes outils, boucle d'agent et gestion du contexte qui alimentent Claude Code. Il est disponible en tant que CLI pour les scripts et CI/CD, ou en tant que packages [Python](#) et [TypeScript](#) pour un contrôle programmatique complet.

Note:

La CLI s'appelait auparavant « mode sans interface ». Le flag `-p` et toutes les options CLI fonctionnent de la même manière.

Pour exécuter Claude Code par programmation depuis la CLI, passez `-p` avec votre prompt et toute [option CLI](#) :

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

Cette page couvre l'utilisation de l'Agent SDK via la CLI (`claude -p`). Pour les packages SDK Python et TypeScript avec sorties structurées, callbacks d'approbation d'outils et objets de message natifs, consultez la [documentation complète de l'Agent SDK](#).

Utilisation basique

Ajoutez le flag `-p` (ou `--print`) à n'importe quelle commande `claude` pour l'exécuter de manière non-interactive. Toutes les [options CLI](#) fonctionnent avec `-p`, notamment :

- `--continue` pour [continuer les conversations](#)
- `--allowedTools` pour [approuver automatiquement les outils](#)
- `--output-format` pour [obtenir une sortie structurée](#)

Cet exemple pose une question à Claude sur votre base de code et affiche la réponse :

```
claude -p "What does the auth module do?"
```

Exemples

Ces exemples mettent en évidence les modèles CLI courants.

Obtenir une sortie structurée

Utilisez `--output-format` pour contrôler la façon dont les réponses sont retournées :

- `text` (par défaut) : sortie en texte brut
- `json` : JSON structuré avec résultat, ID de session et métadonnées
- `stream-json` : JSON délimité par des sauts de ligne pour le streaming en temps réel

Cet exemple retourne un résumé du projet au format JSON avec les métadonnées de session, avec le résultat textuel dans le champ `result` :

```
claude -p "Summarize this project" --output-format json
```

Pour obtenir une sortie conforme à un schéma spécifique, utilisez `--output-format json` avec `--json-schema` et une définition [JSON Schema](#). La réponse inclut les métadonnées sur la requête (ID de session, utilisation, etc.) avec la sortie structurée dans le champ `structured_output`.

Cet exemple extrait les noms de fonctions et les retourne sous forme de tableau de chaînes :

```
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:

Utilisez un outil comme [jq](#) pour analyser la réponse et extraire des champs spécifiques :

```
## 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'
```

Diffuser les réponses

Utilisez `--output-format stream-json` avec `--verbose` et `--include-partial-messages` pour recevoir les tokens au fur et à mesure qu'ils sont générés. Chaque ligne est un objet JSON représentant un événement :

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-
partial-messages
```

L'exemple suivant utilise `jq` pour filtrer les deltas de texte et afficher uniquement le texte en streaming. Le flag `-r` affiche les chaînes brutes (sans guillemets) et `-j` joint sans sauts de ligne pour que les tokens se diffusent continuellement :

```
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'
```

Pour le streaming programmatique avec callbacks et objets de message, consultez [Diffuser les réponses en temps réel](#) dans la documentation de l'Agent SDK.

Approuver automatiquement les outils

Utilisez `--allowedTools` pour permettre à Claude d'utiliser certains outils sans demander. Cet exemple exécute une suite de tests et corrige les défaillances, permettant à Claude d'exécuter des commandes Bash et de lire/modifier des fichiers sans demander la permission :

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

Créer un commit

Cet exemple examine les modifications mises en scène et crée un commit avec un message approprié :

```
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 *)"
```

Le flag `--allowedTools` utilise la [syntaxe des règles de permission](#). Le `*` final active la correspondance de préfixe, donc `Bash(git diff *)` autorise n'importe quelle commande commençant par `git diff`. L'espace avant `*` est important : sans lui, `Bash(git diff*)` correspondrait également à `git diff-index`.

Note:

Les [skills](#) invoquées par l'utilisateur comme `/commit` et les [commandes intégrées](#) ne sont disponibles qu'en mode interactif. En mode `-p`, décrivez plutôt la tâche que vous souhaitez accomplir.

Personnaliser le prompt système

Utilisez `--append-system-prompt` pour ajouter des instructions tout en conservant le comportement par défaut de Claude Code. Cet exemple envoie un diff de PR à Claude et lui demande de vérifier les vulnérabilités de sécurité :

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for
  vulnerabilities." \
  --output-format json
```

Consultez les [flags de prompt système](#) pour plus d'options, notamment `--system-prompt` pour remplacer complètement le prompt par défaut.

Continuer les conversations

Utilisez `--continue` pour continuer la conversation la plus récente, ou `--resume` avec un ID de session pour continuer une conversation spécifique. Cet exemple exécute un examen, puis envoie des prompts de suivi :

```
## 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
```

Si vous exécutez plusieurs conversations, capturez l'ID de session pour reprendre une conversation spécifique :

```
session_id=$(claude -p "Start a review" --output-format json | jq -r
'.session_id')
claude -p "Continue that review" --resume "$session_id"
```

Étapes suivantes

- [Démarrage rapide de l'Agent SDK](#) : créez votre premier agent avec Python ou TypeScript
- [Référence CLI](#) : tous les flags et options CLI
- [GitHub Actions](#) : utilisez l'Agent SDK dans les workflows GitHub
- [GitLab CI/CD](#) : utilisez l'Agent SDK dans les pipelines GitLab

Continuer les sessions locales depuis n'importe quel appareil avec Remote Control

Continuez une session Claude Code locale depuis votre téléphone, tablette ou n'importe quel navigateur en utilisant Remote Control. Fonctionne avec claude.ai/code et l'application Claude mobile.

Note:

Remote Control est disponible sur tous les plans. Les administrateurs Team et Enterprise doivent d'abord activer Claude Code dans les [paramètres d'administration](#).

Remote Control connecte claude.ai/code ou l'application Claude pour [iOS](#) et [Android](#) à une session Claude Code s'exécutant sur votre machine. Démarrez une tâche à votre bureau, puis reprenez-la depuis votre téléphone sur le canapé ou un navigateur sur un autre ordinateur.

Lorsque vous démarrez une session Remote Control sur votre machine, Claude continue de s'exécuter localement à tout moment, donc rien ne se déplace vers le cloud. Avec Remote Control, vous pouvez :

- **Utiliser votre environnement local complet à distance** : votre système de fichiers, [serveurs MCP](#), outils et configuration de projet restent tous disponibles
- **Travailler depuis les deux surfaces à la fois** : la conversation reste synchronisée sur tous les appareils connectés, vous pouvez donc envoyer des messages depuis votre terminal, navigateur et téléphone de manière interchangeable
- **Survivre aux interruptions** : si votre ordinateur portable s'endort ou votre réseau tombe en panne, la session se reconnecte automatiquement lorsque votre machine revient en ligne

Contrairement à [Claude Code sur le web](#), qui s'exécute sur l'infrastructure cloud, les sessions Remote Control s'exécutent directement sur votre machine et interagissent avec votre système de fichiers local. Les interfaces web et mobile ne sont qu'une fenêtre dans cette session locale.

Note:

Remote Control nécessite Claude Code v2.1.51 ou version ultérieure. Vérifiez votre version avec `claude --version`.

Cette page couvre la configuration, comment démarrer et se connecter aux sessions, et comment Remote Control se compare à Claude Code sur le web.

Conditions requises

Avant d'utiliser Remote Control, confirmez que votre environnement répond à ces conditions :

- **Abonnement** : disponible sur les plans Pro, Max, Team et Enterprise. Les administrateurs Team et Enterprise doivent d'abord activer Claude Code dans les [paramètres d'administration](#). Les clés API ne sont pas prises en charge.
- **Authentification** : exécutez `claude` et utilisez `/login` pour vous connecter via `claude.ai` si vous ne l'avez pas déjà fait.
- **Confiance de l'espace de travail** : exécutez `claude` dans votre répertoire de projet au moins une fois pour accepter la boîte de dialogue de confiance de l'espace de travail.

Démarrer une session Remote Control

Vous pouvez démarrer une nouvelle session directement dans Remote Control, ou connecter une session qui s'exécute déjà.

Nouvelle session

Accédez à votre répertoire de projet et exécutez :

```
claude remote-control
```

Le processus continue de s'exécuter dans votre terminal, en attente de connexions distantes. Il affiche une URL de session que vous pouvez utiliser pour [vous connecter depuis un autre appareil](#), et vous pouvez appuyer sur la barre d'espace pour afficher un code QR pour un accès rapide depuis votre téléphone. Pendant qu'une session distante est active, le terminal affiche l'état de la connexion et l'activité des outils.

Cette commande prend en charge les drapeaux suivants :

- `--name "My Project"` : définir un titre de session personnalisé visible dans la liste des sessions sur `claude.ai/code`. Vous pouvez également passer le nom comme argument positionnel : `claude remote-control "My Project"`

- `--verbose` : afficher les journaux détaillés de connexion et de session
- `--sandbox` / `--no-sandbox` : activer ou désactiver le [sandboxing](#) pour l'isolation du système de fichiers et du réseau pendant la session. Le sandboxing est désactivé par défaut.

À partir d'une session existante

Si vous êtes déjà dans une session Claude Code et que vous souhaitez la continuer à distance, utilisez la commande `/remote-control` (ou `/rc`) :

```
/remote-control
```

Passez un nom comme argument pour définir un titre de session personnalisé :

```
/remote-control My Project
```

Cela démarre une session Remote Control qui reprend votre historique de conversation actuel et affiche une URL de session et un code QR que vous pouvez utiliser pour [vous connecter depuis un autre appareil](#). Les drapeaux `--verbose`, `--sandbox` et `--no-sandbox` ne sont pas disponibles avec cette commande.

Se connecter depuis un autre appareil

Une fois qu'une session Remote Control est active, vous avez plusieurs façons de vous connecter depuis un autre appareil :

- **Ouvrir l'URL de session** dans n'importe quel navigateur pour accéder directement à la session sur [claude.ai/code](#). À la fois `claude remote-control` et `/remote-control` affichent cette URL dans le terminal.
- **Scannez le code QR** affiché à côté de l'URL de session pour l'ouvrir directement dans l'application Claude. Avec `claude remote-control`, appuyez sur la barre d'espace pour basculer l'affichage du code QR.
- **Ouvrir [claude.ai/code](#) ou l'application Claude** et trouver la session par nom dans la liste des sessions. Les sessions Remote Control affichent une icône d'ordinateur avec un point d'état vert lorsqu'elles sont en ligne.

La session distante prend son nom à partir de l'argument `--name` (ou du nom passé à `/remote-control`), votre dernier message, votre valeur `/rename`, ou « Remote Control session » s'il n'y a pas d'historique de conversation. Si l'environnement a déjà une session active, vous serez invité à choisir si vous souhaitez la continuer ou en démarrer une nouvelle.

Si vous n'avez pas encore l'application Claude, utilisez la commande `/mobile` dans Claude Code pour afficher un code QR de téléchargement pour [iOS](#) ou [Android](#).

Activer Remote Control pour toutes les sessions

Par défaut, Remote Control ne s'active que lorsque vous exécutez explicitement `claude remote-control` ou `/remote-control`. Pour l'activer automatiquement pour chaque session, exécutez `/config` dans Claude Code et définissez **Enable Remote Control for all sessions** sur `true`. Définissez-le sur `false` pour désactiver.

Chaque instance Claude Code prend en charge une session distante à la fois. Si vous exécutez plusieurs instances, chacune obtient son propre environnement et sa propre session.

Connexion et sécurité

Votre session Claude Code locale effectue uniquement des requêtes HTTPS sortantes et n'ouvre jamais de ports entrants sur votre machine. Lorsque vous démarrez Remote Control, il s'enregistre auprès de l'API Anthropic et interroge le travail. Lorsque vous vous connectez depuis un autre appareil, le serveur achemine les messages entre le client web ou mobile et votre session locale sur une connexion en continu.

Tout le trafic passe par l'API Anthropic sur TLS, le même transport de sécurité que n'importe quelle session Claude Code. La connexion utilise plusieurs identifiants de courte durée, chacun limité à un seul objectif et expirant indépendamment.

Remote Control vs Claude Code sur le web

Remote Control et [Claude Code sur le web](#) utilisent tous deux l'interface `claude.ai/code`. La différence clé est l'endroit où la session s'exécute : Remote Control s'exécute sur votre machine, donc vos serveurs MCP locaux, outils et configuration de projet restent disponibles. Claude Code sur le web s'exécute dans l'infrastructure cloud gérée par Anthropic.

Utilisez Remote Control lorsque vous êtes au milieu d'un travail local et que vous souhaitez continuer depuis un autre appareil. Utilisez Claude Code sur le web lorsque vous souhaitez lancer une tâche sans aucune configuration locale, travailler sur un référentiel que vous n'avez pas cloné, ou exécuter plusieurs tâches en parallèle.

Limitations

- **Une session distante à la fois** : chaque session Claude Code prend en charge une connexion distante.

- **Le terminal doit rester ouvert** : Remote Control s'exécute en tant que processus local. Si vous fermez le terminal ou arrêtez le processus `claude`, la session se termine. Exécutez `claude remote-control` à nouveau pour démarrer une nouvelle.
- **Panne réseau prolongée** : si votre machine est allumée mais incapable d'atteindre le réseau pendant plus de dix minutes environ, la session expire et le processus se termine. Exécutez `claude remote-control` à nouveau pour démarrer une nouvelle session.

Ressources connexes

- [Claude Code sur le web](#) : exécuter des sessions dans des environnements cloud gérés par Anthropic au lieu de sur votre machine
- [Authentification](#) : configurer `/login` et gérer les identifiants pour claude.ai
- [Référence CLI](#) : liste complète des drapeaux et commandes incluant `claude remote-control`
- [Sécurité](#) : comment les sessions Remote Control s'intègrent dans le modèle de sécurité Claude Code
- [Utilisation des données](#) : quelles données circulent via l'API Anthropic pendant les sessions locales et distantes

Exécuter des prompts selon un calendrier

Utilisez `/loop` et les outils de planification cron pour exécuter des prompts de manière répétée, interroger l'état ou définir des rappels ponctuels dans une session Claude Code.

Note:

Les tâches planifiées nécessitent Claude Code v2.1.72 ou version ultérieure. Vérifiez votre version avec `claude --version`.

Les tâches planifiées permettent à Claude de réexécuter automatiquement un prompt à intervalles réguliers. Utilisez-les pour interroger un déploiement, surveiller une PR, vérifier une compilation longue ou vous rappeler de faire quelque chose plus tard dans la session.

Les tâches sont limitées à la session : elles vivent dans le processus Claude Code actuel et disparaissent lorsque vous quittez. Pour une planification durable qui survit aux redémarrages et s'exécute sans session de terminal active, consultez [Tâches planifiées sur le Bureau](#) ou [GitHub Actions](#).

Planifier un prompt récurrent avec `/loop`

La [compétence groupée](#) `/loop` est le moyen le plus rapide de planifier un prompt récurrent. Passez un intervalle optionnel et un prompt, et Claude configure une tâche cron qui s'exécute en arrière-plan pendant que la session reste ouverte.

```
/loop 5m check if the deployment finished and tell me what happened
```

Claude analyse l'intervalle, le convertit en expression cron, planifie la tâche et confirme la cadence et l'ID de la tâche.

Syntaxe d'intervalle

Les intervalles sont optionnels. Vous pouvez les placer au début, à la fin ou les omettre entièrement.

Forme	Exemple	Intervalle analysé
Jeton initial	<code>/loop 30m check the build</code>	toutes les 30 minutes
Clause <code>every</code> finale	<code>/loop check the build every 2 hours</code>	toutes les 2 heures
Pas d'intervalle	<code>/loop check the build</code>	par défaut toutes les 10 minutes

Les unités supportées sont `s` pour les secondes, `m` pour les minutes, `h` pour les heures et `d` pour les jours. Les secondes sont arrondies à la minute la plus proche puisque cron a une granularité d'une minute. Les intervalles qui ne se divisent pas uniformément dans leur unité, comme `7m` ou `90m`, sont arrondis à l'intervalle le plus proche et Claude vous indique ce qu'il a choisi.

Boucler sur une autre commande

Le prompt planifié peut lui-même être une invocation de commande ou de compétence. C'est utile pour réexécuter un flux de travail que vous avez déjà empaqueté.

```
/loop 20m /review-pr 1234
```

Chaque fois que la tâche s'exécute, Claude exécute `/review-pr 1234` comme si vous l'aviez tapé.

Définir un rappel ponctuel

Pour les rappels ponctuels, décrivez ce que vous voulez en langage naturel au lieu d'utiliser `/loop`. Claude planifie une tâche à exécution unique qui se supprime après son exécution.

```
remind me at 3pm to push the release branch
```

```
in 45 minutes, check whether the integration tests passed
```

Claude épingle l'heure d'exécution à une minute et une heure spécifiques en utilisant une expression cron et confirme quand elle s'exécutera.

Gérer les tâches planifiées

Demandez à Claude en langage naturel de lister ou d’annuler les tâches, ou référencez directement les outils sous-jacents.

```
what scheduled tasks do I have?
```

```
cancel the deploy check job
```

Sous le capot, Claude utilise ces outils :

Outil	Objectif
<code>CronCreate</code>	Planifier une nouvelle tâche. Accepte une expression cron à 5 champs, le prompt à exécuter et si elle est récurrente ou s’exécute une seule fois.
<code>CronList</code>	Lister toutes les tâches planifiées avec leurs ID, leurs calendriers et leurs prompts.
<code>CronDelete</code>	Annuler une tâche par ID.

Chaque tâche planifiée a un ID de 8 caractères que vous pouvez passer à `CronDelete`. Une session peut contenir jusqu’à 50 tâches planifiées à la fois.

Comment les tâches planifiées s’exécutent

Le planificateur vérifie chaque seconde les tâches dues et les met en file d’attente à faible priorité. Un prompt planifié s’exécute entre vos tours, pas pendant que Claude répond. Si Claude est occupé quand une tâche arrive à échéance, le prompt attend la fin du tour actuel.

Tous les horaires sont interprétés dans votre fuseau horaire local. Une expression cron comme `0 9 * * *` signifie 9h là où vous exécutez Claude Code, pas UTC.

Gigue

Pour éviter que chaque session ne frappe l’API au même moment mural, le planificateur ajoute un petit décalage déterministe aux heures d’exécution :

- Les tâches récurrentes s’exécutent jusqu’à 10 % de leur période en retard, plafonnées à 15 minutes. Une tâche horaire peut s’exécuter n’importe où de `:00` à `:06`.

- Les tâches ponctuelles planifiées pour le haut ou le bas de l'heure s'exécutent jusqu'à 90 secondes plus tôt.

Le décalage est dérivé de l'ID de la tâche, donc la même tâche obtient toujours le même décalage. Si le timing exact est important, choisissez une minute qui n'est pas `:00` ou `:30`, par exemple `3 9 * * *` au lieu de `0 9 * * *`, et la gigue ponctuelle ne s'appliquera pas.

Expiration de trois jours

Les tâches récurrentes expirent automatiquement 3 jours après leur création. La tâche s'exécute une dernière fois, puis se supprime. Cela limite la durée pendant laquelle une boucle oubliée peut s'exécuter. Si vous avez besoin qu'une tâche récurrente dure plus longtemps, annulez et recréez-la avant son expiration, ou utilisez [Tâches planifiées sur le Bureau](#) pour une planification durable.

Référence d'expression cron

`CronCreate` accepte les expressions cron standard à 5 champs : `minute heure jour-du-mois mois jour-de-la-semaine`. Tous les champs supportent les caractères génériques (`*`), les valeurs uniques (`5`), les étapes (`*/15`), les plages (`1-5`) et les listes séparées par des virgules (`1,15,30`).

Exemple	Signification
<code>* / 5 * * * *</code>	Toutes les 5 minutes
<code>0 * * * *</code>	Chaque heure à l'heure pile
<code>7 * * * *</code>	Chaque heure à 7 minutes passées
<code>0 9 * * *</code>	Chaque jour à 9h local
<code>0 9 * * 1-5</code>	Les jours de semaine à 9h local
<code>30 14 15 3 *</code>	15 mars à 14h30 local

Le jour de la semaine utilise `0` ou `7` pour dimanche jusqu'à `6` pour samedi. La syntaxe étendue comme `L`, `W`, `?` et les alias de noms tels que `MON` ou `JAN` ne sont pas supportés.

Quand le jour du mois et le jour de la semaine sont tous deux contraints, une date correspond si l'un ou l'autre champ correspond. Cela suit la sémantique standard de vixie-cron.

Désactiver les tâches planifiées

Définissez `CLAUDE_CODE_DISABLE_CRON=1` dans votre environnement pour désactiver complètement le planificateur. Les outils `cron` et `/Loop` deviennent indisponibles, et toutes les tâches déjà planifiées cessent de s'exécuter. Consultez [Variables d'environnement](#) pour la liste complète des drapeaux de désactivation.

Limitations

La planification limitée à la session a des contraintes inhérentes :

- Les tâches ne s'exécutent que pendant que Claude Code s'exécute et est inactif. Fermer le terminal ou laisser la session se terminer annule tout.
- Pas de rattrapage pour les exécutions manquées. Si l'heure planifiée d'une tâche passe pendant que Claude est occupé par une requête longue, elle s'exécute une fois quand Claude devient inactif, pas une fois par intervalle manqué.
- Pas de persistance entre les redémarrages. Redémarrer Claude Code efface toutes les tâches limitées à la session.

Pour l'automatisation pilotée par `cron` qui doit s'exécuter sans surveillance, utilisez un [flux de travail GitHub Actions](#) avec un déclencheur `schedule`, ou [Tâches planifiées sur le Bureau](#) si vous voulez un flux de configuration graphique.

Révision de code

Configurez des révisions de PR automatisées qui détectent les erreurs logiques, les vulnérabilités de sécurité et les régressions en utilisant une analyse multi-agents de votre base de code complète

Note:

Code Review est en aperçu de recherche, disponible pour les abonnements [Teams et Enterprise](#). Il n'est pas disponible pour les organisations avec [Zero Data Retention](#) activé.

Code Review analyse vos pull requests GitHub et publie les résultats sous forme de commentaires en ligne sur les lignes de code où il a trouvé des problèmes. Une flotte d'agents spécialisés examine les modifications de code dans le contexte de votre base de code complète, en recherchant les erreurs logiques, les vulnérabilités de sécurité, les cas limites cassés et les régressions subtiles.

Les résultats sont étiquetés par gravité et n'approuvent ni ne bloquent votre PR, de sorte que les flux de travail d'examen existants restent intacts. Vous pouvez affiner ce que Claude signale en ajoutant un fichier `CLAUDE.md` ou `REVIEW.md` à votre référentiel.

Pour exécuter Claude dans votre propre infrastructure CI au lieu de ce service géré, consultez [GitHub Actions](#) ou [GitLab CI/CD](#).

Cette page couvre :

- [Comment fonctionnent les révisions](#)
- [Configuration](#)
- [Personnalisation des révisions](#) avec `CLAUDE.md` et `REVIEW.md`
- [Tarification](#)

Comment fonctionnent les révisions

Une fois qu'un administrateur [active Code Review](#) pour votre organisation, les révisions s'exécutent automatiquement lorsqu'une pull request s'ouvre ou se met à jour. Plusieurs agents analysent le diff et le code environnant en parallèle sur l'infrastructure Anthropic. Chaque agent recherche une classe de problème différente, puis une étape de vérification vérifie les candidats par rapport au comportement réel du code pour filtrer les faux

positifs. Les résultats sont dédoublés, classés par gravité et publiés sous forme de commentaires en ligne sur les lignes spécifiques où les problèmes ont été trouvés. Si aucun problème n'est trouvé, Claude publie un court commentaire de confirmation sur la PR.

Les révisions s'adaptent en coût à la taille et à la complexité de la PR, se complétant en moyenne en 20 minutes. Les administrateurs peuvent surveiller l'activité de révision et les dépenses via le [tableau de bord analytique](#).

Niveaux de gravité

Chaque résultat est étiqueté avec un niveau de gravité :

Marqueur	Gravité	Signification
	Normal	Un bug qui devrait être corrigé avant la fusion
	Nit	Un problème mineur, utile à corriger mais non bloquant
	Préexistant	Un bug qui existe dans la base de code mais n'a pas été introduit par cette PR

Les résultats incluent une section de raisonnement étendu réductible que vous pouvez développer pour comprendre pourquoi Claude a signalé le problème et comment il a vérifié le problème.

Ce que Code Review vérifie

Par défaut, Code Review se concentre sur la correction : les bugs qui cassent la production, pas les préférences de formatage ou la couverture de test manquante. Vous pouvez développer ce qu'il vérifie en [ajoutant des fichiers de guidance](#) à votre référentiel.

Configurer Code Review

Un administrateur active Code Review une fois pour l'organisation et sélectionne les référentiels à inclure.

Step 1: Ouvrir les paramètres d'administration Claude Code

Allez à [claude.ai/admin-settings/claude-code](#) et trouvez la section Code Review. Vous avez besoin d'un accès administrateur à votre organisation Claude et de la permission d'installer des GitHub Apps dans votre organisation GitHub.

Step 2: Démarrer la configuration

Cliquez sur **Configuration**. Cela commence le flux d'installation de GitHub App.

Step 3: Installer la GitHub App Claude

Suivez les invites pour installer la GitHub App Claude à votre organisation GitHub. L'application demande ces permissions de référentiel :

- **Contents** : lecture et écriture
- **Issues** : lecture et écriture
- **Pull requests** : lecture et écriture

Code Review utilise l'accès en lecture aux contenus et l'accès en écriture aux pull requests. L'ensemble de permissions plus large supporte également [GitHub Actions](#) si vous l'activez plus tard.

Step 4: Sélectionner les référentiels

Choisissez les référentiels à activer pour Code Review. Si vous ne voyez pas un référentiel, assurez-vous d'avoir donné à la GitHub App Claude l'accès à celui-ci lors de l'installation. Vous pouvez ajouter d'autres référentiels plus tard.

Step 5: Définir les déclencheurs de révision par référentiel

Une fois la configuration terminée, la section Code Review affiche vos référentiels dans un tableau. Pour chaque référentiel, utilisez la liste déroulante pour choisir quand les révisions s'exécutent :

- **Après la création de PR uniquement** : la révision s'exécute une fois lorsqu'une PR est ouverte ou marquée comme prête pour révision
- **Après chaque push vers la branche PR** : la révision s'exécute à chaque push, détectant les nouveaux problèmes au fur et à mesure que la PR évolue et résolvant automatiquement les threads lorsque vous corrigez les problèmes signalés

La révision à chaque push exécute plus de révisions et coûte plus cher. Commencez par la création de PR uniquement et passez à on-push pour les référentiels où vous voulez une couverture continue et un nettoyage automatique des threads.

Le tableau des référentiels affiche également le coût moyen par révision pour chaque référentiel en fonction de l'activité récente. Utilisez le menu des actions de ligne pour activer ou désactiver Code Review par référentiel, ou pour supprimer complètement un référentiel.

Pour vérifier la configuration, ouvrez une PR de test. Une exécution de vérification nommée **Claude Code Review** apparaît dans quelques minutes. Si ce n'est pas le cas, confirmez que le référentiel est listé dans vos paramètres d'administration et que la GitHub App Claude y a accès.

Personnaliser les révisions

Code Review lit deux fichiers de votre référentiel pour guider ce qu'il signale. Les deux s'ajoutent aux vérifications de correction par défaut :

- **CLAUDE.md** : instructions de projet partagées que Claude Code utilise pour toutes les tâches, pas seulement les révisions. Utilisez-le lorsque la guidance s'applique également aux sessions Claude Code interactives.
- **REVIEW.md** : guidance réservée à la révision, lue exclusivement lors des révisions de code. Utilisez-le pour les règles qui concernent strictement ce qu'il faut signaler ou ignorer lors de la révision et qui encombreraient votre **CLAUDE.md** général.

CLAUDE.md

Code Review lit vos fichiers **CLAUDE.md** du référentiel et traite les violations nouvellement introduites comme des résultats au niveau nit. Cela fonctionne bidirectionnellement : si votre PR modifie le code d'une manière qui rend une déclaration **CLAUDE.md** obsolète, Claude signale que la documentation doit être mise à jour aussi.

Claude lit les fichiers **CLAUDE.md** à chaque niveau de votre hiérarchie de répertoires, de sorte que les règles dans le **CLAUDE.md** d'un sous-répertoire s'appliquent uniquement aux fichiers sous ce chemin. Consultez la [documentation de mémoire](#) pour plus d'informations sur le fonctionnement de **CLAUDE.md**.

Pour la guidance spécifique à la révision que vous ne voulez pas appliquer aux sessions Claude Code générales, utilisez **REVIEW.md** à la place.

REVIEW.md

Ajoutez un fichier **REVIEW.md** à la racine de votre référentiel pour les règles spécifiques à la révision. Utilisez-le pour encoder :

- Directives de style de l'entreprise ou de l'équipe : « préférer les retours précoces aux conditionnels imbriqués »
- Conventions spécifiques au langage ou au framework non couvertes par les linters
- Choses que Claude devrait toujours signaler : « tout nouvel itinéraire API doit avoir un test d'intégration »

- Choses que Claude devrait ignorer : « ne pas commenter le formatage dans le code généré sous `/gen/` »

Exemple `REVIEW.md` :

```
## Directives de révision de code

### Toujours vérifier
- Les nouveaux points de terminaison API ont des tests d'intégration correspondants
- Les migrations de base de données sont rétrocompatibles
- Les messages d'erreur ne divulguent pas les détails internes aux utilisateurs

### Style
- Préférer les déclarations `match` aux vérifications `isinstance` chaînées
- Utiliser la journalisation structurée, pas l'interpolation de chaîne f dans les appels de journal

### Ignorer
- Fichiers générés sous `src/gen/`
- Modifications de formatage uniquement dans les fichiers `*.lock`
```

Claude découvre automatiquement `REVIEW.md` à la racine du référentiel. Aucune configuration nécessaire.

Afficher l'utilisation

Allez à claude.ai/analytics/code-review pour voir l'activité de Code Review dans votre organisation. Le tableau de bord affiche :

Section	Ce qu'il affiche
PRs révisées	Nombre quotidien de pull requests révisées sur la plage de temps sélectionnée
Coût hebdomadaire	Dépenses hebdomadaires sur Code Review
Feedback	Nombre de commentaires de révision qui ont été résolus automatiquement parce qu'un développeur a résolu le problème

Section	Ce qu'il affiche
Répartition par référentiel	Comptages par référentiel des PRs révisées et des commentaires résolus

Le tableau des référentiels dans les paramètres d'administration affiche également le coût moyen par révision pour chaque référentiel.

Tarification

Code Review est facturé en fonction de l'utilisation des tokens. Les révisions coûtent en moyenne 15 à 25 dollars, s'adaptant à la taille de la PR, à la complexité de la base de code et au nombre de problèmes nécessitant une vérification. L'utilisation de Code Review est facturée séparément via [l'utilisation supplémentaire](#) et ne compte pas par rapport à l'utilisation incluse de votre plan.

Le déclencheur de révision que vous choisissez affecte le coût total :

- **Après la création de PR uniquement** : s'exécute une fois par PR
- **Après chaque push** : s'exécute à chaque commit, multipliant le coût par le nombre de pushes

Les coûts apparaissent sur votre facture Anthropic indépendamment du fait que votre organisation utilise AWS Bedrock ou Google Vertex AI pour d'autres fonctionnalités Claude Code. Pour définir un plafond de dépenses mensuelles pour Code Review, allez à [claude.ai/admin-settings/usage](#) et configurez la limite pour le service Claude Code Review.

Surveillez les dépenses via le graphique des coûts hebdomadaires dans [analytics](#) ou la colonne de coût moyen par référentiel dans les paramètres d'administration.

Ressources connexes

Code Review est conçu pour fonctionner aux côtés du reste de Claude Code. Si vous voulez exécuter des révisions localement avant d'ouvrir une PR, avez besoin d'une configuration auto-hébergée ou voulez approfondir la façon dont `CLAUDE.md` façonne le comportement de Claude dans tous les outils, ces pages sont de bonnes prochaines étapes :

- [Plugins](#) : parcourez la place de marché des plugins, y compris un plugin `code-review` pour exécuter des révisions à la demande localement avant de pousser
- [GitHub Actions](#) : exécutez Claude dans vos propres flux de travail GitHub Actions pour une automatisation personnalisée au-delà de la révision de code
- [GitLab CI/CD](#) : intégration Claude auto-hébergée pour les pipelines GitLab

- [Memory](#) : comment les fichiers `CLAUDE.md` fonctionnent dans Claude Code
- [Analytics](#) : suivez l'utilisation de Claude Code au-delà de la révision de code

Part 7: CI/CD & Integrations

Claude Code GitHub Actions

Découvrez comment intégrer Claude Code dans votre flux de travail de développement avec Claude Code GitHub Actions

Claude Code GitHub Actions apporte l'automatisation alimentée par l'IA à votre flux de travail GitHub. Avec une simple mention `@claude` dans n'importe quelle PR ou issue, Claude peut analyser votre code, créer des pull requests, implémenter des fonctionnalités et corriger des bugs - tout en respectant les normes de votre projet. Pour les révisions automatiques publiées sur chaque PR sans déclencheur, consultez [GitHub Code Review](#).

Note:

Claude Code GitHub Actions est construit sur le [Claude Agent SDK](#), qui permet l'intégration programmatique de Claude Code dans vos applications. Vous pouvez utiliser le SDK pour créer des flux de travail d'automatisation personnalisés au-delà de GitHub Actions.

Info:

Claude Opus 4.6 est maintenant disponible. Claude Code GitHub Actions utilise par défaut Sonnet. Pour utiliser Opus 4.6, configurez le [paramètre model](#) pour utiliser `claude-opus-4-6`.

Pourquoi utiliser Claude Code GitHub Actions ?

- **Création instantanée de PR** : Décrivez ce dont vous avez besoin, et Claude crée une PR complète avec tous les changements nécessaires
- **Implémentation de code automatisée** : Transformez les issues en code fonctionnel avec une seule commande
- **Respecte vos normes** : Claude respecte vos directives `CLAUDE.md` et les modèles de code existants
- **Configuration simple** : Commencez en quelques minutes avec notre installateur et votre clé API
- **Sécurisé par défaut** : Votre code reste sur les runners de Github

Que peut faire Claude ?

Claude Code fournit une GitHub Action puissante qui transforme votre façon de travailler avec le code :

Claude Code Action

Cette GitHub Action vous permet d'exécuter Claude Code dans vos flux de travail GitHub Actions. Vous pouvez l'utiliser pour créer n'importe quel flux de travail personnalisé sur Claude Code.

[Voir le repository →](#)

Configuration

Configuration rapide

Le moyen le plus simple de configurer cette action est via Claude Code dans le terminal. Ouvrez simplement claude et exécutez `/install-github-app`.

Cette commande vous guidera à travers la configuration de l'application GitHub et des secrets requis.

Note:

- Vous devez être administrateur du repository pour installer l'application GitHub et ajouter des secrets
- L'application GitHub demandera des permissions de lecture et d'écriture pour Contents, Issues et Pull requests
- Cette méthode de démarrage rapide n'est disponible que pour les utilisateurs directs de l'API Claude. Si vous utilisez AWS Bedrock ou Google Vertex AI, veuillez consulter la section [Utilisation avec AWS Bedrock et Google Vertex AI](#).

Configuration manuelle

Si la commande `/install-github-app` échoue ou si vous préférez une configuration manuelle, veuillez suivre ces instructions de configuration manuelle :

1. **Installez l'application Claude GitHub** dans votre repository : <https://github.com/apps/claude>

L'application Claude GitHub nécessite les permissions de repository suivantes :

- **Contents** : Lecture et écriture (pour modifier les fichiers du repository)
- **Issues** : Lecture et écriture (pour répondre aux issues)

- **Pull requests** : Lecture et écriture (pour créer des PRs et pousser les changements)

Pour plus de détails sur la sécurité et les permissions, consultez la [documentation de sécurité](#).

2. **Ajoutez ANTHROPIC_API_KEY** à vos secrets de repository ([Apprenez comment utiliser les secrets dans GitHub Actions](#))
3. **Copiez le fichier de flux de travail** depuis [examples/claude.yml](#) dans le répertoire `.github/workflows/` de votre repository

Tip:

Après avoir complété la configuration rapide ou manuelle, testez l'action en marquant `@claude` dans un commentaire d'issue ou de PR.

Mise à niveau depuis la version bêta

Warning:

Claude Code GitHub Actions v1.0 introduit des changements majeurs qui nécessitent de mettre à jour vos fichiers de flux de travail pour passer de la version bêta à v1.0.

Si vous utilisez actuellement la version bêta de Claude Code GitHub Actions, nous vous recommandons de mettre à jour vos flux de travail pour utiliser la version GA. La nouvelle version simplifie la configuration tout en ajoutant des fonctionnalités puissantes comme la détection automatique du mode.

Changements essentiels

Tous les utilisateurs bêta doivent apporter ces changements à leurs fichiers de flux de travail pour mettre à niveau :

1. **Mettez à jour la version de l'action** : Changez `@beta` en `@v1`
2. **Supprimez la configuration du mode** : Supprimez `mode: "tag"` ou `mode: "agent"` (maintenant détecté automatiquement)
3. **Mettez à jour les entrées de prompt** : Remplacez `direct_prompt` par `prompt`
4. **Déplacez les options CLI** : Convertissez `max_turns`, `model`, `custom_instructions`, etc. en `claude_args`

Référence des changements majeurs

Ancienne entrée bêta	Nouvelle entrée v1.0
mode	(Supprimée - détectée automatiquement)
direct_prompt	prompt
override_prompt	prompt avec variables GitHub
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 format JSON

Exemple avant et après

Version bêta :

```
- 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"
```

Version 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 détecte maintenant automatiquement s'il faut s'exécuter en mode interactif (répond aux mentions `@claude`) ou en mode automatisé (s'exécute immédiatement avec un prompt) en fonction de votre configuration.

Exemples de cas d'usage

Claude Code GitHub Actions peut vous aider avec une variété de tâches. Le [répertoire d'exemples](#) contient des flux de travail prêts à l'emploi pour différents scénarios.

Flux de travail basique

```

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

```

Utilisation de skills

```
name: Code Review
on:
  pull_request:
    types: [opened, synchronize]
jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Review this pull request for code quality, correctness, and
security. Analyze the diff, then post your findings as review comments."
          claude_args: "--max-turns 5"
```

Automatisation personnalisée avec prompts

```
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${ secrets.ANTHROPIC_API_KEY }
          prompt: "Generate a summary of yesterday's commits and open issues"
          claude_args: "--model opus"
```

Cas d'usage courants

Dans les commentaires d'issue ou de 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 analysera automatiquement le contexte et répondra de manière appropriée.

Bonnes pratiques

Configuration CLAUDE.md

Créez un fichier `CLAUDE.md` à la racine de votre repository pour définir les directives de style de code, les critères de révision, les règles spécifiques au projet et les modèles préférés. Ce fichier guide la compréhension de Claude des normes de votre projet.

Considérations de sécurité

Warning:

Ne commitez jamais les clés API directement dans votre repository.

Pour des conseils de sécurité complets incluant les permissions, l'authentification et les bonnes pratiques, consultez la [documentation de sécurité de Claude Code Action](#).

Utilisez toujours GitHub Secrets pour les clés API :

- Ajoutez votre clé API en tant que secret de repository nommé `ANTHROPIC_API_KEY`
- Référez-la dans les flux de travail : `anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`
- Limitez les permissions de l'action à ce qui est nécessaire uniquement
- Examinez les suggestions de Claude avant de fusionner

Utilisez toujours GitHub Secrets (par exemple, `${{ secrets.ANTHROPIC_API_KEY }}`) plutôt que de coder en dur les clés API directement dans vos fichiers de flux de travail.

Optimisation des performances

Utilisez les modèles d'issue pour fournir du contexte, gardez votre `CLAUDE.md` concis et ciblé, et configurez les délais d'attente appropriés pour vos flux de travail.

Coûts CI

Lorsque vous utilisez Claude Code GitHub Actions, soyez conscient des coûts associés :

Coûts GitHub Actions :

- Claude Code s'exécute sur les runners hébergés par GitHub, qui consomment vos minutes GitHub Actions
- Consultez la [documentation de facturation de GitHub](#) pour les tarifs détaillés et les limites de minutes

Coûts API :

- Chaque interaction Claude consomme des tokens API en fonction de la longueur des prompts et des réponses
- L'utilisation des tokens varie selon la complexité de la tâche et la taille de la base de code
- Consultez la [page de tarification de Claude](#) pour les tarifs actuels des tokens

Conseils d'optimisation des coûts :

- Utilisez des commandes `@claude` spécifiques pour réduire les appels API inutiles
- Configurez `--max-turns` approprié dans `claude_args` pour éviter les itérations excessives
- Définissez les délais d'attente au niveau du flux de travail pour éviter les jobs qui s'exécutent indéfiniment
- Envisagez d'utiliser les contrôles de concurrence de GitHub pour limiter les exécutions parallèles

Exemples de configuration

Claude Code Action v1 simplifie la configuration avec des paramètres unifiés :

```
- 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
```

Fonctionnalités clés :

- **Interface de prompt unifiée** - Utilisez `prompt` pour toutes les instructions
- **Skills** - Invoquez les [skills](#) installés directement depuis le prompt
- **Passthrough CLI** - N'importe quel argument Claude Code CLI via `claude_args`
- **Déclencheurs flexibles** - Fonctionne avec n'importe quel événement GitHub

Visitez le [répertoire d'exemples](#) pour les fichiers de flux de travail complets.

Tip:

Lorsque vous répondez à des commentaires d'issue ou de PR, Claude répond automatiquement aux mentions @claude. Pour les autres événements, utilisez le paramètre `prompt` pour fournir des instructions.

Utilisation avec AWS Bedrock et Google Vertex AI

Pour les environnements d'entreprise, vous pouvez utiliser Claude Code GitHub Actions avec votre propre infrastructure cloud. Cette approche vous donne le contrôle sur la résidence des données et la facturation tout en maintenant les mêmes fonctionnalités.

Prérequis

Avant de configurer Claude Code GitHub Actions avec les fournisseurs cloud, vous avez besoin de :

Pour Google Cloud Vertex AI :

1. Un projet Google Cloud avec Vertex AI activé
2. Workload Identity Federation configuré pour GitHub Actions
3. Un compte de service avec les permissions requises
4. Une application GitHub (recommandée) ou utilisez le `GITHUB_TOKEN` par défaut

Pour AWS Bedrock :

1. Un compte AWS avec Amazon Bedrock activé
2. GitHub OIDC Identity Provider configuré dans AWS
3. Un rôle IAM avec les permissions Bedrock
4. Une application GitHub (recommandée) ou utilisez le `GITHUB_TOKEN` par défaut

Step 1: Créer une application GitHub personnalisée (Recommandée pour les fournisseurs tiers)

Pour un meilleur contrôle et une meilleure sécurité lors de l'utilisation de fournisseurs tiers comme Vertex AI ou Bedrock, nous recommandons de créer votre propre application GitHub :

1. Allez à <https://github.com/settings/apps/new>

2. Remplissez les informations de base :

- **Nom de l'application GitHub** : Choisissez un nom unique (par exemple, 'YourOrg Claude Assistant')
- **URL de la page d'accueil** : Le site web de votre organisation ou l'URL du repository

1. Configurez les paramètres de l'application :

- **Webhooks** : Décochez 'Active' (non nécessaire pour cette intégration)

1. Définissez les permissions requises :

- **Permissions du repository** :
- Contents : Lecture et écriture
- Issues : Lecture et écriture
- Pull requests : Lecture et écriture

1. Cliquez sur 'Create GitHub App'

2. Après la création, cliquez sur 'Generate a private key' et enregistrez le fichier `.pem` téléchargé

3. Notez votre ID d'application à partir de la page des paramètres de l'application

4. Installez l'application dans votre repository :

- À partir de la page des paramètres de votre application, cliquez sur 'Install App' dans la barre latérale gauche
- Sélectionnez votre compte ou organisation
- Choisissez 'Only select repositories' et sélectionnez le repository spécifique
- Cliquez sur 'Install'

1. Ajoutez la clé privée en tant que secret à votre repository :

- Allez à Settings → Secrets and variables → Actions de votre repository
- Créez un nouveau secret nommé `APP_PRIVATE_KEY` avec le contenu du fichier `.pem`

1. Ajoutez l'ID de l'application en tant que secret :

- Créez un nouveau secret nommé `APP_ID` avec l'ID de votre application GitHub

Note:

Cette application sera utilisée avec l'action [actions/create-github-app-token](#) pour générer des tokens d'authentification dans vos flux de travail.

Alternative pour l'API Claude ou si vous ne voulez pas configurer votre propre application Github : Utilisez l'application officielle Anthropic :

1. Installez depuis : <https://github.com/apps/claude>
2. Aucune configuration supplémentaire nécessaire pour l'authentification

Step 2: Configurer l'authentification du fournisseur cloud

Choisissez votre fournisseur cloud et configurez l'authentification sécurisée :

Configurez AWS pour permettre à GitHub Actions de s'authentifier de manière sécurisée sans stocker les credentials.

***Note de sécurité :** Utilisez des configurations spécifiques au repository et accordez uniquement les permissions minimales requises.*

Configuration requise :

1. **Activez Amazon Bedrock :**
 - Demandez l'accès aux modèles Claude dans Amazon Bedrock
 - Pour les modèles multi-régions, demandez l'accès dans toutes les régions requises

1. **Configurez le fournisseur d'identité GitHub OIDC :**

- URL du fournisseur : `https://token.actions.githubusercontent.com`
- Audience : `sts.amazonaws.com`

1. **Créez un rôle IAM pour GitHub Actions :**

- Type d'entité de confiance : Web identity
- Fournisseur d'identité : `token.actions.githubusercontent.com`
- Permissions : politique `AmazonBedrockFullAccess`
- Configurez la politique de confiance pour votre repository spécifique

Valeurs requises :

Après la configuration, vous aurez besoin de :

- **AWS_ROLE_TO_ASSUME :** L'ARN du rôle IAM que vous avez créé

Tip:

OIDC est plus sécurisé que l'utilisation de clés d'accès AWS statiques car les credentials sont temporaires et automatiquement renouvelés.

Consultez la [documentation AWS](#) pour les instructions détaillées de configuration OIDC.

Configurez Google Cloud pour permettre à GitHub Actions de s'authentifier de manière sécurisée sans stocker les credentials.

Note de sécurité : Utilisez des configurations spécifiques au repository et accordez uniquement les permissions minimales requises.

Configuration requise :

1. Activez les APIs dans votre projet Google Cloud :

- IAM Credentials API
- Security Token Service (STS) API
- Vertex AI API

1. Créez les ressources Workload Identity Federation :

- Créez un Workload Identity Pool
- Ajoutez un fournisseur GitHub OIDC avec :
 - Émetteur : <https://token.actions.githubusercontent.com>
 - Mappages d'attributs pour le repository et le propriétaire
- **Recommandation de sécurité :** Utilisez des conditions d'attribut spécifiques au repository

1. Créez un compte de service :

- Accordez uniquement le rôle `Vertex AI User`
- **Recommandation de sécurité :** Créez un compte de service dédié par repository

1. Configurez les liaisons IAM :

- Autorisez le Workload Identity Pool à emprunter l'identité du compte de service
- **Recommandation de sécurité :** Utilisez des ensembles de principaux spécifiques au repository

Valeurs requises :

Après la configuration, vous aurez besoin de :

- **GCP_WORKLOAD_IDENTITY_PROVIDER :** Le nom complet de la ressource du fournisseur
- **GCP_SERVICE_ACCOUNT :** L'adresse e-mail du compte de service

Tip:

Workload Identity Federation élimine le besoin de clés de compte de service téléchargeables, améliorant la sécurité.

Pour les instructions de configuration détaillées, consultez la [documentation Google Cloud Workload Identity Federation](#).

Step 3: Ajouter les secrets requis

Ajoutez les secrets suivants à votre repository (Settings → Secrets and variables → Actions) :

Pour l'API Claude (Direct) :**1. Pour l'authentification API :**

- `ANTHROPIC_API_KEY` : Votre clé API Claude depuis console.anthropic.com

1. Pour l'application GitHub (si vous utilisez votre propre application) :

- `APP_ID` : L'ID de votre application GitHub
- `APP_PRIVATE_KEY` : Le contenu de la clé privée (.pem)

Pour Google Cloud Vertex AI**1. Pour l'authentification GCP :**

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

1. Pour l'application GitHub (si vous utilisez votre propre application) :

- `APP_ID` : L'ID de votre application GitHub
- `APP_PRIVATE_KEY` : Le contenu de la clé privée (.pem)

Pour AWS Bedrock**1. Pour l'authentification AWS :**

- `AWS_ROLE_TO_ASSUME`

1. Pour l'application GitHub (si vous utilisez votre propre application) :

- `APP_ID` : L'ID de votre application GitHub
- `APP_PRIVATE_KEY` : Le contenu de la clé privée (.pem)

Step 4: Créer des fichiers de flux de travail

Créez des fichiers de flux de travail GitHub Actions qui s'intègrent à votre fournisseur cloud. Les exemples ci-dessous montrent des configurations complètes pour AWS Bedrock et Google Vertex AI :

Prérequis :

- Accès AWS Bedrock activé avec permissions de modèle Claude
- GitHub configuré en tant que fournisseur d'identité OIDC dans AWS
- Rôle IAM avec permissions Bedrock qui fait confiance à GitHub Actions

Secrets GitHub requis :

Nom du secret	Description
<code>AWS_ROLE_TO_ASSUME</code>	ARN du rôle IAM pour l'accès à Bedrock
<code>APP_ID</code>	Votre ID d'application GitHub (à partir des paramètres de l'application)
<code>APP_PRIVATE_KEY</code>	La clé privée que vous avez générée pour votre application GitHub

```

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:

Le format d'ID de modèle pour Bedrock inclut un préfixe de région (par exemple, `us.anthropic.claude-sonnet-4-6`).

Prérequis :

- Vertex AI API activée dans votre projet GCP
- Workload Identity Federation configurée pour GitHub
- Compte de service avec permissions Vertex AI

Secrets GitHub requis :

Nom du secret	Description
GCP_WORKLOAD_IDENTITY_PROVIDER	Nom de ressource du fournisseur d'identité de charge de travail
GCP_SERVICE_ACCOUNT	E-mail du compte de service avec accès à Vertex AI
APP_ID	Votre ID d'application GitHub (à partir des paramètres de l'application)
APP_PRIVATE_KEY	La clé privée que vous avez générée pour votre application GitHub

```

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 du projet est automatiquement récupéré à partir de l'étape d'authentification Google Cloud, vous n'avez donc pas besoin de le coder en dur.

Dépannage

Claude ne répond pas aux commandes @claude

Vérifiez que l'application GitHub est correctement installée, vérifiez que les flux de travail sont activés, assurez-vous que la clé API est définie dans les secrets du repository et confirmez que le commentaire contient `@claude` (pas `/claude`).

CI ne s'exécute pas sur les commits de Claude

Assurez-vous que vous utilisez l'application GitHub ou une application personnalisée (pas l'utilisateur Actions), vérifiez que les déclencheurs de flux de travail incluent les événements nécessaires et vérifiez que les permissions de l'application incluent les déclencheurs CI.

Erreurs d'authentification

Confirmez que la clé API est valide et dispose des permissions suffisantes. Pour Bedrock/Vertex, vérifiez la configuration des credentials et assurez-vous que les secrets sont nommés correctement dans les flux de travail.

Configuration avancée

Paramètres de l'action

Claude Code Action v1 utilise une configuration simplifiée :

Paramètre	Description	Requis
<code>prompt</code>	Instructions pour Claude (texte brut ou un nom de skill)	Non*
<code>claude_args</code>	Arguments CLI passés à Claude Code	Non
<code>anthropic_api_key</code>	Clé API Claude	Oui**
<code>github_token</code>	Token GitHub pour l'accès API	Non
<code>trigger_phrase</code>	Phrase de déclenchement personnalisée (par défaut : « @claude »)	Non
<code>use_bedrock</code>	Utiliser AWS Bedrock au lieu de l'API Claude	Non
<code>use_vertex</code>	Utiliser Google Vertex AI au lieu de l'API Claude	Non

*Le prompt est optionnel - lorsqu'il est omis pour les commentaires d'issue/PR, Claude répond à la phrase de déclenchement

**Requis pour l'API Claude directe, pas pour Bedrock/Vertex

Passer les arguments CLI

Le paramètre `claude_args` accepte n'importe quel argument Claude Code CLI :

```
claude_args: "--max-turns 5 --model claude-sonnet-4-6 --mcp-config /path/to/config.json"
```

Arguments courants :

- `--max-turns` : Nombre maximum de tours de conversation (par défaut : 10)
- `--model` : Modèle à utiliser (par exemple, `claude-sonnet-4-6`)
- `--mcp-config` : Chemin vers la configuration MCP
- `--allowed-tools` : Liste séparée par des virgules des outils autorisés
- `--debug` : Activer la sortie de débogage

Méthodes d'intégration alternatives

Bien que la commande `/install-github-app` soit l'approche recommandée, vous pouvez également :

- **Application GitHub personnalisée** : Pour les organisations ayant besoin de noms d'utilisateur de marque ou de flux d'authentification personnalisés. Créez votre propre application GitHub avec les permissions requises (contents, issues, pull requests) et utilisez l'action `actions/create-github-app-token` pour générer des tokens dans vos flux de travail.
- **GitHub Actions manuel** : Configuration directe du flux de travail pour une flexibilité maximale
- **Configuration MCP** : Chargement dynamique des serveurs Model Context Protocol

Consultez la [documentation Claude Code Action](#) pour des guides détaillés sur l'authentification, la sécurité et la configuration avancée.

Personnalisation du comportement de Claude

Vous pouvez configurer le comportement de Claude de deux façons :

1. **CLAUDE.md** : Définissez les normes de codage, les critères de révision et les règles spécifiques au projet dans un fichier `CLAUDE.md` à la racine de votre repository. Claude suivra ces directives lors de la création de PRs et de la réponse aux demandes. Consultez notre [documentation Memory](#) pour plus de détails.
2. **Prompts personnalisés** : Utilisez le paramètre `prompt` dans le fichier de flux de travail pour fournir des instructions spécifiques au flux de travail. Cela vous permet de personnaliser le comportement de Claude pour différents flux de travail ou tâches.

Claude suivra ces directives lors de la création de PRs et de la réponse aux demandes.

Claude Code GitLab CI/CD

Découvrez comment intégrer Claude Code dans votre flux de travail de développement avec GitLab CI/CD

Info:

Claude Code pour GitLab CI/CD est actuellement en bêta. Les fonctionnalités et les capacités peuvent évoluer au fur et à mesure que nous affinons l'expérience.

Cette intégration est maintenue par GitLab. Pour obtenir de l'aide, consultez le [problème GitLab](#) suivant.

Note:

Cette intégration est construite sur la base de [Claude Code CLI](#) et [Agent SDK](#), permettant l'utilisation programmatique de Claude dans vos tâches CI/CD et vos flux de travail d'automatisation personnalisés.

Pourquoi utiliser Claude Code avec GitLab ?

- **Création instantanée de MR** : Décrivez ce dont vous avez besoin, et Claude propose une MR complète avec les modifications et une explication
- **Implémentation automatisée** : Transformez les problèmes en code fonctionnel avec une seule commande ou mention
- **Conscient du projet** : Claude suit vos directives `CLAUDE.md` et les modèles de code existants
- **Configuration simple** : Ajoutez une tâche à `.gitlab-ci.yml` et une variable CI/CD masquée
- **Prêt pour l'entreprise** : Choisissez Claude API, AWS Bedrock ou Google Vertex AI pour répondre aux besoins de résidence des données et d'approvisionnement
- **Sécurisé par défaut** : S'exécute dans vos exécuteurs GitLab avec votre protection de branche et vos approbations

Comment ça marche

Claude Code utilise GitLab CI/CD pour exécuter des tâches d'IA dans des tâches isolées et valider les résultats via des MR :

1. **Orchestration basée sur les événements** : GitLab écoute les déclencheurs que vous choisissez (par exemple, un commentaire qui mentionne `@claude` dans un problème, une MR ou un fil de discussion). La tâche collecte le contexte du fil et du référentiel, construit des invites à partir de cette entrée et exécute Claude Code.
2. **Abstraction du fournisseur** : Utilisez le fournisseur qui correspond à votre environnement :
 - Claude API (SaaS)
 - AWS Bedrock (accès basé sur IAM, options multi-régions)
 - Google Vertex AI (natif GCP, Workload Identity Federation)
3. **Exécution en bac à sable** : Chaque interaction s'exécute dans un conteneur avec des règles strictes de réseau et de système de fichiers. Claude Code applique des autorisations limitées à l'espace de travail pour limiter les écritures. Chaque modification passe par une MR afin que les examinateurs voient la différence et que les approbations s'appliquent toujours.

Choisissez des points de terminaison régionaux pour réduire la latence et respecter les exigences de souveraineté des données tout en utilisant les accords cloud existants.

Que peut faire Claude ?

Claude Code active des flux de travail CI/CD puissants qui transforment votre façon de travailler avec le code :

- Créer et mettre à jour des MR à partir de descriptions ou de commentaires de problèmes
- Analyser les régressions de performance et proposer des optimisations
- Implémenter des fonctionnalités directement dans une branche, puis ouvrir une MR
- Corriger les bogues et les régressions identifiés par les tests ou les commentaires
- Répondre aux commentaires de suivi pour itérer sur les modifications demandées

Configuration

Configuration rapide

Le moyen le plus rapide de commencer est d'ajouter une tâche minimale à votre `.gitlab-ci.yml` et de définir votre clé API comme variable masquée.

1. Ajouter une variable CI/CD masquée

- Allez à **Paramètres** → **CI/CD** → **Variables**
- Ajoutez `ANTHROPIC_API_KEY` (masquée, protégée selon les besoins)

2. Ajouter une tâche Claude à `.gitlab-ci.yml`


```

stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # Ajustez les règles pour adapter la façon dont vous souhaitez déclencher la
  tâche :
  # - exécutions manuelles
  # - événements de demande de fusion
  # - déclencheurs web/API lorsqu'un commentaire contient « @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:
    # Optionnel : démarrer un serveur GitLab MCP si votre configuration en fournit
    un
    - /bin/gitlab-mcp-server || true
    # Utilisez les variables AI_FLOW_* lors de l'invocation via des déclencheurs
    web/API avec des charges utiles de contexte
    - 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

```

Après avoir ajouté la tâche et votre variable `ANTHROPIC_API_KEY`, testez en exécutant la tâche manuellement à partir de **CI/CD** → **Pipelines**, ou déclenchez-la à partir d'une MR pour laisser Claude proposer des mises à jour dans une branche et ouvrir une MR si nécessaire.

Note:

Pour exécuter sur AWS Bedrock ou Google Vertex AI au lieu de Claude API, consultez la section [Utilisation avec AWS Bedrock et Google Vertex AI](#) ci-dessous pour la configuration de l'authentification et de l'environnement.

Configuration manuelle (recommandée pour la production)

Si vous préférez une configuration plus contrôlée ou si vous avez besoin de fournisseurs d'entreprise :

1. Configurer l'accès au fournisseur :

- **Claude API** : Créez et stockez `ANTHROPIC_API_KEY` comme variable CI/CD masquée
- **AWS Bedrock** : Configurer GitLab → AWS OIDC et créer un rôle IAM pour Bedrock
- **Google Vertex AI** : Configurer Workload Identity Federation pour GitLab → GCP

2. Ajouter les identifiants du projet pour les opérations de l'API GitLab :

- Utilisez `CI_JOB_TOKEN` par défaut, ou créez un jeton d'accès au projet avec la portée `api`
- Stockez comme `GITLAB_ACCESS_TOKEN` (masqué) si vous utilisez un PAT

3. Ajouter la tâche Claude à `.gitlab-ci.yml` (voir les exemples ci-dessous)

4. (Optionnel) Activer les déclencheurs basés sur les mentions :

- Ajoutez un webhook de projet pour « Commentaires (notes) » à votre écouteur d'événements (si vous en utilisez un)
- Faites en sorte que l'écouteur appelle l'API de déclenchement du pipeline avec des variables comme `AI_FLOW_INPUT` et `AI_FLOW_CONTEXT` lorsqu'un commentaire contient `@claude`

Exemples de cas d'utilisation

Transformer les problèmes en MR

Dans un commentaire de problème :

```
@claude implement this feature based on the issue description
```

Claude analyse le problème et la base de code, écrit les modifications dans une branche et ouvre une MR pour examen.

Obtenir de l'aide à l'implémentation

Dans une discussion MR :

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude propose des modifications, ajoute du code avec la mise en cache appropriée et met à jour la MR.

Corriger les bogues rapidement

Dans un commentaire de problème ou de MR :

```
@claude fix the TypeError in the user dashboard component
```

Claude localise le bogue, implémente un correctif et met à jour la branche ou ouvre une nouvelle MR.

Utilisation avec AWS Bedrock et Google Vertex AI

Pour les environnements d'entreprise, vous pouvez exécuter Claude Code entièrement sur votre infrastructure cloud avec la même expérience développeur.

AWS Bedrock

Conditions préalables

Avant de configurer Claude Code avec AWS Bedrock, vous avez besoin de :

1. Un compte AWS avec accès à Amazon Bedrock pour les modèles Claude souhaités
2. GitLab configuré comme fournisseur d'identité OIDC dans AWS IAM
3. Un rôle IAM avec les autorisations Bedrock et une politique de confiance limitée à votre projet/références GitLab
4. Variables CI/CD GitLab pour l'assumption de rôle :
 - `AWS_ROLE_TO_ASSUME` (ARN du rôle)
 - `AWS_REGION` (région Bedrock)

Instructions de configuration

Configurez AWS pour permettre aux tâches CI GitLab d'assumer un rôle IAM via OIDC (pas de clés statiques).

Configuration requise :

1. Activez Amazon Bedrock et demandez l'accès à vos modèles Claude cibles
2. Créez un fournisseur OIDC IAM pour GitLab s'il n'existe pas déjà
3. Créez un rôle IAM approuvé par le fournisseur OIDC GitLab, limité à votre projet et aux références protégées
4. Attachez les autorisations de moindre privilège pour les API d'invocation Bedrock

Valeurs requises à stocker dans les variables CI/CD :

- `AWS_ROLE_TO_ASSUME`
- `AWS_REGION`

Ajoutez les variables dans Paramètres → CI/CD → Variables :

```
## Pour AWS Bedrock :
- AWS_ROLE_TO_ASSUME
- AWS_REGION
```

Utilisez l'exemple de tâche AWS Bedrock ci-dessus pour échanger le jeton de tâche GitLab contre des identifiants AWS temporaires au moment de l'exécution.

Google Vertex AI

Conditions préalables

Avant de configurer Claude Code avec Google Vertex AI, vous avez besoin de :

1. Un projet Google Cloud avec :
 - API Vertex AI activée
 - Workload Identity Federation configurée pour faire confiance à GitLab OIDC
1. Un compte de service dédié avec uniquement les rôles Vertex AI requis
2. Variables CI/CD GitLab pour WIF :
 - `GCP_WORKLOAD_IDENTITY_PROVIDER` (nom complet de la ressource)
 - `GCP_SERVICE_ACCOUNT` (e-mail du compte de service)

Instructions de configuration

Configurez Google Cloud pour permettre aux tâches CI GitLab d'emprunter l'identité d'un compte de service via Workload Identity Federation.

Configuration requise :

1. Activez l'API IAM Credentials, l'API STS et l'API Vertex AI
2. Créez un pool Workload Identity et un fournisseur pour GitLab OIDC
3. Créez un compte de service dédié avec les rôles Vertex AI
4. Accordez au principal WIF la permission d'emprunter l'identité du compte de service

Valeurs requises à stocker dans les variables CI/CD :

- `GCP_WORKLOAD_IDENTITY_PROVIDER`
- `GCP_SERVICE_ACCOUNT`

Ajoutez les variables dans Paramètres → CI/CD → Variables :

```
## Pour Google Vertex AI :  
- GCP_WORKLOAD_IDENTITY_PROVIDER  
- GCP_SERVICE_ACCOUNT  
- CLOUD_ML_REGION (par exemple, us-east5)
```

Utilisez l'exemple de tâche Google Vertex AI ci-dessus pour vous authentifier sans stocker de clés.

Exemples de configuration

Voici des extraits prêts à l'emploi que vous pouvez adapter à votre pipeline.

.gitlab-ci.yml basique (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 utilisera ANTHROPIC_API_KEY à partir des variables CI/CD

```

Exemple de tâche AWS Bedrock (OIDC)

Conditions préalables :

- Amazon Bedrock activé avec accès à votre ou vos modèles Claude choisis
- GitLab OIDC configuré dans AWS avec un rôle qui fait confiance à votre projet et vos références GitLab
- Rôle IAM avec autorisations Bedrock (moindre privilège recommandé)

Variables CI/CD requises :

- `AWS_ROLE_TO_ASSUME` : ARN du rôle IAM pour l'accès à Bedrock
- `AWS_REGION` : Région Bedrock (par exemple, `us-west-2`)

```

claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apk add --no-cache bash curl jq git python3 py3-pip
    - pip install --no-cache-dir awscli
    - curl -fsSL https://claude.ai/install.sh | bash
    # Échanger le jeton OIDC GitLab contre les identifiants AWS
    - export AWS_WEB_IDENTITY_TOKEN_FILE="${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:

Les identifiants de modèle pour Bedrock incluent des préfixes spécifiques à la région (par exemple, `us.anthropic.claude-sonnet-4-6`). Transmettez le modèle souhaité via votre configuration de tâche ou votre invite si votre flux de travail le supporte.

Exemple de tâche Google Vertex AI (Workload Identity Federation)

Conditions préalables :

- API Vertex AI activée dans votre projet GCP
- Workload Identity Federation configurée pour faire confiance à GitLab OIDC
- Un compte de service avec les autorisations Vertex AI

Variables CI/CD requises :

- `GCP_WORKLOAD_IDENTITY_PROVIDER` : Nom complet de la ressource du fournisseur
- `GCP_SERVICE_ACCOUNT` : E-mail du compte de service
- `CLOUD_ML_REGION` : Région Vertex (par exemple, `us-east5`)


```

claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # S'authentifier auprès de Google Cloud via WIF (pas de clés téléchargées)
    - >
      gcloud auth login --cred-file=<(cat <<EOF
      {
        "type": "external_account",
        "audience": "${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:

Avec Workload Identity Federation, vous n'avez pas besoin de stocker les clés du compte de service. Utilisez des conditions de confiance spécifiques au référentiel et des comptes de service avec le moindre privilège.

Bonnes pratiques

Configuration CLAUDE.md

Créez un fichier `CLAUDE.md` à la racine du référentiel pour définir les normes de codage, les critères d'examen et les règles spécifiques au projet. Claude lit ce fichier lors des exécutions et suit vos conventions lors de la proposition de modifications.

Considérations de sécurité

Ne validez jamais les clés API ou les identifiants cloud dans votre référentiel.

Utilisez toujours les variables CI/CD GitLab :

- Ajoutez `ANTHROPIC_API_KEY` comme variable masquée (et protégez-la si nécessaire)
- Utilisez OIDC spécifique au fournisseur si possible (pas de clés longue durée)
- Limitez les autorisations des tâches et la sortie réseau
- Examinez les MR de Claude comme tout autre contributeur

Optimisation des performances

- Gardez `CLAUDE.md` concentré et concis
- Fournissez des descriptions claires de problèmes/MR pour réduire les itérations
- Configurez des délais d'expiration de tâche raisonnables pour éviter les exécutions incontrôlées
- Mettez en cache les installations npm et de paquets dans les exécuteurs si possible

Coûts CI

Lorsque vous utilisez Claude Code avec GitLab CI/CD, soyez conscient des coûts associés :

- **Temps d'exécution GitLab Runner :**
 - Claude s'exécute sur vos exécuteurs GitLab et consomme des minutes de calcul
 - Consultez la facturation des exécuteurs de votre plan GitLab pour plus de détails

- **Coûts API :**

- Chaque interaction Claude consomme des jetons en fonction de la taille de l'invite et de la réponse
- L'utilisation des jetons varie selon la complexité de la tâche et la taille de la base de code
- Consultez [Tarification Anthropic](#) pour plus de détails

- **Conseils d'optimisation des coûts :**

- Utilisez des commandes `@claude` spécifiques pour réduire les tours inutiles
- Définissez les valeurs `max_turns` et le délai d'expiration de la tâche appropriés
- Limitez la concurrence pour contrôler les exécutions parallèles

Sécurité et gouvernance

- Chaque tâche s'exécute dans un conteneur isolé avec accès réseau restreint
- Les modifications de Claude passent par des MR afin que les examinateurs voient chaque différence
- Les règles de protection de branche et d'approbation s'appliquent au code généré par l'IA
- Claude Code utilise des autorisations limitées à l'espace de travail pour limiter les écritures
- Les coûts restent sous votre contrôle car vous apportez vos propres identifiants de fournisseur

Dépannage

Claude ne répond pas aux commandes `@claude`

- Vérifiez que votre pipeline est déclenché (manuellement, événement MR ou via un écouteur d'événements/webhook de note)
- Assurez-vous que les variables CI/CD (`ANTHROPIC_API_KEY` ou les paramètres du fournisseur cloud) sont présentes et non masquées
- Vérifiez que le commentaire contient `@claude` (pas `/claude`) et que votre déclencheur de mention est configuré

La tâche ne peut pas écrire de commentaires ou ouvrir des MR

- Assurez-vous que `CI_JOB_TOKEN` dispose des autorisations suffisantes pour le projet, ou utilisez un jeton d'accès au projet avec la portée `api`

- Vérifiez que l'outil `mcp__gitlab` est activé dans `--allowedTools`
- Confirmez que la tâche s'exécute dans le contexte de la MR ou dispose de suffisamment de contexte via les variables `AI_FLOW_*`

Erreurs d'authentification

- **Pour Claude API** : Confirmez que `ANTHROPIC_API_KEY` est valide et non expiré
- **Pour Bedrock/Vertex** : Vérifiez la configuration OIDC/WIF, l'emprunt d'identité de rôle et les noms secrets ; confirmez la disponibilité de la région et du modèle

Configuration avancée

Paramètres et variables courants

Claude Code supporte ces entrées couramment utilisées :

- `prompt` / `prompt_file` : Fournissez les instructions en ligne (`-p`) ou via un fichier
- `max_turns` : Limitez le nombre d'itérations aller-retour
- `timeout_minutes` : Limitez le temps d'exécution total
- `ANTHROPIC_API_KEY` : Requis pour Claude API (non utilisé pour Bedrock/Vertex)
- Environnement spécifique au fournisseur : `AWS_REGION`, variables de projet/région pour Vertex

Note:

Les drapeaux et paramètres exacts peuvent varier selon la version de `@anthropic-ai/claude-code`. Exécutez `claude --help` dans votre tâche pour voir les options prises en charge.

Personnalisation du comportement de Claude

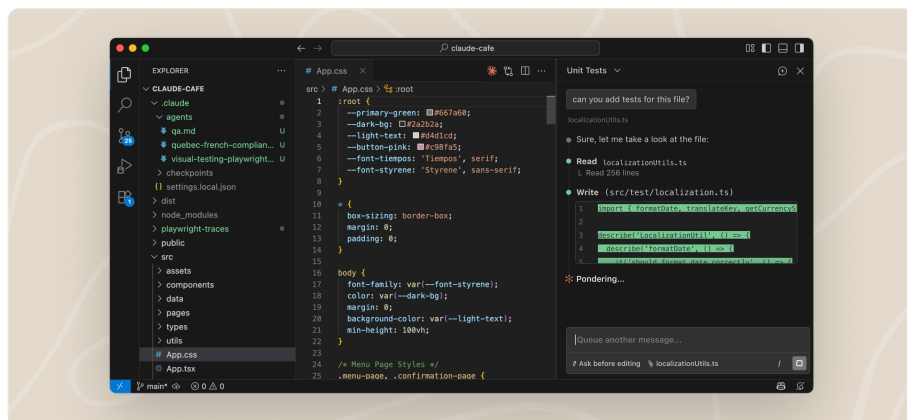
Vous pouvez guider Claude de deux façons principales :

1. **CLAUDE.md** : Définissez les normes de codage, les exigences de sécurité et les conventions du projet. Claude lit ceci lors des exécutions et suit vos règles.
2. **Invites personnalisées** : Transmettez les instructions spécifiques à la tâche via `prompt` / `prompt_file` dans la tâche. Utilisez différentes invites pour différentes tâches (par exemple, examen, implémentation, refactorisation).

Part 8: IDE & Platform Integration

Utiliser Claude Code dans VS Code

Installez et configurez l'extension Claude Code pour VS Code. Obtenez une assistance de codage IA avec des diffs en ligne, des mentions @, un examen du plan et des raccourcis clavier.



Éditeur VS Code avec le panneau d'extension Claude Code ouvert sur le côté droit, montrant une conversation avec Claude

L'extension VS Code fournit une interface graphique native pour Claude Code, intégrée directement dans votre IDE. C'est la façon recommandée d'utiliser Claude Code dans VS Code.

Avec l'extension, vous pouvez examiner et modifier les plans de Claude avant de les accepter, accepter automatiquement les modifications au fur et à mesure qu'elles sont apportées, mentionner des fichiers avec des plages de lignes spécifiques à partir de votre sélection, accéder à l'historique des conversations et ouvrir plusieurs conversations dans des onglets ou des fenêtres séparés.

Conditions préalables

Avant d'installer, assurez-vous que vous avez :

- VS Code 1.98.0 ou supérieur

- Un compte Anthropic (vous vous connecterez lors de la première ouverture de l'extension). Si vous utilisez un fournisseur tiers comme Amazon Bedrock ou Google Vertex AI, consultez plutôt [Utiliser des fournisseurs tiers](#).

Tip:

L'extension inclut l'interface de ligne de commande (CLI), à laquelle vous pouvez accéder depuis le terminal intégré de VS Code pour les fonctionnalités avancées. Consultez [Extension VS Code vs. CLI Claude Code](#) pour plus de détails.

Installer l'extension

Cliquez sur le lien de votre IDE pour installer directement :

- [Installer pour VS Code](#)
- [Installer pour Cursor](#)

Ou dans VS Code, appuyez sur **Cmd+Shift+X** (Mac) ou **Ctrl+Shift+X** (Windows/Linux) pour ouvrir la vue Extensions, recherchez « Claude Code » et cliquez sur **Installer**.

Note:

Si l'extension n'apparaît pas après l'installation, redémarrez VS Code ou exécutez « Developer: Reload Window » depuis la Palette de commandes.

Commencer

Une fois installée, vous pouvez commencer à utiliser Claude Code via l'interface VS Code :

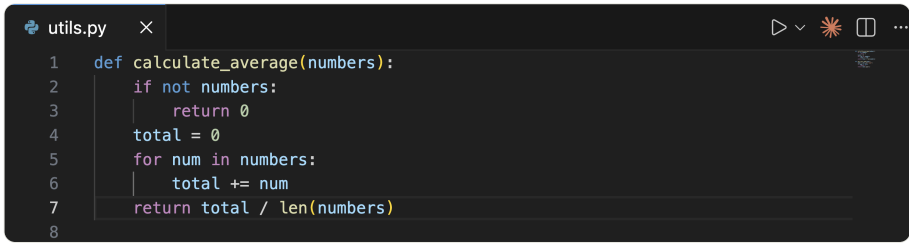
Step 1: Ouvrir le panneau Claude Code

Dans VS Code, l'icône Spark indique Claude Code :



Ikône Spark

Le moyen le plus rapide d'ouvrir Claude est de cliquer sur l'icône Spark dans la **Barre d'outils de l'éditeur** (coin supérieur droit de l'éditeur). L'icône n'apparaît que lorsque vous avez un fichier ouvert.



Éditeur VS Code montrant l'icône Spark dans la Barre d'outils de l'éditeur

Autres façons d'ouvrir Claude Code :

- **Barre d'activité** : cliquez sur l'icône Spark dans la barre latérale gauche pour ouvrir la liste des sessions. Cliquez sur n'importe quelle session pour l'ouvrir en tant qu'onglet d'éditeur complet, ou démarrez-en une nouvelle. Cette icône est toujours visible dans la Barre d'activité.
- **Palette de commandes** : `Cmd+Shift+P` (Mac) ou `Ctrl+Shift+P` (Windows/Linux), tapez « Claude Code » et sélectionnez une option comme « Ouvrir dans un nouvel onglet »
- **Barre d'état** : cliquez sur **Claude Code** dans le coin inférieur droit de la fenêtre. Cela fonctionne même quand aucun fichier n'est ouvert.

Lorsque vous ouvrez le panneau pour la première fois, une liste de contrôle **Apprendre Claude Code** apparaît. Parcourez chaque élément en cliquant sur **Montrer-moi**, ou fermez-la avec le X. Pour la rouvrir plus tard, décochez **Masquer l'intégration** dans les paramètres VS Code sous Extensions → Claude Code.

Vous pouvez faire glisser le panneau Claude pour le repositionner n'importe où dans VS Code. Consultez [Personnaliser votre flux de travail](#) pour plus de détails.

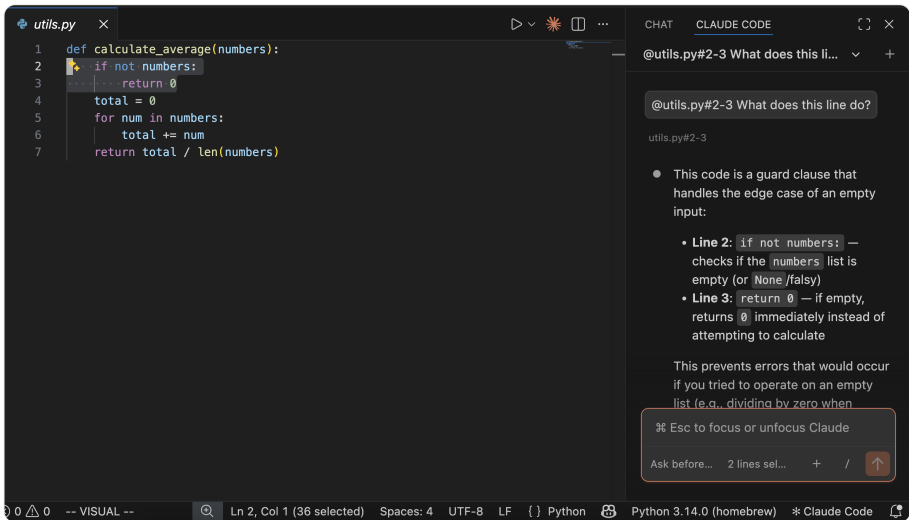
Step 2: Envoyer une invite

Demandez à Claude de vous aider avec votre code ou vos fichiers, qu'il s'agisse d'expliquer comment quelque chose fonctionne, de déboguer un problème ou d'apporter des modifications.

Tip:

Claude voit automatiquement votre texte sélectionné. Appuyez sur `Option+K` (Mac) / `Alt+K` (Windows/Linux) pour insérer également une référence de mention `@` (comme `@file.ts#5-10`) dans votre invite.

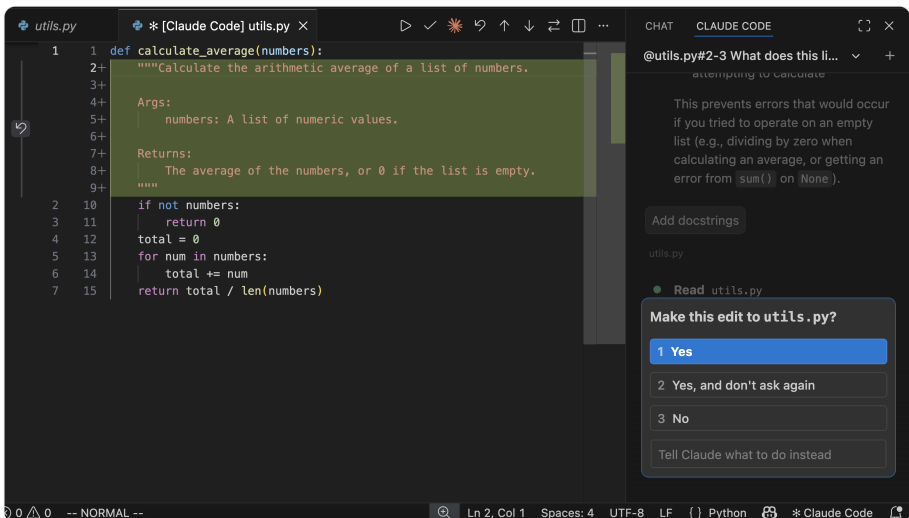
Voici un exemple de question sur une ligne particulière dans un fichier :



Éditeur VS Code avec les lignes 2-3 sélectionnées dans un fichier Python, et le panneau Claude Code montrant une question sur ces lignes avec une référence de mention @

Step 3: Examiner les modifications

Lorsque Claude souhaite modifier un fichier, il affiche une comparaison côte à côte de l'original et des modifications proposées, puis demande une permission. Vous pouvez accepter, rejeter ou dire à Claude ce qu'il faut faire à la place.



VS Code montrant un diff des modifications proposées par Claude avec une invite de permission demandant si vous souhaitez effectuer la modification

Pour plus d'idées sur ce que vous pouvez faire avec Claude Code, consultez [Flux de travail courants](#).

Tip:

Exécutez « Claude Code: Open Walkthrough » depuis la Palette de commandes pour une visite guidée des bases.

Utiliser la zone de saisie

La zone de saisie prend en charge plusieurs fonctionnalités :

- **Modes de permission** : cliquez sur l'indicateur de mode en bas de la zone de saisie pour changer de mode. En mode normal, Claude demande une permission avant chaque action. En Plan Mode, Claude décrit ce qu'il fera et attend l'approbation avant d'apporter des modifications. VS Code ouvre automatiquement le plan en tant que document markdown complet où vous pouvez ajouter des commentaires en ligne pour donner des commentaires avant que Claude ne commence. En mode acceptation automatique, Claude apporte des modifications sans demander. Définissez la valeur par défaut dans les paramètres VS Code sous `claudeCode.initialPermissionMode`.
- **Menu de commandes** : cliquez sur `/` ou tapez `/` pour ouvrir le menu de commandes. Les options incluent l'attachement de fichiers, le changement de modèles, l'activation de la réflexion étendue et l'affichage de l'utilisation du plan (`/usage`). La section Personnaliser fournit l'accès aux serveurs MCP, hooks, mémoire, permissions et plugins. Les éléments avec une icône de terminal s'ouvrent dans le terminal intégré.
- **Indicateur de contexte** : la zone de saisie affiche la quantité de fenêtre de contexte de Claude que vous utilisez. Claude se compacte automatiquement si nécessaire, ou vous pouvez exécuter `/compact` manuellement.
- **Réflexion étendue** : permet à Claude de consacrer plus de temps à raisonner sur des problèmes complexes. Activez-la via le menu de commandes (`/`). Consultez [Réflexion étendue](#) pour plus de détails.
- **Entrée multiligne** : appuyez sur `Shift+Entrée` pour ajouter une nouvelle ligne sans envoyer. Cela fonctionne également dans l'entrée en texte libre « Autre » des dialogues de question.

Référencer des fichiers et des dossiers

Utilisez les mentions @ pour donner à Claude du contexte sur des fichiers ou des dossiers spécifiques. Lorsque vous tapez @ suivi d'un nom de fichier ou de dossier, Claude lit ce contenu et peut répondre à des questions à ce sujet ou y apporter des modifications. Claude Code prend en charge la correspondance floue, vous pouvez donc taper des noms partiels pour trouver ce dont vous avez besoin :

```
> Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
> What's in @src/components/ (include a trailing slash for folders)
```

Pour les grands fichiers PDF, vous pouvez demander à Claude de lire des pages spécifiques au lieu du fichier entier : une seule page, une plage comme les pages 1-10, ou une plage ouverte comme la page 3 et au-delà.

Lorsque vous sélectionnez du texte dans l'éditeur, Claude peut voir votre code en surbrillance automatiquement. Le pied de page de la zone de saisie affiche le nombre de lignes sélectionnées. Appuyez sur **Option+K** (Mac) / **Alt+K** (Windows/Linux) pour insérer une mention @ avec le chemin du fichier et les numéros de ligne (par exemple, @app.ts#5-10). Cliquez sur l'indicateur de sélection pour basculer si Claude peut voir votre texte en surbrillance - l'icône en forme de barre oblique signifie que la sélection est masquée à Claude.

Vous pouvez également maintenir **Shift** enfoncé tout en faisant glisser des fichiers dans la zone de saisie pour les ajouter en tant que pièces jointes. Cliquez sur le X sur n'importe quelle pièce jointe pour la supprimer du contexte.

Reprendre les conversations passées

Cliquez sur la liste déroulante en haut du panneau Claude Code pour accéder à votre historique de conversations. Vous pouvez rechercher par mot-clé ou parcourir par heure (Aujourd'hui, Hier, 7 derniers jours, etc.). Cliquez sur n'importe quelle conversation pour la reprendre avec l'historique complet des messages. Survolez une session pour révéler les actions de renommage et de suppression : renommez pour lui donner un titre descriptif, ou supprimez pour la supprimer de la liste. Pour plus d'informations sur la reprise des sessions, consultez [Flux de travail courants](#).

Reprendre les sessions distantes depuis Claude.ai

Si vous utilisez [Claude Code sur le web](#), vous pouvez reprendre ces sessions distantes directement dans VS Code. Cela nécessite de se connecter avec **Claude.ai Subscription**, pas Anthropic Console.

Step 1: Ouvrir les conversations passées

Cliquez sur la liste déroulante **Conversations passées** en haut du panneau Claude Code.

Step 2: Sélectionner l'onglet Distant

Le dialogue affiche deux onglets : Local et Distant. Cliquez sur **Distant** pour voir les sessions de claude.ai.

Step 3: Sélectionner une session à reprendre

Parcourez ou recherchez vos sessions distantes. Cliquez sur n'importe quelle session pour la télécharger et continuer la conversation localement.

Note:

Seules les sessions web démarrées avec un référentiel GitHub apparaissent dans l'onglet Distant. La reprise charge l'historique de la conversation localement ; les modifications ne sont pas resynchronisées vers claude.ai.

Personnaliser votre flux de travail

Une fois que vous êtes opérationnel, vous pouvez repositionner le panneau Claude, exécuter plusieurs sessions ou passer au mode terminal.

Choisir où Claude se trouve

Vous pouvez faire glisser le panneau Claude pour le repositionner n'importe où dans VS Code. Saisissez l'onglet ou la barre de titre du panneau et faites-le glisser vers :

- **Barre latérale secondaire** : le côté droit de la fenêtre. Garde Claude visible pendant que vous codez.
- **Barre latérale principale** : la barre latérale gauche avec les icônes pour l'Explorateur, la Recherche, etc.
- **Zone d'éditeur** : ouvre Claude en tant qu'onglet à côté de vos fichiers. Utile pour les tâches secondaires.

Tip:

Utilisez la barre latérale pour votre session Claude principale et ouvrez des onglets supplémentaires pour les tâches secondaires. Claude se souvient de votre emplacement préféré. L'icône de la liste des sessions de la Barre d'activité est séparée du panneau Claude : la liste des sessions est toujours visible dans la Barre d'activité, tandis que l'icône du panneau Claude n'y apparaît que lorsque le panneau est ancré à la barre latérale gauche.

Exécuter plusieurs conversations

Utilisez **Ouvrir dans un nouvel onglet** ou **Ouvrir dans une nouvelle fenêtre** depuis la Palette de commandes pour démarrer des conversations supplémentaires. Chaque conversation maintient son propre historique et contexte, vous permettant de travailler sur différentes tâches en parallèle.

Lors de l'utilisation d'onglets, un petit point coloré sur l'icône spark indique l'état : bleu signifie qu'une demande de permission est en attente, orange signifie que Claude a terminé pendant que l'onglet était masqué.

Passer au mode terminal

Par défaut, l'extension ouvre un panneau de chat graphique. Si vous préférez l'interface de style CLI, ouvrez le [paramètre Utiliser le terminal](#) et cochez la case.

Vous pouvez également ouvrir les paramètres VS Code (**Cmd+** , sur Mac ou **Ctrl+** , sur Windows/Linux), aller à Extensions → Claude Code et cocher **Utiliser le terminal**.

Gérer les plugins

L'extension VS Code inclut une interface graphique pour installer et gérer les [plugins](#). Tapez **/plugins** dans la zone de saisie pour ouvrir l'interface **Gérer les plugins**.

Installer les plugins

Le dialogue des plugins affiche deux onglets : **Plugins** et **Marchés**.

Dans l'onglet Plugins :

- Les **plugins installés** apparaissent en haut avec des commutateurs pour les activer ou les désactiver
- Les **plugins disponibles** de vos marchés configurés apparaissent ci-dessous
- Recherchez pour filtrer les plugins par nom ou description
- Cliquez sur **Installer** sur n'importe quel plugin disponible

Lorsque vous installez un plugin, choisissez l'étendue de l'installation :

- **Installer pour vous** : disponible dans tous vos projets (étendue utilisateur)

- **Installer pour ce projet** : partagé avec les collaborateurs du projet (étendue du projet)
- **Installer localement** : uniquement pour vous, uniquement dans ce référentiel (étendue locale)

Gérer les marchés

Basculez vers l'onglet **Marchés** pour ajouter ou supprimer des sources de plugins :

- Entrez un référentiel GitHub, une URL ou un chemin local pour ajouter un nouveau marché
- Cliquez sur l'icône d'actualisation pour mettre à jour la liste des plugins d'un marché
- Cliquez sur l'icône de corbeille pour supprimer un marché

Après avoir apporté des modifications, une bannière vous invite à redémarrer Claude Code pour appliquer les mises à jour.

Note:

La gestion des plugins dans VS Code utilise les mêmes commandes CLI en arrière-plan. Les plugins et les marchés que vous configurez dans l'extension sont également disponibles dans l'interface CLI, et vice versa.

Pour plus d'informations sur le système de plugins, consultez [Plugins](#) et [Marchés de plugins](#).

Automatiser les tâches du navigateur avec Chrome

Connectez Claude à votre navigateur Chrome pour tester des applications web, déboguer avec des journaux de console et automatiser les flux de travail du navigateur sans quitter VS Code. Cela nécessite l'[extension Claude in Chrome](#) version 1.0.36 ou supérieure.

Tapez `@browser` dans la zone de saisie suivi de ce que vous voulez que Claude fasse :

```
@browser go to localhost:3000 and check the console for errors
```

Vous pouvez également ouvrir le menu des pièces jointes pour sélectionner des outils de navigateur spécifiques comme ouvrir un nouvel onglet ou lire le contenu de la page.

Claude ouvre de nouveaux onglets pour les tâches du navigateur et partage l'état de connexion de votre navigateur, il peut donc accéder à n'importe quel site auquel vous êtes déjà connecté.

Pour les instructions de configuration, la liste complète des capacités et le dépannage, consultez [Utiliser Claude Code avec Chrome](#).

Commandes et raccourcis VS Code

Ouvrez la Palette de commandes (`Cmd+Shift+P` sur Mac ou `Ctrl+Shift+P` sur Windows/Linux) et tapez ' Claude Code ' pour voir toutes les commandes VS Code disponibles pour l'extension Claude Code.

Certains raccourcis dépendent du panneau qui est ' actif ' (recevant l'entrée au clavier). Lorsque votre curseur est dans un fichier de code, l'éditeur est actif. Lorsque votre curseur est dans la zone de saisie de Claude, Claude est actif. Utilisez `Cmd+Esc` / `Ctrl+Esc` pour basculer entre eux.

Note:

Ce sont des commandes VS Code pour contrôler l'extension. Toutes les commandes Claude Code intégrées ne sont pas disponibles dans l'extension. Consultez [Extension VS Code vs. CLI Claude Code](#) pour plus de détails.

Commande	Raccourci	Description
Focus Input	<code>Cmd+Esc</code> (Mac) / <code>Ctrl+Esc</code> (Windows/Linux)	Basculer le focus entre l'éditeur et Claude
Open in Side Bar	-	Ouvrir Claude dans la barre latérale gauche
Open in Terminal	-	Ouvrir Claude en mode terminal
Open in New Tab	<code>Cmd+Shift+Esc</code> (Mac) / <code>Ctrl+Shift+Esc</code> (Windows/Linux)	Ouvrir une nouvelle conversation en tant qu'onglet d'éditeur
Open in New Window	-	Ouvrir une nouvelle conversation dans une fenêtre séparée
New Conversation	<code>Cmd+N</code> (Mac) / <code>Ctrl+N</code> (Windows/Linux)	Démarrer une nouvelle conversation (nécessite que Claude soit actif)

Commande	Raccourci	Description
Insert @-Mention Reference	<code>Option+K</code> (Mac) / <code>Alt+K</code> (Windows/Linux)	Insérer une référence au fichier actuel et à la sélection (nécessite que l'éditeur soit actif)
Show Logs	-	Afficher les journaux de débogage de l'extension
Logout	-	Se déconnecter de votre compte Anthropic

Configurer les paramètres

L'extension a deux types de paramètres :

- **Paramètres d'extension** dans VS Code : contrôlent le comportement de l'extension dans VS Code. Ouvrez avec `Cmd+,` (Mac) ou `Ctrl+,` (Windows/Linux), puis allez à Extensions → Claude Code. Vous pouvez également taper `/` et sélectionner **General Config** pour ouvrir les paramètres.
- **Paramètres Claude Code** dans `~/.claude/settings.json` : partagés entre l'extension et l'interface CLI. Utilisez pour les commandes autorisées, les variables d'environnement, les hooks et les serveurs MCP. Consultez [Paramètres](#) pour plus de détails.

Tip:

Ajoutez `"$schema": "https://json.schemastore.org/claude-code-settings.json"` à votre `settings.json` pour obtenir l'autocomplétion et la validation en ligne pour tous les paramètres disponibles directement dans VS Code.

Paramètres d'extension

Paramètre	Par défaut	Description
<code>selectedModel</code>	<code>default</code>	Modèle pour les nouvelles conversations. Changez par session avec <code>/model</code> .
<code>useTerminal</code>	<code>false</code>	Lancer Claude en mode terminal au lieu du panneau graphique

Paramètre	Par défaut	Description
<code>initialPermissionMode</code>	<code>default</code>	Contrôle les invites d'approbation : <code>default</code> (demander à chaque fois), <code>plan</code> , <code>acceptEdits</code> ou <code>bypassPermissions</code>
<code>preferredLocation</code>	<code>panel</code>	Où Claude s'ouvre : <code>sidebar</code> (droite) ou <code>panel</code> (nouvel onglet)
<code>autosave</code>	<code>true</code>	Enregistrement automatique des fichiers avant que Claude ne les lise ou ne les écrive
<code>useCtrlEnterToSend</code>	<code>false</code>	Utiliser Ctrl/Cmd+Entrée au lieu d'Entrée pour envoyer les invites
<code>enableNewConversationShortcut</code>	<code>true</code>	Activer Cmd/Ctrl+N pour démarrer une nouvelle conversation
<code>hideOnboarding</code>	<code>false</code>	Masquer la liste de contrôle d'intégration (icône de chapeau de graduation)
<code>respectGitIgnore</code>	<code>true</code>	Exclure les modèles .gitignore des recherches de fichiers
<code>environmentVariables</code>	<code>[]</code>	Définir les variables d'environnement pour le processus Claude. Utilisez plutôt les paramètres Claude Code pour la configuration partagée.
<code>disableLoginPrompt</code>	<code>false</code>	Ignorer les invites d'authentification (pour les configurations de fournisseur tiers)

Paramètre	Par défaut	Description
<code>allowDangerouslySkipPermissions</code>	<code>false</code>	Contourner toutes les invites de permission. À utiliser avec une extrême prudence.
<code>claudeProcessWrapper</code>	-	Chemin exécutable utilisé pour lancer le processus Claude

Extension VS Code vs. CLI Claude Code

Claude Code est disponible à la fois en tant qu'extension VS Code (panneau graphique) et en tant qu'interface CLI (interface de ligne de commande dans le terminal). Certaines fonctionnalités ne sont disponibles que dans l'interface CLI. Si vous avez besoin d'une fonctionnalité CLI uniquement, exécutez `claude` dans le terminal intégré de VS Code.

Fonctionnalité	CLI	Extension VS Code
Commandes et skills	Tous	Sous-ensemble (tapez <code>/</code> pour voir les options disponibles)
Configuration du serveur MCP	Oui	Partiel (ajouter des serveurs via l'interface CLI ; gérer les serveurs existants avec <code>/mcp</code> dans le panneau de chat)
Checkpoints	Oui	Oui
Raccourci bash <code>!</code>	Oui	Non
Complément de tabulation	Oui	Non

Rembobiner avec les checkpoints

L'extension VS Code prend en charge les checkpoints, qui suivent les modifications de fichiers de Claude et vous permettent de rembobiner à un état précédent. Surveillez n'importe quel message pour révéler le bouton de rembobinage, puis choisissez parmi trois options :

- **Créer une branche de conversation à partir d'ici** : démarrer une nouvelle branche de conversation à partir de ce message tout en conservant toutes les modifications de code
- **Rembobiner le code jusqu'ici** : annuler les modifications de fichiers jusqu'à ce point dans la conversation tout en conservant l'historique complet de la conversation
- **Créer une branche de conversation et rembobiner le code** : démarrer une nouvelle branche de conversation et annuler les modifications de fichiers jusqu'à ce point

Pour tous les détails sur le fonctionnement des checkpoints et leurs limitations, consultez [Checkpointing](#).

Exécuter l'interface CLI dans VS Code

Pour utiliser l'interface CLI tout en restant dans VS Code, ouvrez le terminal intégré (`Ctrl+`` sur Windows/Linux ou `Cmd+`` sur Mac) et exécutez `claude`. L'interface CLI s'intègre automatiquement à votre IDE pour des fonctionnalités comme l'affichage des diffs et le partage des diagnostics.

Si vous utilisez un terminal externe, exécutez `/ide` dans Claude Code pour le connecter à VS Code.

Basculer entre l'extension et l'interface CLI

L'extension et l'interface CLI partagent le même historique de conversations. Pour continuer une conversation d'extension dans l'interface CLI, exécutez `claude --resume` dans le terminal. Cela ouvre un sélecteur interactif où vous pouvez rechercher et sélectionner votre conversation.

Inclure la sortie du terminal dans les invites

Référez la sortie du terminal dans vos invites en utilisant `@terminal:name` où `name` est le titre du terminal. Cela permet à Claude de voir la sortie de la commande, les messages d'erreur ou les journaux sans copier-coller.

Surveiller les processus en arrière-plan

Lorsque Claude exécute des commandes longues, l'extension affiche la progression dans la barre d'état. Cependant, la visibilité des tâches en arrière-plan est limitée par rapport à l'interface CLI. Pour une meilleure visibilité, demandez à Claude de générer la commande afin que vous puissiez l'exécuter dans le terminal intégré de VS Code.

Connecter à des outils externes avec MCP

Les serveurs MCP (Model Context Protocol) donnent à Claude accès à des outils externes, des bases de données et des API.

Pour ajouter un serveur MCP, ouvrez le terminal intégré (**Ctrl+`** ou **Cmd+`**) et exécutez :

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/
```

Une fois configuré, demandez à Claude d'utiliser les outils (par exemple, « Review PR #456 »).

Pour gérer les serveurs MCP sans quitter VS Code, tapez **/mcp** dans le panneau de chat. Le dialogue de gestion MCP vous permet d'activer ou de désactiver les serveurs, de vous reconnecter à un serveur et de gérer l'authentification OAuth. Consultez la [documentation MCP](#) pour les serveurs disponibles.

Travailler avec git

Claude Code s'intègre à git pour vous aider avec les flux de travail de contrôle de version directement dans VS Code. Demandez à Claude de valider les modifications, de créer des demandes de tirage ou de travailler sur plusieurs branches.

Créer des commits et des demandes de tirage

Claude peut mettre en scène les modifications, écrire des messages de commit et créer des demandes de tirage en fonction de votre travail :

```
> commit my changes with a descriptive message
> create a pr for this feature
> summarize the changes I've made to the auth module
```

Lors de la création de demandes de tirage, Claude génère des descriptions basées sur les modifications de code réelles et peut ajouter du contexte sur les tests ou les décisions de mise en œuvre.

Utiliser les worktrees git pour les tâches parallèles

Utilisez l'indicateur `--worktree` (`-w`) pour démarrer Claude dans un worktree isolé avec ses propres fichiers et branche :

```
claude --worktree feature-auth
```

Chaque worktree maintient un état de fichier indépendant tout en partageant l'historique git. Cela empêche les instances de Claude d'interférer les unes avec les autres lorsqu'elles travaillent sur différentes tâches. Pour plus de détails, consultez [Exécuter des sessions parallèles avec les worktrees Git](#).

Utiliser des fournisseurs tiers

Par défaut, Claude Code se connecte directement à l'API d'Anthropic. Si votre organisation utilise Amazon Bedrock, Google Vertex AI ou Microsoft Foundry pour accéder à Claude, configurez l'extension pour utiliser votre fournisseur à la place :

Step 1: Désactiver l'invite de connexion

Ouvrez le [paramètre Désactiver l'invite de connexion](#) et cochez la case.

Vous pouvez également ouvrir les paramètres VS Code (`Cmd+`, sur Mac ou `Ctrl+`, sur Windows/Linux), recherchez ' Claude Code login ' et cochez **Désactiver l'invite de connexion**.

Step 2: Configurer votre fournisseur

Suivez le guide de configuration de votre fournisseur :

- [Claude Code sur Amazon Bedrock](#)
- [Claude Code sur Google Vertex AI](#)
- [Claude Code sur Microsoft Foundry](#)

Ces guides couvrent la configuration de votre fournisseur dans `~/.claude/settings.json`, ce qui garantit que vos paramètres sont partagés entre l'extension VS Code et l'interface CLI.

Sécurité et confidentialité

Votre code reste privé. Claude Code traite votre code pour fournir une assistance mais ne l'utilise pas pour entraîner les modèles. Pour plus de détails sur la gestion des données et comment refuser la journalisation, consultez [Données et confidentialité](#).

Avec les permissions d'édition automatique activées, Claude Code peut modifier les fichiers de configuration VS Code (comme `settings.json` ou `tasks.json`) que VS Code peut exécuter automatiquement. Pour réduire le risque lorsque vous travaillez avec du code non fiable :

- Activez le [Mode restreint VS Code](#) pour les espaces de travail non fiables
- Utilisez le mode d'approbation manuelle au lieu de l'acceptation automatique pour les modifications
- Examinez attentivement les modifications avant de les accepter

Corriger les problèmes courants

L'extension ne s'installe pas

- Assurez-vous que vous avez une version compatible de VS Code (1.98.0 ou supérieure)
- Vérifiez que VS Code a la permission d'installer des extensions
- Essayez d'installer directement depuis la [Place de marché VS Code](#)

L'icône Spark n'est pas visible

L'icône Spark apparaît dans la **Barre d'outils de l'éditeur** (coin supérieur droit de l'éditeur) lorsque vous avez un fichier ouvert. Si vous ne la voyez pas :

1. **Ouvrir un fichier** : L'icône nécessite qu'un fichier soit ouvert. Avoir juste un dossier ouvert ne suffit pas.
2. **Vérifier la version de VS Code** : Nécessite 1.98.0 ou supérieure (Aide → À propos)
3. **Redémarrer VS Code** : Exécutez « Developer: Reload Window » depuis la Palette de commandes
4. **Désactiver les extensions conflictuelles** : Désactivez temporairement les autres extensions IA (Cline, Continue, etc.)
5. **Vérifier la confiance de l'espace de travail** : L'extension ne fonctionne pas en Mode restreint

Vous pouvez également cliquer sur «  Claude Code » dans la **Barre d'état** (coin inférieur droit). Cela fonctionne même sans fichier ouvert. Vous pouvez également utiliser la **Palette de commandes** (`Cmd+Shift+P` / `Ctrl+Shift+P`) et taper « Claude Code ».

Claude Code ne répond jamais

Si Claude Code ne répond pas à vos invites :

1. **Vérifier votre connexion Internet** : Assurez-vous que vous avez une connexion Internet stable
2. **Démarrer une nouvelle conversation** : Essayez de démarrer une nouvelle conversation pour voir si le problème persiste
3. **Essayer l'interface CLI** : Exécutez `claude` depuis le terminal pour voir si vous obtenez des messages d'erreur plus détaillés

Si les problèmes persistent, [déposez un problème sur GitHub](#) avec des détails sur l'erreur.

Désinstaller l'extension

Pour désinstaller l'extension Claude Code :

1. Ouvrez la vue Extensions (`Cmd+Shift+X` sur Mac ou `Ctrl+Shift+X` sur Windows/Linux)
2. Recherchez « Claude Code »
3. Cliquez sur **Désinstaller**

Pour également supprimer les données d'extension et réinitialiser tous les paramètres :

```
rm -rf ~/.vscode/globalStorage/anthropic.claude-code
```

Pour une aide supplémentaire, consultez le [guide de dépannage](#).

Étapes suivantes

Maintenant que vous avez Claude Code configuré dans VS Code :

- [Explorez les flux de travail courants](#) pour tirer le meilleur parti de Claude Code
- [Configurez les serveurs MCP](#) pour étendre les capacités de Claude avec des outils externes. Ajoutez des serveurs en utilisant l'interface CLI, puis gérez-les avec `/mcp` dans le panneau de chat.
- [Configurez les paramètres Claude Code](#) pour personnaliser les commandes autorisées, les hooks et bien d'autres. Ces paramètres sont partagés entre l'extension et l'interface CLI.

JetBrains IDEs

Utilisez Claude Code avec les IDEs JetBrains, notamment IntelliJ, PyCharm, WebStorm et bien d'autres

Claude Code s'intègre aux IDEs JetBrains via un plugin dédié, offrant des fonctionnalités telles que l'affichage interactif des différences, le partage du contexte de sélection, et bien d'autres.

IDEs supportés

Le plugin Claude Code fonctionne avec la plupart des IDEs JetBrains, notamment :

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

Fonctionnalités

- **Lancement rapide** : Utilisez `Cmd+Esc` (Mac) ou `Ctrl+Esc` (Windows/Linux) pour ouvrir Claude Code directement depuis votre éditeur, ou cliquez sur le bouton Claude Code dans l'interface utilisateur
- **Affichage des différences** : Les modifications de code peuvent être affichées directement dans la visionneuse de différences de l'IDE au lieu du terminal
- **Contexte de sélection** : La sélection/l'onglet actuel dans l'IDE est automatiquement partagé avec Claude Code
- **Raccourcis de référence de fichier** : Utilisez `Cmd+Option+K` (Mac) ou `Alt+Ctrl+K` (Linux/Windows) pour insérer des références de fichier (par exemple, `@File#L1-99`)
- **Partage des diagnostics** : Les erreurs de diagnostic (lint, syntaxe, etc.) de l'IDE sont automatiquement partagées avec Claude au fur et à mesure que vous travaillez

Installation

Installation via la Marketplace

Trouvez et installez le [plugin Claude Code](#) depuis la marketplace JetBrains et redémarrez votre IDE.

Si vous n'avez pas encore installé Claude Code, consultez [notre guide de démarrage rapide](#) pour les instructions d'installation.

Note:

Après l'installation du plugin, vous devrez peut-être redémarrer complètement votre IDE pour que les modifications prennent effet.

Utilisation

Depuis votre IDE

Exécutez `claude` depuis le terminal intégré de votre IDE, et toutes les fonctionnalités d'intégration seront actives.

Depuis des terminaux externes

Utilisez la commande `/ide` dans n'importe quel terminal externe pour connecter Claude Code à votre IDE JetBrains et activer toutes les fonctionnalités :

```
claude
```

```
/ide
```

Si vous souhaitez que Claude ait accès aux mêmes fichiers que votre IDE, démarrez Claude Code à partir du même répertoire que la racine du projet de votre IDE.

Configuration

Paramètres de Claude Code

Configurez l'intégration de l'IDE via les paramètres de Claude Code :

1. Exécutez `claude`
2. Entrez la commande `/config`

3. Définissez l'outil de différence sur `auto` pour la détection automatique de l'IDE

Paramètres du plugin

Configurez le plugin Claude Code en accédant à **Paramètres** → **Outils** → **Claude Code [Beta]** :

Paramètres généraux

- **Commande Claude** : Spécifiez une commande personnalisée pour exécuter Claude (par exemple, `claude`, `/usr/local/bin/claude`, ou `npx @anthropic/claude`)
- **Supprimer la notification pour la commande Claude non trouvée** : Ignorez les notifications concernant la non-détection de la commande Claude
- **Activer l'utilisation d'Option+Entrée pour les invites multi-lignes** (macOS uniquement) : Lorsqu'elle est activée, Option+Entrée insère de nouvelles lignes dans les invites Claude Code. Désactivez si vous rencontrez des problèmes avec la touche Option capturée de manière inattendue (nécessite un redémarrage du terminal)
- **Activer les mises à jour automatiques** : Vérifiez automatiquement et installez les mises à jour du plugin (appliquées au redémarrage)

Tip:

Pour les utilisateurs WSL : Définissez `wsl -d Ubuntu -- bash -lic "claude"` comme votre commande Claude (remplacez `Ubuntu` par le nom de votre distribution WSL)

Configuration de la touche ESC

Si la touche ESC n'interrompt pas les opérations Claude Code dans les terminaux JetBrains :

1. Accédez à **Paramètres** → **Outils** → **Terminal**
2. Soit :
 - Décochez « Déplacer le focus vers l'éditeur avec Échap », soit
 - Cliquez sur « Configurer les raccourcis clavier du terminal » et supprimez le raccourci « Basculer le focus vers l'éditeur »
3. Appliquez les modifications

Cela permet à la touche ESC d'interrompre correctement les opérations Claude Code.

Configurations spéciales

Développement à distance

Warning:

Lors de l'utilisation du développement à distance JetBrains, vous devez installer le plugin sur l'hôte distant via **Paramètres** → **Plugin (Hôte)**.

Le plugin doit être installé sur l'hôte distant, et non sur votre machine cliente locale.

Configuration WSL

Warning:

Les utilisateurs WSL peuvent avoir besoin d'une configuration supplémentaire pour que la détection de l'IDE fonctionne correctement. Consultez notre [guide de dépannage WSL](#) pour des instructions de configuration détaillées.

La configuration WSL peut nécessiter :

- Une configuration appropriée du terminal
- Des ajustements du mode de mise en réseau
- Des mises à jour des paramètres du pare-feu

Dépannage

Le plugin ne fonctionne pas

- Assurez-vous que vous exécutez Claude Code à partir du répertoire racine du projet
- Vérifiez que le plugin JetBrains est activé dans les paramètres de l'IDE
- Redémarrez complètement l'IDE (vous devrez peut-être le faire plusieurs fois)
- Pour le développement à distance, assurez-vous que le plugin est installé sur l'hôte distant

IDE non détecté

- Vérifiez que le plugin est installé et activé
- Redémarrez complètement l'IDE
- Vérifiez que vous exécutez Claude Code à partir du terminal intégré
- Pour les utilisateurs WSL, consultez le [guide de dépannage WSL](#)

Commande non trouvée

Si cliquer sur l'icône Claude affiche « commande non trouvée » :

1. Vérifiez que Claude Code est installé : `npm list -g @anthropic-ai/claude-code`
2. Configurez le chemin de la commande Claude dans les paramètres du plugin
3. Pour les utilisateurs WSL, utilisez le format de commande WSL mentionné dans la section configuration

Considérations de sécurité

Lorsque Claude Code s'exécute dans un IDE JetBrains avec les permissions d'édition automatique activées, il peut être en mesure de modifier les fichiers de configuration de l'IDE qui peuvent être exécutés automatiquement par votre IDE. Cela peut augmenter le risque d'exécution de Claude Code en mode édition automatique et permettre de contourner les invites de permission de Claude Code pour l'exécution bash.

Lors de l'exécution dans les IDEs JetBrains, considérez :

- L'utilisation du mode d'approbation manuelle pour les modifications
- La prise de précautions supplémentaires pour vous assurer que Claude n'est utilisé qu'avec des invites de confiance
- La sensibilisation aux fichiers auxquels Claude Code a accès pour les modifier

Pour une aide supplémentaire, consultez notre [guide de dépannage](#).

Utiliser Claude Code Desktop

Tirez le meilleur parti de Claude Code Desktop : sessions parallèles avec isolation Git, examen visuel des différences, aperçus d'applications, surveillance des PR, modes de permission, connecteurs et configuration d'entreprise.

L'onglet Code dans l'application Claude Desktop vous permet d'utiliser Claude Code via une interface graphique au lieu du terminal.

Desktop ajoute ces capacités à l'expérience standard de Claude Code :

- [Examen visuel des différences](#) avec commentaires en ligne
- [Aperçu d'application en direct](#) avec serveurs de développement
- [Surveillance des PR GitHub](#) avec correction automatique et fusion automatique
- [Sessions parallèles](#) avec isolation automatique des Git worktrees
- [Tâches planifiées](#) qui exécutent Claude selon un calendrier récurrent
- [Connecteurs](#) pour GitHub, Slack, Linear et bien d'autres
- Environnements locaux, [SSH](#) et [cloud](#)

Tip:

Nouveau sur Desktop ? Commencez par [Démarrer](#) pour installer l'application et effectuer votre première modification.

Cette page couvre [travailler avec le code](#), [gérer les sessions](#), [étendre Claude Code](#), [tâches planifiées](#) et [configuration](#). Elle inclut également une [comparaison CLI](#) et [dépannage](#).

Démarrer une session

Avant d'envoyer votre premier message, configurez quatre choses dans la zone de message :

- **Environnement** : choisissez où Claude s'exécute. Sélectionnez **Local** pour votre machine, **Remote** pour les sessions cloud hébergées par Anthropic, ou une [connexion SSH](#) pour une machine distante que vous gérez. Voir [configuration de l'environnement](#).
- **Dossier du projet** : sélectionnez le dossier ou le référentiel dans lequel Claude travaille. Pour les sessions distantes, vous pouvez ajouter [plusieurs référentiels](#).

- **Modèle** : choisissez un [modèle](#) dans la liste déroulante à côté du bouton d'envoi. Le modèle est verrouillé une fois la session démarrée.
- **Mode de permission** : choisissez le niveau d'autonomie de Claude à partir du [sélecteur de mode](#). Vous pouvez modifier ceci pendant la session.

Tapez votre tâche et appuyez sur **Entrée** pour démarrer. Chaque session suit son propre contexte et les modifications indépendamment.

Travailler avec le code

Donnez à Claude le bon contexte, contrôlez le volume de travail qu'il effectue seul et examinez ce qu'il a modifié.

Utiliser la zone de message

Tapez ce que vous voulez que Claude fasse et appuyez sur **Entrée** pour envoyer. Claude lit vos fichiers de projet, effectue des modifications et exécute des commandes en fonction de votre [mode de permission](#). Vous pouvez interrompre Claude à tout moment : cliquez sur le bouton d'arrêt ou tapez votre correction et appuyez sur **Entrée**. Claude arrête ce qu'il fait et s'ajuste en fonction de votre entrée.

Le bouton + à côté de la zone de message vous donne accès aux pièces jointes de fichiers, [skills](#), [connecteurs](#) et [plugins](#).

Ajouter des fichiers et du contexte aux messages

La zone de message supporte deux façons d'apporter du contexte externe :

- **Fichiers @mention** : tapez @ suivi d'un nom de fichier pour ajouter un fichier au contexte de la conversation. Claude peut alors lire et référencer ce fichier.
- **Joindre des fichiers** : joignez des images, des PDF et d'autres fichiers à votre message en utilisant le bouton de pièce jointe, ou glissez-déposez les fichiers directement dans le message. Ceci est utile pour partager des captures d'écran de bugs, des maquettes de conception ou des documents de référence.

Choisir un mode de permission

Les modes de permission contrôlent le niveau d'autonomie de Claude pendant une session : s'il demande avant de modifier des fichiers, d'exécuter des commandes ou les deux. Vous pouvez changer de mode à tout moment en utilisant le sélecteur de mode à côté du bouton d'envoi. Commencez par Demander les permissions pour voir exactement ce que Claude fait, puis passez à Accepter automatiquement les modifications ou Plan mode à mesure que vous vous sentez à l'aise.

Mode	Clé de paramètres	Comportement
Demander les permissions	<code>default</code>	Claude demande avant de modifier des fichiers ou d'exécuter des commandes. Vous voyez une différence et pouvez accepter ou rejeter chaque modification. Recommandé pour les nouveaux utilisateurs.
Accepter automatiquement les modifications	<code>acceptEdits</code>	Claude accepte automatiquement les modifications de fichiers mais demande toujours avant d'exécuter les commandes du terminal. Utilisez ceci quand vous faites confiance aux modifications de fichiers et voulez une itération plus rapide.
Plan mode	<code>plan</code>	Claude analyse votre code et crée un plan sans modifier les fichiers ou exécuter les commandes. Bon pour les tâches complexes où vous voulez d'abord examiner l'approche.

Mode	Clé de paramètres	Comportement
Contourner les permissions	<code>bypassPermissions</code>	Claude s'exécute sans aucune invite de permission, équivalent à <code>--dangerously-skip-permissions</code> dans la CLI. Activez dans vos Paramètres → Claude Code sous « Autoriser le mode de contournement des permissions ». Utilisez uniquement dans les conteneurs sandboxés ou les machines virtuelles. Les administrateurs d'entreprise peuvent désactiver cette option.

Le mode de permission `dontAsk` est disponible uniquement dans la [CLI](#).

Tip:

Commencez les tâches complexes en Plan mode pour que Claude trace une approche avant de faire des modifications. Une fois que vous approuvez le plan, passez à Accepter automatiquement les modifications ou Demander les permissions pour l'exécuter. Voir [explorer d'abord](#), [puis planifier](#), [puis coder](#) pour plus d'informations sur ce flux de travail.

Les sessions distantes supportent Accepter automatiquement les modifications et Plan mode. Demander les permissions n'est pas disponible car les sessions distantes acceptent automatiquement les modifications de fichiers par défaut, et Contourner les permissions n'est pas disponible car l'environnement distant est déjà sandboxé.

Les administrateurs d'entreprise peuvent restreindre les modes de permission disponibles. Voir [configuration d'entreprise](#) pour les détails.

Aperçu de votre application

Claude peut démarrer un serveur de développement et ouvrir un navigateur intégré pour vérifier ses modifications. Cela fonctionne pour les applications web frontend ainsi que les serveurs backend : Claude peut tester les points de terminaison API, afficher les journaux du serveur et itérer sur les problèmes qu'il trouve. Dans la plupart des cas, Claude démarre

le serveur automatiquement après la modification des fichiers du projet. Vous pouvez également demander à Claude de prévisualiser à tout moment. Par défaut, Claude [vérifie automatiquement](#) les modifications après chaque modification.

À partir du panneau d'aperçu, vous pouvez :

- Interagir avec votre application en cours d'exécution directement dans le navigateur intégré
- Regarder Claude vérifier ses propres modifications automatiquement : il prend des captures d'écran, inspecte le DOM, clique sur les éléments, remplit les formulaires et corrige les problèmes qu'il trouve
- Démarrer ou arrêter les serveurs à partir de la liste déroulante **Aperçu** dans la barre d'outils de la session
- Conserver les cookies et le stockage local entre les redémarrages du serveur en sélectionnant **Conserver les sessions** dans la liste déroulante, afin que vous n'ayez pas à vous reconnecter pendant le développement
- Modifier la configuration du serveur ou arrêter tous les serveurs à la fois

Claude crée la configuration initiale du serveur en fonction de votre projet. Si votre application utilise une commande de développement personnalisée, modifiez `.claude/launch.json` pour correspondre à votre configuration. Voir [Configurer les serveurs d'aperçu](#) pour la référence complète.

Pour effacer les données de session enregistrées, basculez **Conserver les sessions d'aperçu** dans Paramètres → Claude Code. Pour désactiver complètement l'aperçu, basculez **Aperçu** dans Paramètres → Claude Code.

Examiner les modifications avec la vue de différence

Après que Claude ait modifié votre code, la vue de différence vous permet d'examiner les modifications fichier par fichier avant de créer une demande de tirage.

Quand Claude modifie des fichiers, un indicateur de statistiques de différence apparaît montrant le nombre de lignes ajoutées et supprimées, comme `+12 -1`. Cliquez sur cet indicateur pour ouvrir la visionneuse de différences, qui affiche une liste de fichiers à gauche et les modifications pour chaque fichier à droite.

Pour commenter des lignes spécifiques, cliquez sur n'importe quelle ligne dans la différence pour ouvrir une boîte de commentaire. Tapez votre retour et appuyez sur **Entrée** pour ajouter le commentaire. Après avoir ajouté des commentaires à plusieurs lignes, soumettez tous les commentaires à la fois :

- **macOS** : appuyez sur **Cmd+Entrée**

- **Windows** : appuyez sur **Ctrl+Entrée**

Claude lit vos commentaires et effectue les modifications demandées, qui apparaissent comme une nouvelle différence que vous pouvez examiner.

Examiner votre code

Dans la vue de différence, cliquez sur **Examiner le code** dans la barre d'outils en haut à droite pour demander à Claude d'évaluer les modifications avant de les valider. Claude examine les différences actuelles et laisse des commentaires directement dans la vue de différence. Vous pouvez répondre à n'importe quel commentaire ou demander à Claude de réviser.

L'examen se concentre sur les problèmes à haut signal : erreurs de compilation, erreurs logiques définies, vulnérabilités de sécurité et bugs évidents. Il ne signale pas le style, le formatage, les problèmes préexistants ou quoi que ce soit qu'un linter attraperait.

Surveiller l'état de la demande de tirage

Après avoir ouvert une demande de tirage, une barre d'état CI apparaît dans la session. Claude Code utilise la CLI GitHub pour interroger les résultats des vérifications et afficher les défaillances.

- **Correction automatique** : quand elle est activée, Claude tente automatiquement de corriger les vérifications CI défaillantes en lisant la sortie de défaillance et en itérant.
- **Fusion automatique** : quand elle est activée, Claude fusionne la PR une fois que toutes les vérifications réussissent. La méthode de fusion est squash. La fusion automatique doit être [activée dans les paramètres de votre référentiel GitHub](#) pour que cela fonctionne.

Utilisez les bascules **Correction automatique** et **Fusion automatique** dans la barre d'état CI pour activer l'une ou l'autre option. Claude Code envoie également une notification de bureau quand CI se termine.

Note:

La surveillance des PR nécessite que la [CLI GitHub \(gh \)](#) soit installée et authentifiée sur votre machine. Si **gh** n'est pas installé, Desktop vous invite à l'installer la première fois que vous essayez de créer une PR.

Gérer les sessions

Chaque session est une conversation indépendante avec son propre contexte et ses propres modifications. Vous pouvez exécuter plusieurs sessions en parallèle ou envoyer du travail vers le cloud.

Travailler en parallèle avec les sessions

Cliquez sur **+ Nouvelle session** dans la barre latérale pour travailler sur plusieurs tâches en parallèle. Pour les référentiels Git, chaque session obtient sa propre copie isolée de votre projet en utilisant [Git worktrees](#), donc les modifications dans une session n'affectent pas les autres sessions jusqu'à ce que vous les validiez.

Les worktrees sont stockés dans `<project-root>/ .claude/worktrees/` par défaut. Vous pouvez modifier ceci en un répertoire personnalisé dans Paramètres → Claude Code sous 'Emplacement du worktree'. Vous pouvez également définir un préfixe de branche qui est ajouté au début de chaque nom de branche worktree, ce qui est utile pour garder les branches créées par Claude organisées. Pour supprimer un worktree quand vous avez terminé, survolez la session dans la barre latérale et cliquez sur l'icône d'archive.

Note:

L'isolation des sessions nécessite [Git](#). La plupart des Macs incluent Git par défaut. Exécutez `git --version` dans Terminal pour vérifier. Sur Windows, Git est requis pour que l'onglet Code fonctionne : [téléchargez Git pour Windows](#), installez-le et redémarrez l'application. Si vous rencontrez des erreurs Git, essayez une session Cowork pour aider à dépanner votre configuration.

Utilisez l'icône de filtre en haut de la barre latérale pour filtrer les sessions par statut (Actif, Archivé) et environnement (Local, Cloud). Pour renommer une session ou vérifier l'utilisation du contexte, cliquez sur le titre de la session dans la barre d'outils en haut de la session active. Quand le contexte se remplit, Claude résume automatiquement la conversation et continue à travailler. Vous pouvez également taper `/compact` pour déclencher la compaction plus tôt et libérer de l'espace de contexte. Voir [la fenêtre de contexte](#) pour les détails sur le fonctionnement de la compaction.

Exécuter des tâches longues à distance

Pour les refactorisations importantes, les suites de tests, les migrations ou autres tâches longues, sélectionnez **Remote** au lieu de **Local** au démarrage d'une session. Les sessions distantes s'exécutent sur l'infrastructure cloud d'Anthropic et continuent même si vous

fermez l'application ou arrêtez votre ordinateur. Revenez à tout moment pour voir la progression ou orienter Claude dans une direction différente. Vous pouvez également surveiller les sessions distantes à partir de claude.ai/code ou de l'application Claude iOS.

Les sessions distantes supportent également plusieurs référentiels. Après avoir sélectionné un environnement cloud, cliquez sur le bouton + à côté de la pastille du référentiel pour ajouter des référentiels supplémentaires à la session. Chaque référentiel obtient son propre sélecteur de branche. Ceci est utile pour les tâches qui s'étendent sur plusieurs bases de code, comme la mise à jour d'une bibliothèque partagée et ses consommateurs.

Voir [Claude Code sur le web](#) pour plus d'informations sur le fonctionnement des sessions distantes.

Continuer sur une autre surface

Le menu **Continuer dans**, accessible à partir de l'icône VS Code en bas à droite de la barre d'outils de la session, vous permet de déplacer votre session vers une autre surface :

- **Claude Code sur le Web** : envoie votre session locale pour continuer à s'exécuter à distance. Desktop pousse votre branche, génère un résumé de la conversation et crée une nouvelle session distante avec le contexte complet. Vous pouvez ensuite choisir d'archiver la session locale ou de la conserver. Cela nécessite un arbre de travail propre et n'est pas disponible pour les sessions SSH.
- **Votre IDE** : ouvre votre projet dans un IDE supporté au répertoire de travail actuel.

Étendre Claude Code

Connectez les services externes, ajoutez des flux de travail réutilisables, personnalisez le comportement de Claude et configurez les serveurs d'aperçu.

Connecter les outils externes

Pour les sessions locales et [SSH](#), cliquez sur le bouton + à côté de la zone de message et sélectionnez **Connecteurs** pour ajouter des intégrations comme Google Calendar, Slack, GitHub, Linear, Notion et bien d'autres. Vous pouvez ajouter des connecteurs avant ou pendant une session. Les connecteurs ne sont pas disponibles pour les sessions distantes.

Pour gérer ou déconnecter les connecteurs, allez à Paramètres → Connecteurs dans l'application de bureau, ou sélectionnez **Gérer les connecteurs** dans le menu Connecteurs dans la zone de message.

Une fois connecté, Claude peut lire votre calendrier, envoyer des messages, créer des problèmes et interagir avec vos outils directement. Vous pouvez demander à Claude quels connecteurs sont configurés dans votre session.

Les connecteurs sont [MCP servers](#) avec un flux de configuration graphique. Utilisez-les pour une intégration rapide avec les services supportés. Pour les intégrations non listées dans Connecteurs, ajoutez les MCP servers manuellement via [fichiers de paramètres](#). Vous pouvez également [créer des connecteurs personnalisés](#).

Utiliser les skills

[Skills](#) étendent ce que Claude peut faire. Claude les charge automatiquement quand ils sont pertinents, ou vous pouvez en invoquer un directement : tapez `/` dans la zone de message ou cliquez sur le bouton + et sélectionnez **Slash commands** pour parcourir ce qui est disponible. Cela inclut [les commandes intégrées](#), vos [skills personnalisés](#), les skills du projet de votre base de code et les skills de tout [plugin installé](#). Sélectionnez-en un et il apparaît en surbrillance dans le champ d'entrée. Tapez votre tâche après et envoyez comme d'habitude.

Installer les plugins

[Plugins](#) sont des packages réutilisables qui ajoutent des skills, des agents, des hooks, des MCP servers et des configurations LSP à Claude Code. Vous pouvez installer les plugins à partir de l'application de bureau sans utiliser le terminal.

Pour les sessions locales et [SSH](#), cliquez sur le bouton + à côté de la zone de message et sélectionnez **Plugins** pour voir vos plugins installés et leurs commandes. Pour ajouter un plugin, sélectionnez **Ajouter un plugin** dans le sous-menu pour ouvrir le navigateur de plugins, qui affiche les plugins disponibles à partir de vos [marketplaces](#) configurés, y compris le marketplace officiel d'Anthropic. Sélectionnez **Gérer les plugins** pour activer, désactiver ou désinstaller les plugins.

Les plugins peuvent être limités à votre compte utilisateur, un projet spécifique ou local uniquement. Les plugins ne sont pas disponibles pour les sessions distantes. Pour la référence complète des plugins, y compris la création de vos propres plugins, voir [plugins](#).

Configurer les serveurs d'aperçu

Claude détecte automatiquement votre configuration de serveur de développement et stocke la configuration dans `.claude/launch.json` à la racine du dossier que vous avez sélectionné au démarrage de la session. L'aperçu utilise ce dossier comme répertoire de travail, donc si vous avez sélectionné un dossier parent, les sous-dossiers avec leurs propres serveurs de développement ne seront pas détectés automatiquement. Pour travailler avec le serveur d'un sous-dossier, soit démarrez une session dans ce dossier directement, soit ajoutez une configuration manuellement.

Pour personnaliser le démarrage de votre serveur, par exemple pour utiliser `yarn dev` au lieu de `npm run dev` ou pour modifier le port, modifiez le fichier manuellement ou cliquez sur **Modifier la configuration** dans la liste déroulante Aperçu pour l'ouvrir dans votre éditeur de code. Le fichier supporte JSON avec commentaires.

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

Vous pouvez définir plusieurs configurations pour exécuter différents serveurs à partir du même projet, comme un frontend et une API. Voir les [exemples](#) ci-dessous.

Vérification automatique des modifications

Quand `autoVerify` est activé, Claude vérifie automatiquement les modifications de code après la modification des fichiers. Il prend des captures d'écran, vérifie les erreurs et confirme que les modifications fonctionnent avant de terminer sa réponse.

La vérification automatique est activée par défaut. Désactivez-la par projet en ajoutant `"autoVerify": false` à `.claude/launch.json`, ou basculez-la à partir du menu déroulant **Aperçu**.

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [...]
}
```

Quand elle est désactivée, les outils d'aperçu sont toujours disponibles et vous pouvez demander à Claude de vérifier à tout moment. La vérification automatique la rend automatique après chaque modification.

Champs de configuration

Chaque entrée dans le tableau `configurations` accepte les champs suivants :

Champ	Type	Description
<code>name</code>	string	Un identifiant unique pour ce serveur
<code>runtimeExecutable</code>	string	La commande à exécuter, comme <code>npm</code> , <code>yarn</code> ou <code>node</code>
<code>runtimeArgs</code>	string[]	Arguments passés à <code>runtimeExecutable</code> , comme <code>["run", "dev"]</code>
<code>port</code>	number	Le port sur lequel votre serveur écoute. Par défaut 3000
<code>cwd</code>	string	Répertoire de travail relatif à la racine de votre projet. Par défaut la racine du projet. Utilisez <code>\$ {workspaceFolder}</code> pour référencer la racine du projet explicitement
<code>env</code>	object	Variables d'environnement supplémentaires comme paires clé-valeur, comme <code>{ "NODE_ENV": "development" }</code> . Ne mettez pas de secrets ici car ce fichier est validé dans votre référentiel. Les secrets définis dans votre profil shell sont hérités automatiquement.
<code>autoPort</code>	boolean	Comment gérer les conflits de port. Voir ci-dessous

Champ	Type	Description
<code>program</code>	<code>string</code>	Un script à exécuter avec <code>node</code> . Voir quand utiliser <code>program</code> vs <code>runtimeExecutable</code>
<code>args</code>	<code>string[]</code>	Arguments passés à <code>program</code> . Utilisé uniquement quand <code>program</code> est défini

Quand utiliser `program` vs `runtimeExecutable`

Utilisez `runtimeExecutable` avec `runtimeArgs` pour démarrer un serveur de développement via un gestionnaire de packages. Par exemple, `"runtimeExecutable": "npm"` avec `"runtimeArgs": ["run", "dev"]` exécute `npm run dev`.

Utilisez `program` quand vous avez un script autonome que vous voulez exécuter avec `node` directement. Par exemple, `"program": "server.js"` exécute `node server.js`. Passez des drapeaux supplémentaires avec `args`.

Conflits de port

Le champ `autoPort` contrôle ce qui se passe quand votre port préféré est déjà utilisé :

- **true** : Claude trouve et utilise un port libre automatiquement. Convient à la plupart des serveurs de développement.
- **false** : Claude échoue avec une erreur. Utilisez ceci quand votre serveur doit utiliser un port spécifique, comme pour les rappels OAuth ou les listes blanches CORS.
- **Non défini (par défaut)** : Claude demande si le serveur a besoin de ce port exact, puis enregistre votre réponse.

Quand Claude choisit un port différent, il passe le port assigné à votre serveur via la variable d'environnement `PORT`.

Exemples

Ces configurations montrent les configurations courantes pour différents types de projets :

Next.js

Cette configuration exécute une application Next.js en utilisant Yarn sur le port 3000 :

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

Plusieurs serveurs

Pour un monorepo avec un frontend et un serveur API, définissez plusieurs configurations. Le frontend utilise `autoPort: true` pour qu'il choisisse un port libre si 3000 est pris, tandis que le serveur API nécessite le port 8080 exactement :


```
{  
  "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  
    }  
  ]  
}
```

Script Node.js

Pour exécuter un script Node.js directement au lieu d'utiliser une commande du gestionnaire de packages, utilisez le champ `program` :

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

Planifier les tâches récurrentes

Les tâches planifiées démarrent une nouvelle session locale automatiquement à une heure et une fréquence que vous choisissez. Utilisez-les pour le travail récurrent comme les examens de code quotidiens, les vérifications de mise à jour des dépendances ou les briefings matinaux qui tirent de votre calendrier et de votre boîte de réception.

Les tâches s'exécutent sur votre machine, donc l'application de bureau doit être ouverte et votre ordinateur éveillé pour qu'elles se déclenchent. Voir [Comment les tâches planifiées s'exécutent](#) pour les détails sur les exécutions manquées et le comportement de rattrapage.

Note:

Par défaut, les tâches planifiées s'exécutent contre l'état dans lequel se trouve votre répertoire de travail, y compris les modifications non validées. Activez le basculement `worktree` dans l'entrée du message pour donner à chaque exécution son propre Git worktree isolé, de la même manière que les [sessions parallèles](#) fonctionnent.

Pour créer une tâche planifiée, cliquez sur **Planifier** dans la barre latérale, puis + **Nouvelle tâche**. Configurez ces champs :

Champ	Description
Nom	Identifiant pour la tâche. Converti en kebab-case minuscule et utilisé comme nom de dossier sur le disque. Doit être unique parmi vos tâches.
Description	Résumé court affiché dans la liste des tâches.

Champ	Description
Message	Les instructions envoyées à Claude quand la tâche s'exécute. Écrivez ceci de la même manière que vous écririez n'importe quel message dans la zone de message. L'entrée du message inclut également les contrôles pour le modèle, le mode de permission, le dossier de travail et le worktree.
Fré- quence	La fréquence à laquelle la tâche s'exécute. Voir options de fréquence ci-dessous.

Vous pouvez également créer une tâche en décrivant ce que vous voulez dans n'importe quelle session. Par exemple, « configurer un examen de code quotidien qui s'exécute chaque matin à 9h ».

Options de fréquence

- **Manuel** : pas de calendrier, s'exécute uniquement quand vous cliquez sur **Exécuter maintenant**. Utile pour enregistrer un message que vous déclenchez à la demande
- **Horaire** : s'exécute chaque heure. Chaque tâche obtient un décalage fixe de jusqu'à 10 minutes à partir de l'heure pour échelonner le trafic API
- **Quotidien** : affiche un sélecteur d'heure, par défaut 9h00 heure locale
- **Jours de semaine** : identique à Quotidien mais saute samedi et dimanche
- **Hebdomadaire** : affiche un sélecteur d'heure et un sélecteur de jour

Pour les intervalles que le sélecteur n'offre pas (toutes les 15 minutes, premier de chaque mois, etc.), demandez à Claude dans n'importe quelle session Desktop de définir le calendrier. Utilisez le langage naturel ; par exemple, « planifier une tâche pour exécuter tous les tests toutes les 6 heures ».

Comment les tâches planifiées s'exécutent

Les tâches planifiées s'exécutent localement sur votre machine. Desktop vérifie le calendrier chaque minute tandis que l'application est ouverte et démarre une session fraîche quand une tâche est due, indépendamment de toute session manuelle que vous avez ouverte. Chaque tâche obtient un délai fixe de jusqu'à 10 minutes après l'heure planifiée pour échelonner le trafic API. Le délai est déterministe : la même tâche démarre toujours au même décalage.

Quand une tâche se déclenche, vous recevez une notification de bureau et une nouvelle session apparaît sous une section **Planifiée** dans la barre latérale. Ouvrez-la pour voir ce que Claude a fait, examiner les modifications ou répondre aux invites de permission. La session fonctionne comme n'importe quelle autre : Claude peut modifier des fichiers, exécuter des commandes, créer des validations et ouvrir des demandes de tirage.

Les tâches s'exécutent uniquement tandis que l'application de bureau est en cours d'exécution et votre ordinateur est éveillé. Si votre ordinateur dort pendant une heure planifiée, l'exécution est ignorée. Pour empêcher le sommeil inactif, activez **Garder l'ordinateur éveillé** dans Paramètres sous **Application de bureau** → **Général**. Fermer le couvercle de l'ordinateur portable le met toujours en sommeil.

Exécutions manquées

Quand l'application démarre ou votre ordinateur se réveille, Desktop vérifie si chaque tâche a manqué des exécutions au cours des sept derniers jours. Si c'est le cas, Desktop démarre exactement une exécution de rattrapage pour l'heure manquée la plus récente et rejette tout ce qui est plus ancien. Une tâche quotidienne qui a manqué six jours s'exécute une fois au réveil. Desktop affiche une notification quand une exécution de rattrapage démarre.

Gardez ceci à l'esprit quand vous écrivez des messages. Une tâche planifiée pour 9h pourrait s'exécuter à 23h si votre ordinateur a dormi toute la journée. Si le timing est important, ajoutez des garde-fous au message lui-même, par exemple : « Examinez uniquement les validations d'aujourd'hui. S'il est après 17h, ignorez l'examen et publiez simplement un résumé de ce qui a été manqué ».

Permissions pour les tâches planifiées

Chaque tâche a son propre mode de permission, que vous définissez lors de la création ou de la modification de la tâche. Les règles d'autorisation de `~/claude/settings.json` s'appliquent également aux sessions de tâches planifiées. Si une tâche s'exécute en mode Ask et doit exécuter un outil pour lequel elle n'a pas de permission, l'exécution s'arrête jusqu'à ce que vous l'approuviez. La session reste ouverte dans la barre latérale pour que vous puissiez répondre plus tard.

Pour éviter les arrêts, cliquez sur **Exécuter maintenant** après avoir créé une tâche, regardez les invites de permission et sélectionnez « toujours autoriser » pour chacune. Les exécutions futures de cette tâche approuvent automatiquement les mêmes outils sans demander. Vous pouvez examiner et révoquer ces approbations à partir de la page de détail de la tâche.

Gérer les tâches planifiées

Cliquez sur une tâche dans la liste **Planifier** pour ouvrir sa page de détail. À partir d'ici, vous pouvez :

- **Exécuter maintenant** : démarrer la tâche immédiatement sans attendre l'heure planifiée suivante

- **Basculer les répétitions** : mettre en pause ou reprendre les exécutions planifiées sans supprimer la tâche
- **Modifier** : modifier le message, la fréquence, le dossier ou d'autres paramètres
- **Examiner l'historique** : voir chaque exécution passée, y compris celles qui ont été ignorées parce que votre ordinateur dormait
- **Examiner les permissions autorisées** : voir et révoquer les approbations d'outils enregistrées pour cette tâche à partir du panneau **Toujours autorisé**
- **Supprimer** : supprimer la tâche et archiver toutes les sessions qu'elle a créées

Vous pouvez également gérer les tâches en demandant à Claude dans n'importe quelle session Desktop. Par exemple, « mettre en pause ma tâche de dépendance-audit », « supprimer la tâche de préparation-standup » ou « afficher mes tâches planifiées ».

Pour modifier le message d'une tâche sur le disque, ouvrez `~/.claude/scheduled-tasks/<task-name>/SKILL.md` (ou sous `CLAUDE_CONFIG_DIR` si défini). Le fichier utilise le frontmatter YAML pour `name` et `description`, avec le message comme corps. Les modifications prennent effet à la prochaine exécution. Le calendrier, le dossier, le modèle et l'état activé ne sont pas dans ce fichier : modifiez-les via le formulaire Modifier ou demandez à Claude.

Configuration de l'environnement

L'environnement que vous choisissez au [démarrage d'une session](#) détermine où Claude s'exécute et comment vous vous connectez :

- **Local** : s'exécute sur votre machine avec accès direct à vos fichiers
- **Remote** : s'exécute sur l'infrastructure cloud d'Anthropic. Les sessions continuent même si vous fermez l'application.
- **SSH** : s'exécute sur une machine distante à laquelle vous vous connectez via SSH, comme vos propres serveurs, des machines virtuelles cloud ou des conteneurs de développement

Sessions locales

Les sessions locales héritent des variables d'environnement de votre shell. Si vous avez besoin de variables supplémentaires, définissez-les dans votre profil shell, comme `~/.zshrc` ou `~/.bashrc`, et redémarrez l'application de bureau. Voir [variables d'environnement](#) pour la liste complète des variables supportées.

[Extended thinking](#) est activé par défaut, ce qui améliore les performances sur les tâches de raisonnement complexe mais utilise des tokens supplémentaires. Pour désactiver complètement la réflexion, définissez `MAX_THINKING_TOKENS=0` dans votre profil shell. Sur Opus, `MAX_THINKING_TOKENS` est ignoré sauf pour `0` car le raisonnement adaptatif contrôle la profondeur de la réflexion à la place.

Sessions distantes

Les sessions distantes continuent en arrière-plan même si vous fermez l'application. L'utilisation compte vers les limites de votre [plan d'abonnement](#) sans frais de calcul séparés.

Vous pouvez créer des environnements cloud personnalisés avec différents niveaux d'accès réseau et variables d'environnement. Sélectionnez la liste déroulante d'environnement au démarrage d'une session distante et choisissez **Ajouter un environnement**. Voir [environnements cloud](#) pour les détails sur la configuration de l'accès réseau et des variables d'environnement.

Sessions SSH

Les sessions SSH vous permettent d'exécuter Claude Code sur une machine distante tout en utilisant l'application de bureau comme interface. Ceci est utile pour travailler avec des bases de code qui vivent sur des machines virtuelles cloud, des conteneurs de développement ou des serveurs avec du matériel ou des dépendances spécifiques.

Pour ajouter une connexion SSH, cliquez sur la liste déroulante d'environnement avant de démarrer une session et sélectionnez + **Ajouter une connexion SSH**. La boîte de dialogue demande :

- **Nom** : une étiquette conviviale pour cette connexion
- **SSH Host** : `user@hostname` ou un hôte défini dans `~/.ssh/config`
- **SSH Port** : par défaut 22 si laissé vide, ou utilise le port de votre configuration SSH
- **Identity File** : chemin vers votre clé privée, comme `~/.ssh/id_rsa`. Laissez vide pour utiliser la clé par défaut ou votre configuration SSH.

Une fois ajoutée, la connexion apparaît dans la liste déroulante d'environnement. Sélectionnez-la pour démarrer une session sur cette machine. Claude s'exécute sur la machine distante avec accès à ses fichiers et outils.

Claude Code doit être installé sur la machine distante. Une fois connecté, les sessions SSH supportent les modes de permission, les connecteurs, les plugins et les MCP servers.

Configuration d'entreprise

Les organisations sur les plans Teams ou Enterprise peuvent gérer le comportement de l'application de bureau via les contrôles de la console d'administration, les fichiers de paramètres gérés et les politiques de gestion des appareils.

Contrôles de la console d'administration

Ces paramètres sont configurés via la [console de paramètres d'administration](#) :

- **Activer ou désactiver l'onglet Code** : contrôlez si les utilisateurs de votre organisation peuvent accéder à Claude Code dans l'application de bureau
- **Désactiver le mode Contourner les permissions** : empêchez les utilisateurs de votre organisation d'activer le mode de contournement des permissions
- **Désactiver Claude Code sur le web** : activez ou désactivez les sessions distantes pour votre organisation

Paramètres gérés

Les paramètres gérés remplacent les paramètres du projet et de l'utilisateur et s'appliquent quand Desktop génère des sessions CLI. Vous pouvez définir ces clés dans le fichier [paramètres gérés](#) de votre organisation ou les pousser à distance via la console d'administration.

Clé	Description
<code>disableBypassPermissionsMode</code>	définissez à "disable" pour empêcher les utilisateurs d'activer le mode de contournement des permissions. Voir paramètres gérés .

Pour la liste complète des paramètres gérés uniquement, y compris

`allowManagedPermissionRulesOnly` et `allowManagedHooksOnly`, voir [paramètres gérés uniquement](#).

Les paramètres gérés distants téléchargés via la console d'administration s'appliquent actuellement uniquement aux sessions CLI et IDE. Pour les restrictions spécifiques à Desktop, utilisez les contrôles de la console d'administration ci-dessus.

Politiques de gestion des appareils

Les équipes informatiques peuvent gérer l'application de bureau via MDM sur macOS ou la politique de groupe sur Windows. Les politiques disponibles incluent l'activation ou la désactivation de la fonctionnalité Claude Code, le contrôle des mises à jour automatiques et la définition d'une URL de déploiement personnalisée.

- **macOS** : configurez via le domaine de préférence `com.anthropic.Claude` en utilisant des outils comme Jamf ou Kandji
- **Windows** : configurez via le registre à `SOFTWARE\Policies\Claude`

Authentification et SSO

Les organisations d'entreprise peuvent exiger SSO pour tous les utilisateurs. Voir [authentification](#) pour les détails au niveau du plan et [Configuration de SSO](#) pour la configuration SAML et OIDC.

Gestion des données

Claude Code traite votre code localement dans les sessions locales ou sur l'infrastructure cloud d'Anthropic dans les sessions distantes. Les conversations et le contexte du code sont envoyés à l'API d'Anthropic pour le traitement. Voir [gestion des données](#) pour les détails sur la rétention des données, la confidentialité et la conformité.

Déploiement

Desktop peut être distribué via les outils de déploiement d'entreprise :

- **macOS** : distribuez via MDM comme Jamf ou Kandji en utilisant l'installateur `.dmg`
- **Windows** : déployez via le package MSIX ou l'installateur `.exe`. Voir [Déployer Claude Desktop pour Windows](#) pour les options de déploiement d'entreprise, y compris l'installation silencieuse

Pour la configuration réseau comme les paramètres de proxy, l'ajout à la liste blanche du pare-feu et les passerelles LLM, voir [configuration réseau](#).

Pour la référence complète de la configuration d'entreprise, voir le [guide de configuration d'entreprise](#).

Venant de la CLI ?

Si vous utilisez déjà la CLI Claude Code, Desktop exécute le même moteur sous-jacent avec une interface graphique. Vous pouvez exécuter les deux simultanément sur la même machine, même sur le même projet. Chacun maintient un historique de session séparé, mais ils partagent la configuration et la mémoire du projet via les fichiers CLAUDE.md.

Pour déplacer une session CLI dans Desktop, exécutez `/desktop` dans le terminal. Claude enregistre votre session et l'ouvre dans l'application de bureau, puis quitte la CLI. Cette commande est disponible uniquement sur macOS et Windows.

Tip:

Quand utiliser Desktop vs CLI : utilisez Desktop quand vous voulez l'examen visuel des différences, les pièces jointes de fichiers ou la gestion des sessions dans une barre latérale. Utilisez la CLI quand vous avez besoin de scripts, d'automatisation, de fournisseurs tiers ou préférez un flux de travail terminal.

Équivalents des drapeaux CLI

Ce tableau montre l'équivalent de l'application de bureau pour les drapeaux CLI courants. Les drapeaux non listés n'ont pas d'équivalent de bureau car ils sont conçus pour les scripts ou l'automatisation.

CLI	Équivalent de bureau
<code>--model sonnet</code>	liste déroulante de modèle à côté du bouton d'envoi, avant de démarrer une session
<code>--resume</code> , <code>--continue</code>	cliquez sur une session dans la barre latérale
<code>--permission-mode</code>	sélecteur de mode à côté du bouton d'envoi
<code>--dangerously-skip-permissions</code>	Mode Contourner les permissions. Activez dans Paramètres → Claude Code → « Autoriser le mode de contournement des permissions ». Les administrateurs d'entreprise peuvent désactiver ce paramètre.
<code>--add-dir</code>	ajouter plusieurs référentiels avec le bouton + dans les sessions distantes
<code>--allowedTools</code> , <code>--disallowedTools</code>	non disponible dans Desktop
<code>--verbose</code>	non disponible. Vérifiez les journaux système : Console.app sur macOS, Observateur d'événements → Journaux Windows → Application sur Windows
<code>--print</code> , <code>--output-format</code>	non disponible. Desktop est interactif uniquement.

CLI	Équivalent de bureau
Variable d'environnement <code>ANTHROPIC_MODEL</code>	liste déroulante de modèle à côté du bouton d'envoi
Variable d'environnement <code>MAX_THINKING_TOKENS</code>	définissez dans le profil shell ; s'applique aux sessions locales. Voir configuration de l'environnement .

Configuration partagée

Desktop et CLI lisent les mêmes fichiers de configuration, donc votre configuration se transfère :

- Les fichiers [CLAUDE.md](#) dans votre projet sont utilisés par les deux
- Les [MCP servers](#) configurés dans `~/.claude.json` ou `.mcp.json` fonctionnent dans les deux
- Les [Hooks](#) et [skills](#) définis dans les paramètres s'appliquent aux deux
- Les [Paramètres](#) dans `~/.claude.json` et `~/.claude/settings.json` sont partagés. Les règles de permission, les outils autorisés et d'autres paramètres dans `settings.json` s'appliquent aux sessions Desktop.
- **Modèles** : Sonnet, Opus et Haiku sont disponibles dans les deux. Dans Desktop, sélectionnez le modèle dans la liste déroulante à côté du bouton d'envoi avant de démarrer une session. Vous ne pouvez pas modifier le modèle pendant une session active.

Note:

MCP servers : application de chat de bureau vs Claude Code : les MCP servers configurés pour l'application de chat Claude Desktop dans `claude_desktop_config.json` sont séparés de Claude Code et n'apparaîtront pas dans l'onglet Code. Pour utiliser les MCP servers dans Claude Code, configurez-les dans `~/.claude.json` ou le fichier `.mcp.json` de votre projet. Voir [configuration MCP](#) pour les détails.

Comparaison des fonctionnalités

Ce tableau compare les capacités principales entre la CLI et Desktop. Pour une liste complète des drapeaux CLI, voir la [référence CLI](#).

Fonctionnalité	CLI	Desktop
Modes de permission	tous les modes y compris <code>dontAsk</code>	Demander les permissions, Accepter automatiquement les modifications, Plan mode et Contourner les permissions via Paramètres
<code>--dangerously-skip-permissions</code>	Drapeau CLI	Mode Contourner les permissions. Activez dans Paramètres → Claude Code → « Autoriser le mode de contournement des permissions »
Fournisseurs tiers	Bedrock, Vertex, Foundry	non disponible. Desktop se connecte directement à l'API d'Anthropic.
MCP servers	configurer dans les fichiers de paramètres	Interface utilisateur Connecteurs pour les sessions locales et SSH, ou fichiers de paramètres
Plugins	Commande <code>/plugin</code>	Interface utilisateur du gestionnaire de plugins
Fichiers @mention	basé sur le texte	avec autocomplétion
Pièces jointes de fichiers	non disponible	images, PDF
Isolation des sessions	Drapeau <code>--worktree</code>	worktrees automatiques
Sessions multiples	terminaux séparés	onglets de barre latérale
Tâches récurrentes	tâches cron, pipelines CI	tâches planifiées
Scripts et automatisation	<code>--print</code> , Agent SDK	non disponible

Ce qui n'est pas disponible dans Desktop

Les fonctionnalités suivantes sont disponibles uniquement dans la CLI ou l'extension VS Code :

- **Fournisseurs tiers** : Desktop se connecte directement à l'API d'Anthropic. Utilisez la [CLI](#) avec Bedrock, Vertex ou Foundry à la place.
- **Linux** : l'application de bureau est disponible uniquement sur macOS et Windows.
- **Suggestions de code en ligne** : Desktop ne fournit pas de suggestions de style auto-complétion. Il fonctionne via des messages conversationnels et des modifications de code explicites.
- **Équipes d'agents** : l'orchestration multi-agents est disponible via la [CLI](#) et [Agent SDK](#), pas dans Desktop.

Dépannage

Vérifier votre version

Pour voir quelle version de l'application de bureau vous exécutez :

- **macOS** : cliquez sur **Claude** dans la barre de menu, puis **À propos de Claude**
- **Windows** : cliquez sur **Aide**, puis **À propos**

Cliquez sur le numéro de version pour le copier dans votre presse-papiers.

Erreurs 403 ou d'authentification dans l'onglet Code

Si vous voyez **Error 403: Forbidden** ou d'autres défaillances d'authentification lors de l'utilisation de l'onglet Code :

1. Déconnectez-vous et reconnectez-vous à partir du menu de l'application. C'est le correctif le plus courant.
2. Vérifiez que vous avez un abonnement payant actif : Pro, Max, Teams ou Entreprise.
3. Si la CLI fonctionne mais Desktop ne fonctionne pas, quittez complètement l'application de bureau, pas seulement fermez la fenêtre, puis rouvrez et reconnectez-vous.
4. Vérifiez votre connexion Internet et vos paramètres de proxy.

Écran blanc ou bloqué au lancement

Si l'application s'ouvre mais affiche un écran blanc ou ne répond pas :

1. Redémarrez l'application.
2. Vérifiez les mises à jour en attente. L'application se met à jour automatiquement au lancement.

3. Sur Windows, vérifiez l'Observateur d'événements pour les journaux de crash sous **Journaux Windows → Application**.

« Impossible de charger la session »

Si vous voyez **Failed to load session**, le dossier sélectionné peut ne plus exister, un référentiel Git peut nécessiter Git LFS qui n'est pas installé, ou les permissions de fichier peuvent empêcher l'accès. Essayez de sélectionner un dossier différent ou redémarrez l'application.

Session ne trouvant pas les outils installés

Si Claude ne peut pas trouver des outils comme **npm**, **node** ou d'autres commandes CLI, vérifiez que les outils fonctionnent dans votre terminal régulier, vérifiez que votre profil shell configure correctement PATH et redémarrez l'application de bureau pour recharger les variables d'environnement.

Erreurs Git et Git LFS

Sur Windows, Git est requis pour que l'onglet Code démarre les sessions locales. Si vous voyez « Git is required », installez [Git pour Windows](#) et redémarrez l'application.

Si vous voyez « Git LFS is required by this repository but is not installed », installez Git LFS à partir de git-lfs.com, exécutez **git lfs install** et redémarrez l'application.

Les MCP servers ne fonctionnent pas sur Windows

Si les bascules MCP server ne répondent pas ou que les serveurs ne se connectent pas sur Windows, vérifiez que le serveur est correctement configuré dans vos paramètres, redémarrez l'application, vérifiez que le processus du serveur s'exécute dans le Gestionnaire des tâches et examinez les journaux du serveur pour les erreurs de connexion.

L'application ne veut pas quitter

- **macOS** : appuyez sur Cmd+Q. Si l'application ne répond pas, utilisez Forcer à quitter avec Cmd+Option+Esc, sélectionnez Claude et cliquez sur Forcer à quitter.
- **Windows** : utilisez le Gestionnaire des tâches avec Ctrl+Maj+Esc pour terminer le processus Claude.

Problèmes spécifiques à Windows

- **PATH non mis à jour après l'installation** : ouvrez une nouvelle fenêtre de terminal. Les mises à jour PATH s'appliquent uniquement aux nouvelles sessions de terminal.

- **Erreur d'installation simultanée** : si vous voyez une erreur concernant une autre installation en cours mais qu'il n'y en a pas, essayez d'exécuter l'installateur en tant qu'administrateur.
- **ARM64** : les appareils Windows ARM64 sont entièrement supportés.

Onglet Cowork indisponible sur les Macs Intel

L'onglet Cowork nécessite Apple Silicon (M1 ou ultérieur) sur macOS. Sur Windows, Cowork est disponible sur tout le matériel supporté. Les onglets Chat et Code fonctionnent normalement sur les Macs Intel.

« La branche n'existe pas encore » lors de l'ouverture dans la CLI

Les sessions distantes peuvent créer des branches qui n'existent pas sur votre machine locale. Cliquez sur le nom de la branche dans la barre d'outils de la session pour le copier, puis récupérez-le localement :

```
git fetch origin <branch-name>
git checkout <branch-name>
```

Toujours bloqué ?

- Recherchez ou signalez un bug sur [GitHub Issues](#)
- Visitez le [centre de support Claude](#)

Lors du signalement d'un bug, incluez la version de votre application de bureau, votre système d'exploitation, le message d'erreur exact et les journaux pertinents. Sur macOS, vérifiez Console.app. Sur Windows, vérifiez Observateur d'événements → Journaux Windows → Application.

Utiliser Claude Code avec Chrome (bêta)

Connectez Claude Code à votre navigateur Chrome pour tester des applications web, déboguer avec les journaux de console, automatiser le remplissage de formulaires et extraire des données des pages web.

Claude Code s'intègre à l'extension Claude in Chrome du navigateur pour vous offrir des capacités d'automatisation du navigateur depuis la CLI ou l'[extension VS Code](#). Créez votre code, puis testez et déboguez dans le navigateur sans changer de contexte.

Claude ouvre de nouveaux onglets pour les tâches du navigateur et partage l'état de connexion de votre navigateur, ce qui lui permet d'accéder à n'importe quel site auquel vous êtes déjà connecté. Les actions du navigateur s'exécutent en temps réel dans une fenêtre Chrome visible. Lorsque Claude rencontre une page de connexion ou un CAPTCHA, il s'arrête et vous demande de le gérer manuellement.

Note:

L'intégration Chrome est en bêta et fonctionne actuellement avec Google Chrome uniquement. Elle n'est pas encore prise en charge sur Brave, Arc ou d'autres navigateurs basés sur Chromium. WSL (Windows Subsystem for Linux) n'est pas non plus pris en charge.

Capacités

Avec Chrome connecté, vous pouvez enchaîner les actions du navigateur avec les tâches de codage dans un seul flux de travail :

- **Débogage en direct** : lisez les erreurs de console et l'état du DOM directement, puis corrigez le code qui les a causées
- **Vérification de la conception** : créez une interface utilisateur à partir d'une maquette Figma, puis ouvrez-la dans le navigateur pour vérifier qu'elle correspond
- **Test d'application web** : testez la validation des formulaires, vérifiez les régressions visuelles ou vérifiez les flux utilisateur
- **Applications web authentifiées** : interagissez avec Google Docs, Gmail, Notion ou n'importe quelle application à laquelle vous êtes connecté sans connecteurs API
- **Extraction de données** : extrayez des informations structurées des pages web et enregistrez-les localement

- **Automatisation des tâches** : automatisez les tâches répétitives du navigateur comme la saisie de données, le remplissage de formulaires ou les flux multi-sites
- **Enregistrement de session** : enregistrez les interactions du navigateur sous forme de GIF pour documenter ou partager ce qui s'est passé

Prérequis

Avant d'utiliser Claude Code avec Chrome, vous avez besoin de :

- Navigateur [Google Chrome](#)
- Extension [Claude in Chrome](#) version 1.0.36 ou supérieure
- [Claude Code](#) version 2.0.73 ou supérieure
- Un plan Anthropic direct (Pro, Max, Team ou Enterprise)

Note:

L'intégration Chrome n'est pas disponible via des fournisseurs tiers comme Amazon Bedrock, Google Cloud Vertex AI ou Microsoft Foundry. Si vous accédez à Claude exclusivement via un fournisseur tiers, vous avez besoin d'un compte [claude.ai](#) séparé pour utiliser cette fonctionnalité.

Démarrer dans la CLI

Step 1: Lancer Claude Code avec Chrome

Démarrez Claude Code avec le drapeau `--chrome` :

```
claude --chrome
```

Vous pouvez également activer Chrome au sein d'une session existante en exécutant `/chrome`.

Step 2: Demander à Claude d'utiliser le navigateur

Cet exemple accède à une page, interagit avec elle et rapporte ce qu'il trouve, le tout depuis votre terminal ou éditeur :

```
Go to code.claude.com/docs, click on the search box,  
type "hooks", and tell me what results appear
```


Exécutez `/chrome` à tout moment pour vérifier l'état de la connexion, gérer les autorisations ou reconnecter l'extension.

Pour VS Code, consultez [l'automatisation du navigateur dans VS Code](#).

Activer Chrome par défaut

Pour éviter de passer `--chrome` à chaque session, exécutez `/chrome` et sélectionnez « Enabled by default ».

Dans l'[extension VS Code](#), Chrome est disponible chaque fois que l'extension Chrome est installée. Aucun drapeau supplémentaire n'est nécessaire.

Note:

L'activation de Chrome par défaut dans la CLI augmente l'utilisation du contexte puisque les outils du navigateur sont toujours chargés. Si vous remarquez une augmentation de la consommation de contexte, désactivez ce paramètre et utilisez `--chrome` uniquement si nécessaire.

Gérer les autorisations du site

Les autorisations au niveau du site sont héritées de l'extension Chrome. Gérez les autorisations dans les paramètres de l'extension Chrome pour contrôler les sites que Claude peut parcourir, cliquer et taper.

Exemples de flux de travail

Ces exemples montrent les façons courantes de combiner les actions du navigateur avec les tâches de codage. Exécutez `/mcp` et sélectionnez `claude-in-chrome` pour voir la liste complète des outils de navigateur disponibles.

Tester une application web locale

Lors du développement d'une application web, demandez à Claude de vérifier que vos modifications fonctionnent correctement :

```
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 accède à votre serveur local, interagit avec le formulaire et rapporte ce qu'il observe.

Déboguer avec les journaux de console

Claude peut lire la sortie de la console pour aider à diagnostiquer les problèmes. Dites à Claude quels modèles rechercher plutôt que de demander toute la sortie de la console, car les journaux peuvent être verbeux :

```
Open the dashboard page and check the console for any errors when  
the page loads.
```

Claude lit les messages de la console et peut filtrer les modèles ou types d'erreurs spécifiques.

Automatiser le remplissage de formulaires

Accélérez les tâches répétitives de saisie de données :

```
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 lit votre fichier local, navigue dans l'interface web et saisit les données pour chaque enregistrement.

Rédiger du contenu dans Google Docs

Utilisez Claude pour écrire directement dans vos documents sans configuration d'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 ouvre le document, clique dans l'éditeur et tape le contenu. Cela fonctionne avec n'importe quelle application web à laquelle vous êtes connecté : Gmail, Notion, Sheets, et plus.

Extraire des données des pages web

Extrayez des informations structurées des sites 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 accède à la page, lit le contenu et compile les données dans un format structuré.

Exécuter des flux de travail multi-sites

Coordonnez les tâches sur plusieurs sites 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 travaille sur plusieurs onglets pour rassembler les informations et terminer le flux de travail.

Enregistrer un GIF de démonstration

Créez des enregistrements partageables des interactions du navigateur :

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude enregistre la séquence d'interaction et l'enregistre sous forme de fichier GIF.

Dépannage

Extension non détectée

Si Claude Code affiche « Chrome extension not detected » :

1. Vérifiez que l'extension Chrome est installée et activée dans `chrome://extensions`
2. Vérifiez que Claude Code est à jour en exécutant `claude --version`
3. Vérifiez que Chrome est en cours d'exécution
4. Exécutez `/chrome` et sélectionnez « Reconnect extension » pour rétablir la connexion
5. Si le problème persiste, redémarrez Claude Code et Chrome

La première fois que vous activez l'intégration Chrome, Claude Code installe un fichier de configuration d'hôte de messagerie native. Chrome lit ce fichier au démarrage, donc si l'extension n'est pas détectée à votre première tentative, redémarrez Chrome pour récupérer la nouvelle configuration.

Si la connexion échoue toujours, vérifiez que le fichier de configuration d'hôte existe à :

- **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** : vérifiez `HKCU\Software\Google\Chrome\NativeMessagingHosts\` dans le Registre Windows

Le navigateur ne répond pas

Si les commandes du navigateur de Claude cessent de fonctionner :

1. Vérifiez si une boîte de dialogue modale (alerte, confirmation, invite) bloque la page. Les boîtes de dialogue JavaScript bloquent les événements du navigateur et empêchent Claude de recevoir des commandes. Fermez la boîte de dialogue manuellement, puis demandez à Claude de continuer.
2. Demandez à Claude de créer un nouvel onglet et réessayez
3. Redémarrez l'extension Chrome en la désactivant et en la réactivant dans `chrome://extensions`

La connexion s'interrompt lors de longues sessions

Le service worker de l'extension Chrome peut devenir inactif lors de sessions prolongées, ce qui rompt la connexion. Si les outils du navigateur cessent de fonctionner après une période d'inactivité, exécutez `/chrome` et sélectionnez « Reconnect extension ».

Problèmes spécifiques à Windows

Sous Windows, vous pouvez rencontrer :

- **Conflits de tuyau nommé (EADDRINUSE)** : si un autre processus utilise le même tuyau nommé, redémarrez Claude Code. Fermez toute autre session Claude Code qui pourrait utiliser Chrome.
- **Erreurs d'hôte de messagerie native** : si l'hôte de messagerie native plante au démarrage, essayez de réinstaller Claude Code pour régénérer la configuration d'hôte.

Messages d'erreur courants

Ce sont les erreurs les plus fréquemment rencontrées et comment les résoudre :

Erreur	Cause	Solution
« Browser extension is not connected »	L'hôte de messagerie native ne peut pas atteindre l'extension	Redémarrez Chrome et Claude Code, puis exécutez <code>/chrome</code> pour reconnecter
« Extension not detected »	L'extension Chrome n'est pas installée ou est désactivée	Installez ou activez l'extension dans <code>chrome://extensions</code>
« No tab available »	Claude a tenté d'agir avant qu'un onglet soit prêt	Demandez à Claude de créer un nouvel onglet et réessayez
« Receiving end does not exist »	Le service worker de l'extension est devenu inactif	Exécutez <code>/chrome</code> et sélectionnez « Reconnect extension »

Voir aussi

- [Utiliser Claude Code dans VS Code](#) : automatisation du navigateur dans l'extension VS Code
- [Référence CLI](#) : drapeaux de ligne de commande incluant `--chrome`
- [Flux de travail courants](#) : plus de façons d'utiliser Claude Code
- [Données et confidentialité](#) : comment Claude Code gère vos données
- [Démarrer avec Claude in Chrome](#) : documentation complète pour l'extension Chrome, incluant les raccourcis, la planification et les autorisations

Claude Code dans Slack

Déléguez les tâches de codage directement depuis votre espace de travail Slack

Claude Code dans Slack apporte la puissance de Claude Code directement dans votre espace de travail Slack. Lorsque vous mentionnez `@Claude` avec une tâche de codage, Claude détecte automatiquement l'intention et crée une session Claude Code sur le web, vous permettant de déléguer le travail de développement sans quitter vos conversations d'équipe.

Cette intégration est construite sur l'application Claude existante pour Slack, mais ajoute un routage intelligent vers Claude Code sur le web pour les demandes liées au codage.

Cas d'usage

- **Enquête et correction de bugs** : Demandez à Claude d'enquêter et de corriger les bugs dès qu'ils sont signalés dans les canaux Slack.
- **Révisions de code rapides et modifications** : Faites en sorte que Claude implémente de petites fonctionnalités ou refactorise le code en fonction des commentaires de l'équipe.
- **Débogage collaboratif** : Lorsque les discussions d'équipe fournissent un contexte crucial (par exemple, les reproductions d'erreurs ou les rapports d'utilisateurs), Claude peut utiliser ces informations pour éclairer son approche du débogage.
- **Exécution de tâches parallèles** : Lancez des tâches de codage dans Slack tout en continuant d'autres travaux, en recevant des notifications à la fin.

Conditions préalables

Avant d'utiliser Claude Code dans Slack, assurez-vous d'avoir les éléments suivants :

Condition	Détails
Plan Claude	Pro, Max, Team ou Enterprise avec accès à Claude Code (sièges premium)
Claude Code sur le web	L'accès à Claude Code sur le web doit être activé
Compte GitHub	Connecté à Claude Code sur le web avec au moins un référentiel authentifié

Condition	Détails
Authentification Slack	Votre compte Slack lié à votre compte Claude via l'application Claude

Configuration de Claude Code dans Slack

Step 1: Installer l'application Claude dans Slack

Un administrateur de l'espace de travail doit installer l'application Claude à partir de la Slack App Marketplace. Visitez la [Slack App Marketplace](#) et cliquez sur « Ajouter à Slack » pour commencer le processus d'installation.

Step 2: Connectez votre compte Claude

Une fois l'application installée, authentifiez votre compte Claude individuel :

1. Ouvrez l'application Claude dans Slack en cliquant sur « Claude » dans votre section Applications
2. Accédez à l'onglet Accueil de l'application
3. Cliquez sur « Connecter » pour lier votre compte Slack à votre compte Claude
4. Complétez le flux d'authentification dans votre navigateur

Step 3: Configurez Claude Code sur le web

Assurez-vous que votre Claude Code sur le web est correctement configuré :

- Visitez [claude.ai/code](#) et connectez-vous avec le même compte que celui que vous avez connecté à Slack
- Connectez votre compte GitHub s'il n'est pas déjà connecté
- Authentifiez au moins un référentiel avec lequel vous souhaitez que Claude travaille

Step 4: Choisissez votre mode de routage

Après avoir connecté vos comptes, configurez la façon dont Claude gère vos messages dans Slack. Accédez à l'Accueil de l'application Claude dans Slack pour trouver le paramètre **Mode de routage**.

Mode	Comportement
Code uni-que-ment	Claude achemine toutes les @mentions vers les sessions Claude Code. Idéal pour les équipes utilisant Claude dans Slack exclusivement pour les tâches de développement.

Mode	Comportement
Code + Chat	Claude analyse chaque message et achemine intelligemment entre Claude Code (pour les tâches de codage) et Claude Chat (pour la rédaction, l'analyse et les questions générales). Idéal pour les équipes qui souhaitent un point d'entrée @Claude unique pour tous les types de travail.

Note:

En mode Code + Chat, si Claude achemine un message vers Chat mais que vous vouliez une session de codage, vous pouvez cliquer sur « Réessayer en tant que Code » pour créer une session Claude Code à la place. De même, s'il est acheminé vers Code mais que vous vouliez une session Chat, vous pouvez choisir cette option dans ce fil.

Fonctionnement

Détection automatique

Lorsque vous mentionnez @Claude dans un canal ou un fil Slack, Claude analyse automatiquement votre message pour déterminer s'il s'agit d'une tâche de codage. Si Claude détecte une intention de codage, il acheminera votre demande vers Claude Code sur le web au lieu de répondre en tant qu'assistant de chat ordinaire.

Vous pouvez également dire explicitement à Claude de traiter une demande en tant que tâche de codage, même s'il ne la détecte pas automatiquement.

Note:

Claude Code dans Slack ne fonctionne que dans les canaux (publics ou privés). Il ne fonctionne pas dans les messages directs (DM).

Collecte de contexte

À partir des fils : Lorsque vous @mentionnez Claude dans un fil, il recueille le contexte de tous les messages de ce fil pour comprendre la conversation complète.

À partir des canaux : Lorsqu'il est mentionné directement dans un canal, Claude examine les messages récents du canal pour un contexte pertinent.

Ce contexte aide Claude à comprendre le problème, à sélectionner le référentiel approprié et à éclairer son approche de la tâche.

Warning:

Lorsque @Claude est invoqué dans Slack, Claude a accès au contexte de la conversation pour mieux comprendre votre demande. Claude peut suivre les directives d'autres messages dans le contexte, donc les utilisateurs doivent s'assurer d'utiliser Claude uniquement dans des conversations Slack de confiance.

Flux de session

1. **Initiation** : Vous @mentionnez Claude avec une demande de codage
2. **Détection** : Claude analyse votre message et détecte l'intention de codage
3. **Création de session** : Une nouvelle session Claude Code est créée sur claude.ai/code
4. **Mises à jour de progression** : Claude publie des mises à jour de statut dans votre fil Slack au fur et à mesure de la progression du travail
5. **Achèvement** : Une fois terminé, Claude vous @mentionne avec un résumé et des boutons d'action
6. **Révision** : Cliquez sur « Afficher la session » pour voir la transcription complète, ou « Créer une PR » pour ouvrir une demande de tirage

Éléments de l'interface utilisateur

Accueil de l'application

L'onglet Accueil de l'application affiche votre statut de connexion et vous permet de connecter ou de déconnecter votre compte Claude de Slack.

Actions de message

- **Afficher la session** : Ouvre la session Claude Code complète dans votre navigateur où vous pouvez voir tout le travail effectué, continuer la session ou faire des demandes supplémentaires.
- **Créer une PR** : Crée une demande de tirage directement à partir des modifications de la session.
- **Réessayer en tant que Code** : Si Claude a initialement répondu en tant qu'assistant de chat mais que vous vouliez une session de codage, cliquez sur ce bouton pour réessayer la demande en tant que tâche Claude Code.
- **Changer de référentiel** : Vous permet de sélectionner un référentiel différent si Claude a mal choisi.

Sélection du référentiel

Claude sélectionne automatiquement un référentiel en fonction du contexte de votre conversation Slack. Si plusieurs référentiels pourraient s'appliquer, Claude peut afficher une liste déroulante vous permettant de choisir le bon.

Accès et permissions

Accès au niveau utilisateur

Type d'accès	Condition
Sessions Claude Code	Chaque utilisateur exécute les sessions sous son propre compte Claude
Utilisation et limites de débit	Les sessions comptent par rapport aux limites du plan de l'utilisateur individuel
Accès au référentiel	Les utilisateurs ne peuvent accéder qu'aux référentiels qu'ils ont personnellement connectés
Historique des sessions	Les sessions apparaissent dans votre historique Claude Code sur claude.ai/code

Permissions d'administrateur de l'espace de travail

Les administrateurs de l'espace de travail Slack contrôlent si l'application Claude peut être installée dans l'espace de travail. Les utilisateurs individuels s'authentifient ensuite avec leurs propres comptes Claude pour utiliser l'intégration.

Ce qui est accessible où

Dans Slack : Vous verrez les mises à jour de statut, les résumés d'achèvement et les boutons d'action. La transcription complète est préservée et toujours accessible.

Sur le web : La session Claude Code complète avec l'historique complet de la conversation, tous les changements de code, les opérations de fichiers et la possibilité de continuer la session ou de créer des demandes de tirage.

Meilleures pratiques

Rédaction de demandes efficaces

- **Soyez précis :** Incluez les noms de fichiers, les noms de fonctions ou les messages d'erreur si pertinent.

- **Fournissez du contexte** : Mentionnez le référentiel ou le projet s'il n'est pas clair à partir de la conversation.
- **Définissez le succès** : Expliquez à quoi ressemble « terminé » — Claude devrait-il écrire des tests ? Mettre à jour la documentation ? Créer une PR ?
- **Utilisez les fils** : Répondez dans les fils lors de la discussion de bugs ou de fonctionnalités afin que Claude puisse recueillir le contexte complet.

Quand utiliser Slack par rapport au web

Utilisez Slack quand : Le contexte existe déjà dans une discussion Slack, vous souhaitez lancer une tâche de manière asynchrone, ou vous collaborez avec des coéquipiers qui ont besoin de visibilité.

Utilisez le web directement quand : Vous devez télécharger des fichiers, souhaitez une interaction en temps réel pendant le développement, ou travaillez sur des tâches plus longues et plus complexes.

Dépannage

Les sessions ne démarrent pas

1. Vérifiez que votre compte Claude est connecté dans l'Accueil de l'application Claude
2. Vérifiez que vous avez accès à Claude Code sur le web activé
3. Assurez-vous que vous avez au moins un référentiel GitHub connecté à Claude Code

Le référentiel ne s'affiche pas

1. Connectez le référentiel dans Claude Code sur le web à claude.ai/code
2. Vérifiez vos permissions GitHub pour ce référentiel
3. Essayez de déconnecter et de reconnecter votre compte GitHub

Mauvais référentiel sélectionné

1. Cliquez sur le bouton « Changer de référentiel » pour sélectionner un référentiel différent
2. Incluez le nom du référentiel dans votre demande pour une sélection plus précise

Erreurs d'authentification

1. Déconnectez et reconnectez votre compte Claude dans l'Accueil de l'application
2. Assurez-vous que vous êtes connecté au bon compte Claude dans votre navigateur
3. Vérifiez que votre plan Claude inclut l'accès à Claude Code

Expiration de la session

1. Les sessions restent accessibles dans votre historique Claude Code sur le web
2. Vous pouvez continuer ou référencer les sessions passées à partir de claude.ai/code

Limitations actuelles

- **GitHub uniquement** : Prend actuellement en charge les référentiels sur GitHub.
- **Une PR à la fois** : Chaque session peut créer une demande de tirage.
- **Les limites de débit s'appliquent** : Les sessions utilisent les limites de débit du plan Claude individuel.
- **Accès web requis** : Les utilisateurs doivent avoir accès à Claude Code sur le web ; ceux qui ne l'ont pas recevront uniquement des réponses de chat Claude standard.

Ressources connexes

- [Claude Code sur le web](#) En savoir plus sur Claude Code sur le web
- [Claude pour Slack](#) Documentation générale de Claude pour Slack
- [Slack App Marketplace](#) Installez l'application Claude à partir de la Slack Marketplace
- [Centre d'aide Claude](#) Obtenir un support supplémentaire

Claude Code sur le web

Exécutez les tâches Claude Code de manière asynchrone sur une infrastructure cloud sécurisée

Note:

Claude Code sur le web est actuellement en aperçu de recherche.

Qu'est-ce que Claude Code sur le web ?

Claude Code sur le web permet aux développeurs de lancer Claude Code depuis l'application Claude. C'est parfait pour :

- **Répondre à des questions** : Posez des questions sur l'architecture du code et la façon dont les fonctionnalités sont implémentées
- **Corrections de bugs et tâches de routine** : Tâches bien définies qui ne nécessitent pas de direction fréquente
- **Travail en parallèle** : Abordez plusieurs corrections de bugs en parallèle
- **Référentiels non sur votre machine locale** : Travaillez sur du code que vous n'avez pas extrait localement
- **Modifications du backend** : Où Claude Code peut écrire des tests puis écrire du code pour passer ces tests

Claude Code est également disponible sur l'application Claude pour [iOS](#) et [Android](#) pour lancer des tâches en déplacement et surveiller le travail en cours.

Vous pouvez [lancer de nouvelles tâches sur le web depuis votre terminal](#) avec `--remote`, ou [téléporter les sessions web vers votre terminal](#) pour continuer localement. Pour utiliser l'interface web tout en exécutant Claude Code sur votre propre machine au lieu de l'infrastructure cloud, consultez [Contrôle à distance](#).

Qui peut utiliser Claude Code sur le web ?

Claude Code sur le web est disponible en aperçu de recherche pour :

- **Les utilisateurs Pro**
- **Les utilisateurs Max**

- **Les utilisateurs Team**
- **Les utilisateurs Enterprise** avec des sièges premium ou des sièges Chat + Claude Code

Démarrage

1. Visitez claude.ai/code
2. Connectez votre compte GitHub
3. Installez l'application Claude GitHub dans vos référentiels
4. Sélectionnez votre environnement par défaut
5. Soumettez votre tâche de codage
6. Examinez les modifications en vue diff, itérez avec des commentaires, puis créez une demande de tirage

Comment ça marche

Lorsque vous démarrez une tâche sur Claude Code sur le web :

1. **Clonage du référentiel** : Votre référentiel est cloné sur une machine virtuelle gérée par Anthropic
2. **Configuration de l'environnement** : Claude prépare un environnement cloud sécurisé avec votre code, puis exécute votre [script de configuration](#) s'il est configuré
3. **Configuration du réseau** : L'accès à Internet est configuré en fonction de vos paramètres
4. **Exécution de la tâche** : Claude analyse le code, apporte des modifications, exécute des tests et vérifie son travail
5. **Achèvement** : Vous êtes notifié lorsque c'est terminé et vous pouvez créer une PR avec les modifications
6. **Résultats** : Les modifications sont poussées vers une branche, prêtes pour la création d'une demande de tirage

Examinez les modifications avec la vue diff

La vue diff vous permet de voir exactement ce que Claude a modifié avant de créer une demande de tirage. Au lieu de cliquer sur « Créer une PR » pour examiner les modifications dans GitHub, affichez le diff directement dans l'application et itérez avec Claude jusqu'à ce que les modifications soient prêtes.

Lorsque Claude apporte des modifications aux fichiers, un indicateur de statistiques diff apparaît indiquant le nombre de lignes ajoutées et supprimées (par exemple, `+12 -1`). Sélectionnez cet indicateur pour ouvrir la visionneuse diff, qui affiche une liste de fichiers à gauche et les modifications pour chaque fichier à droite.

À partir de la vue diff, vous pouvez :

- Examiner les modifications fichier par fichier
- Commenter des modifications spécifiques pour demander des modifications
- Continuer à itérer avec Claude en fonction de ce que vous voyez

Cela vous permet d'affiner les modifications à travers plusieurs cycles de rétroaction sans créer de PR de brouillon ni basculer vers GitHub.

Déplacer les tâches entre le web et le terminal

Vous pouvez démarrer de nouvelles tâches sur le web depuis votre terminal, ou extraire les sessions web dans votre terminal pour continuer localement. Les sessions web persistent même si vous fermez votre ordinateur portable, et vous pouvez les surveiller de n'importe où, y compris depuis l'application mobile Claude.

Note:

Le transfert de session est unidirectionnel : vous pouvez extraire les sessions web dans votre terminal, mais vous ne pouvez pas pousser une session de terminal existante vers le web. L'indicateur `--remote` crée une *nouvelle* session web pour votre référentiel actuel.

Du terminal au web

Démarrez une session web à partir de la ligne de commande avec l'indicateur `--remote` :

```
claude --remote "Fix the authentication bug in src/auth/login.ts"
```

Cela crée une nouvelle session web sur `claude.ai`. La tâche s'exécute dans le cloud tandis que vous continuez à travailler localement. Utilisez `/tasks` pour vérifier la progression, ou ouvrez la session sur `claude.ai` ou l'application mobile Claude pour interagir directement. De là, vous pouvez diriger Claude, fournir des commentaires ou répondre à des questions comme dans n'importe quelle autre conversation.

Conseils pour les tâches distantes

Planifiez localement, exécutez à distance : Pour les tâches complexes, démarrez Claude en mode plan pour collaborer sur l'approche, puis envoyez le travail sur le web :

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

En mode plan, Claude ne peut que lire les fichiers et explorer la base de code. Une fois que vous êtes satisfait du plan, démarrez une session distante pour une exécution autonome :

```
claude --remote "Execute the migration plan in docs/migration-plan.md"
```

Ce modèle vous donne le contrôle sur la stratégie tout en permettant à Claude d'exécuter de manière autonome dans le cloud.

Exécutez les tâches en parallèle : Chaque commande `--remote` crée sa propre session web qui s'exécute indépendamment. Vous pouvez lancer plusieurs tâches et elles s'exécuteront toutes simultanément dans des sessions séparées :

```
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"
```

Surveillez toutes les sessions avec `/tasks`. Lorsqu'une session se termine, vous pouvez créer une PR à partir de l'interface web ou [téléporter](#) la session vers votre terminal pour continuer à travailler.

Du web au terminal

Il existe plusieurs façons d'extraire une session web dans votre terminal :

- **Utilisation de `/teleport`** : Depuis Claude Code, exécutez `/teleport` (ou `/tp`) pour voir un sélecteur interactif de vos sessions web. Si vous avez des modifications non validées, vous serez invité à les ranger d'abord.
- **Utilisation de `--teleport`** : À partir de la ligne de commande, exécutez `claude --teleport` pour un sélecteur de session interactif, ou `claude --teleport <session-id>` pour reprendre une session spécifique directement.
- **À partir de `/tasks`** : Exécutez `/tasks` pour voir vos sessions en arrière-plan, puis appuyez sur `t` pour vous téléporter dans l'une d'elles

- **À partir de l'interface web** : Cliquez sur « Ouvrir dans CLI » pour copier une commande que vous pouvez coller dans votre terminal

Lorsque vous téléportez une session, Claude vérifie que vous êtes dans le bon référentiel, récupère et extrait la branche de la session distante, et charge l'historique complet de la conversation dans votre terminal.

Exigences pour la téléportation

La téléportation vérifie ces exigences avant de reprendre une session. Si une exigence n'est pas satisfaite, vous verrez une erreur ou vous serez invité à résoudre le problème.

Exi- gence	Détails
État git propre	Votre répertoire de travail ne doit avoir aucune modification non validée. La téléportation vous invite à ranger les modifications si nécessaire.
Réfé- rentiel correct	Vous devez exécuter <code>--teleport</code> à partir d'une extraction du même référentiel, pas d'une fourche.
Branche dispo- nible	La branche de la session web doit avoir été poussée vers le serveur distant. La téléportation la récupère et l'extrait automatiquement.
Même compte	Vous devez être authentifié au même compte Claude.ai utilisé dans la session web.

Partage de sessions

Pour partager une session, basculez sa visibilité selon les types de compte ci-dessous. Après cela, partagez le lien de session tel quel. Les destinataires qui ouvrent votre session partagée verront l'état le plus récent de la session au chargement, mais la page du destinataire ne se mettra pas à jour en temps réel.

Partage à partir d'un compte Enterprise ou Teams

Pour les comptes Enterprise et Teams, les deux options de visibilité sont **Privé** et **Team**. La visibilité Team rend la session visible aux autres membres de votre organisation Claude.ai. La vérification de l'accès au référentiel est activée par défaut, en fonction du compte GitHub connecté au compte du destinataire. Le nom d'affichage de votre compte est visible à tous les destinataires ayant accès. Les sessions [Claude dans Slack](#) sont automatiquement partagées avec la visibilité Team.

Partage à partir d'un compte Max ou Pro

Pour les comptes Max et Pro, les deux options de visibilité sont **Privé** et **Public**. La visibilité publique rend la session visible à tout utilisateur connecté à claude.ai.

Vérifiez votre session pour le contenu sensible avant de la partager. Les sessions peuvent contenir du code et des identifiants provenant de référentiels GitHub privés. La vérification de l'accès au référentiel n'est pas activée par défaut.

Activez la vérification de l'accès au référentiel et/ou retenez votre nom de vos sessions partagées en accédant à Paramètres > Claude Code > Paramètres de partage.

Gestion des sessions

Archivage des sessions

Vous pouvez archiver les sessions pour garder votre liste de sessions organisée. Les sessions archivées sont masquées de la liste de sessions par défaut mais peuvent être affichées en filtrant les sessions archivées.

Pour archiver une session, survolez la session dans la barre latérale et cliquez sur l'icône d'archivage.

Suppression des sessions

La suppression d'une session supprime définitivement la session et ses données. Cette action ne peut pas être annulée. Vous pouvez supprimer une session de deux façons :

- **À partir de la barre latérale** : Filtrez les sessions archivées, puis survolez la session que vous souhaitez supprimer et cliquez sur l'icône de suppression
- **À partir du menu de session** : Ouvrez une session, cliquez sur la liste déroulante à côté du titre de la session, et sélectionnez **Supprimer**

Vous serez invité à confirmer avant la suppression d'une session.

Environnement cloud

Image par défaut

Nous construisons et maintenons une image universelle avec des chaînes d'outils courantes et des écosystèmes de langage pré-installés. Cette image inclut :

- Langages de programmation et runtimes populaires
- Outils de construction courants et gestionnaires de paquets
- Frameworks de test et linters

Vérification des outils disponibles

Pour voir ce qui est pré-installé dans votre environnement, demandez à Claude Code d'exécuter :

```
check-tools
```

Cette commande affiche :

- Langages de programmation et leurs versions
- Gestionnaires de paquets disponibles
- Outils de développement installés

Configurations spécifiques au langage

L'image universelle inclut des environnements pré-configurés pour :

- **Python** : Python 3.x avec pip, poetry et des bibliothèques scientifiques courantes
- **Node.js** : Dernières versions LTS avec npm, yarn, pnpm et bun
- **Ruby** : Versions 3.1.6, 3.2.6, 3.3.6 (par défaut : 3.3.6) avec gem, bundler et rbenv pour la gestion des versions
- **PHP** : Version 8.4.14
- **Java** : OpenJDK avec Maven et Gradle
- **Go** : Dernière version stable avec support des modules
- **Rust** : Chaîne d'outils Rust avec cargo
- **C++** : Compilateurs GCC et Clang

Bases de données

L'image universelle inclut les bases de données suivantes :

- **PostgreSQL** : Version 16
- **Redis** : Version 7.0

Configuration de l'environnement

Lorsque vous démarrez une session dans Claude Code sur le web, voici ce qui se passe en coulisse :

1. **Préparation de l'environnement** : Nous clonons votre référentiel et exécutons tout [script de configuration](#) configuré. Le référentiel sera cloné avec la branche par

défaut de votre référentiel GitHub. Si vous souhaitez extraire une branche spécifique, vous pouvez la spécifier dans l'invite.

2. **Configuration du réseau** : Nous configurons l'accès à Internet pour l'agent. L'accès à Internet est limité par défaut, mais vous pouvez configurer l'environnement pour n'avoir aucun accès à Internet ou un accès Internet complet en fonction de vos besoins.
3. **Exécution de Claude Code** : Claude Code s'exécute pour accomplir votre tâche, en écrivant du code, en exécutant des tests et en vérifiant son travail. Vous pouvez guider et diriger Claude tout au long de la session via l'interface web. Claude respecte le contexte que vous avez défini dans votre `CLAUDE.md`.
4. **Résultat** : Lorsque Claude termine son travail, il poussera la branche vers le serveur distant. Vous pourrez créer une PR pour la branche.

Note:

Claude opère entièrement via le terminal et les outils CLI disponibles dans l'environnement. Il utilise les outils pré-installés dans l'image universelle et tous les outils supplémentaires que vous installez via des hooks ou la gestion des dépendances.

Pour ajouter un nouvel environnement : Sélectionnez l'environnement actuel pour ouvrir le sélecteur d'environnement, puis sélectionnez « Ajouter un environnement ». Cela ouvrira une boîte de dialogue où vous pouvez spécifier le nom de l'environnement, le niveau d'accès réseau, les variables d'environnement et un [script de configuration](#).

Pour mettre à jour un environnement existant : Sélectionnez l'environnement actuel, à droite du nom de l'environnement, et sélectionnez le bouton des paramètres. Cela ouvrira une boîte de dialogue où vous pouvez mettre à jour le nom de l'environnement, l'accès réseau, les variables d'environnement et le script de configuration.

Pour sélectionner votre environnement par défaut à partir du terminal : Si vous avez plusieurs environnements configurés, exécutez `/remote-env` pour choisir celui à utiliser lors du démarrage de sessions web à partir de votre terminal avec `--remote`. Avec un seul environnement, cette commande affiche votre configuration actuelle.

Note:

Les variables d'environnement doivent être spécifiées sous forme de paires clé-valeur, au [format](#) `.env`. Par exemple :

```
API_KEY=your_api_key
DEBUG=true
```

Scripts de configuration

Un script de configuration est un script Bash qui s'exécute au démarrage d'une nouvelle session cloud, avant le lancement de Claude Code. Utilisez les scripts de configuration pour installer des dépendances, configurer des outils ou préparer tout ce dont l'environnement cloud a besoin et qui ne figure pas dans [l'image par défaut](#).

Les scripts s'exécutent en tant que root sur Ubuntu 24.04, donc `apt install` et la plupart des gestionnaires de paquets de langage fonctionnent.

Tip:

Pour vérifier ce qui est déjà installé avant de l'ajouter à votre script, demandez à Claude d'exécuter `check-tools` dans une session cloud.

Pour ajouter un script de configuration, ouvrez la boîte de dialogue des paramètres d'environnement et entrez votre script dans le champ **Script de configuration**.

Cet exemple installe le CLI `gh`, qui ne figure pas dans l'image par défaut :

```
#!/bin/bash
apt update && apt install -y gh
```

Les scripts de configuration s'exécutent uniquement lors de la création d'une nouvelle session. Ils sont ignorés lors de la reprise d'une session existante.

Si le script se termine avec un code non nul, la session ne démarre pas. Ajoutez `|| true` aux commandes non critiques pour éviter de bloquer la session sur une installation instable.

Note:

Les scripts de configuration qui installent des paquets ont besoin d'un accès réseau pour atteindre les registres. L'accès réseau par défaut permet les connexions aux [registres de paquets courants](#) y compris npm, PyPI, RubyGems et crates.io. Les scripts échoueront à installer les paquets si votre environnement a l'accès réseau désactivé.

Scripts de configuration vs. hooks SessionStart

Utilisez un script de configuration pour installer les choses dont le cloud a besoin mais que votre ordinateur portable a déjà, comme un runtime de langage ou un outil CLI. Utilisez un hook [SessionStart](#) pour la configuration du projet qui devrait s'exécuter partout, cloud et local, comme `npm install`.

Les deux s'exécutent au démarrage d'une session, mais ils appartiennent à des endroits différents :

	Scripts de configuration	Hooks SessionStart
Attaché à	L'environnement cloud	Votre référentiel
Configuré dans	Interface utilisateur de l'environnement cloud	<code>.claude/settings.json</code> dans votre référentiel
S'exécute	Avant le lancement de Claude Code, sur les nouvelles sessions uniquement	Après le lancement de Claude Code, sur chaque session y compris les sessions reprises
Portée	Environnements cloud uniquement	Local et cloud

Les hooks SessionStart peuvent également être définis dans votre `~/.claude/settings.json` au niveau de l'utilisateur localement, mais les paramètres au niveau de l'utilisateur ne sont pas transférés aux sessions cloud. Dans le cloud, seuls les hooks validés dans le référentiel s'exécutent.

Gestion des dépendances

Les images d'environnement personnalisées et les snapshots ne sont pas encore pris en charge. Utilisez les [scripts de configuration](#) pour installer les paquets au démarrage d'une session, ou les [hooks SessionStart](#) pour l'installation de dépendances qui devrait également s'exécuter dans les environnements locaux. Les hooks SessionStart ont des [limitations connues](#).

Pour configurer l'installation automatique des dépendances avec un script de configuration, ouvrez vos paramètres d'environnement et ajoutez un script :

```
#!/bin/bash
npm install
pip install -r requirements.txt
```

Vous pouvez également utiliser les hooks `SessionStart` dans le fichier `.claude/settings.json` de votre référentiel pour l'installation de dépendances qui devrait également s'exécuter dans les environnements locaux :

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "startup",
        "hooks": [
          {
            "type": "command",
            "command": "\\\"$CLAUDE_PROJECT_DIR\\\"/scripts/install_pkgs.sh"
          }
        ]
      }
    ]
  }
}
```

Créez le script correspondant à `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
```

Rendez-le exécutable : `chmod +x scripts/install_pkgs.sh`

Persister les variables d'environnement

Les hooks `SessionStart` peuvent persister les variables d'environnement pour les commandes Bash suivantes en écrivant dans le fichier spécifié dans la variable d'environnement `CLAUDE_ENV_FILE`. Pour plus de détails, consultez les [hooks SessionStart](#) dans la référence des hooks.

Limitations de la gestion des dépendances

- **Les hooks se déclenchent pour toutes les sessions** : Les hooks `SessionStart` s'exécutent dans les environnements locaux et distants. Il n'y a pas de configuration de hook pour limiter un hook aux sessions distantes uniquement. Pour ignorer l'exécution locale, vérifiez la variable d'environnement `CLAUDE_CODE_REMOTE` dans votre script comme indiqué ci-dessus.
- **Nécessite un accès réseau** : Les commandes d'installation ont besoin d'un accès réseau pour atteindre les registres de paquets. Si votre environnement est configuré avec un accès « Pas d'Internet », ces hooks échoueront. Utilisez l'accès réseau « Limité » (par défaut) ou « Complet ». La [liste d'autorisation par défaut](#) inclut les registres courants comme npm, PyPI, RubyGems et crates.io.
- **Compatibilité du proxy** : Tout le trafic sortant dans les environnements distants passe par un [proxy de sécurité](#). Certains gestionnaires de paquets ne fonctionnent pas correctement avec ce proxy. Bun est un exemple connu.
- **S'exécute à chaque démarrage de session** : Les hooks s'exécutent chaque fois qu'une session démarre ou reprend, ajoutant une latence de démarrage. Gardez les scripts d'installation rapides en vérifiant si les dépendances sont déjà présentes avant de les réinstaller.

Accès réseau et sécurité

Politique réseau

Proxy GitHub

Pour la sécurité, toutes les opérations GitHub passent par un service proxy dédié qui gère de manière transparente toutes les interactions git. À l'intérieur du sandbox, le client git s'authentifie à l'aide d'une identité personnalisée limitée. Ce proxy :

- Gère l'authentification GitHub de manière sécurisée - le client git utilise une identité limitée à l'intérieur du sandbox, que le proxy vérifie et traduit en votre jeton d'authentification GitHub réel

- Restreint les opérations de poussée git à la branche de travail actuelle pour la sécurité
- Permet le clonage, la récupération et les opérations PR transparentes tout en maintenant les limites de sécurité

Proxy de sécurité

Les environnements s'exécutent derrière un proxy réseau HTTP/HTTPS pour la sécurité et la prévention des abus. Tout le trafic Internet sortant passe par ce proxy, qui fournit :

- Protection contre les demandes malveillantes
- Limitation de débit et prévention des abus
- Filtrage de contenu pour une sécurité renforcée

Niveaux d'accès

Par défaut, l'accès réseau est limité aux [domaines autorisés](#).

Vous pouvez configurer un accès réseau personnalisé, y compris la désactivation de l'accès réseau.

Domaines autorisés par défaut

Lors de l'utilisation de l'accès réseau « Limité », les domaines suivants sont autorisés par défaut :

Services Anthropic

- api.anthropic.com
- statsig.anthropic.com
- platform.claude.com
- code.claude.com
- claude.ai

Contrôle de version

- 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

Registres de conteneurs

- 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

Plateformes 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

Gestionnaires de paquets - JavaScript/Node

- registry.npmjs.org
- www.npmjs.com
- www.npmjs.org
- npmjs.com
- npmjs.org
- yarnpkg.com
- registry.yarnpkg.com

Gestionnaires de paquets - Python

- pypi.org
- www.pypi.org
- files.pythonhosted.org

- pythonhosted.org
- test.pypi.org
- pypi.python.org
- pypa.io
- www.pypa.io

Gestionnaires de paquets - 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

Gestionnaires de paquets - Rust

- crates.io
- www.crates.io
- index.crates.io
- static.crates.io
- rustup.rs
- static.rust-lang.org
- www.rust-lang.org

Gestionnaires de paquets - Go

- proxy.golang.org
- sum.golang.org
- index.golang.org
- golang.org

- www.golang.org
- goproxy.io
- pkg.go.dev

Gestionnaires de paquets - 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

Gestionnaires de paquets - Autres langages

- 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

Distributions Linux

- archive.ubuntu.com
- security.ubuntu.com
- ubuntu.com
- www.ubuntu.com
- *.ubuntu.com
- ppa.launchpad.net
- launchpad.net
- www.launchpad.net

Outils de développement et plateformes

- 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

Services cloud et surveillance

- statsig.com
- www.statsig.com
- api.statsig.com
- sentry.io
- *.sentry.io
- http-intake.logs.datadoghq.com
- *.datadoghq.com
- *.datadoghq.eu

Livraison de contenu et miroirs

- sourceforge.net
- *.sourceforge.net
- packagecloud.io
- *.packagecloud.io

Schéma et configuration

- json-schema.org
- www.json-schema.org
- json.schemastore.org
- www.schemastore.org

Model Context Protocol

- *.modelcontextprotocol.io

Note:

Les domaines marqués avec * indiquent une correspondance de sous-domaine générique. Par exemple, *.gcr.io permet l'accès à n'importe quel sous-domaine de gcr.io.

Meilleures pratiques de sécurité pour l'accès réseau personnalisé

1. **Principe du moindre privilège** : N'activez que l'accès réseau minimum requis
2. **Auditez régulièrement** : Examinez les domaines autorisés périodiquement
3. **Utilisez HTTPS** : Préférez toujours les points de terminaison HTTPS à HTTP

Sécurité et isolation

Claude Code sur le web fournit des garanties de sécurité fortes :

- **Machines virtuelles isolées** : Chaque session s'exécute dans une VM isolée gérée par Anthropic
- **Contrôles d'accès réseau** : L'accès réseau est limité par défaut et peut être désactivé

Note:

Lors de l'exécution avec l'accès réseau désactivé, Claude Code est autorisé à communiquer avec l'API Anthropic, ce qui peut toujours permettre aux données de quitter la VM Claude Code isolée.

- **Protection des identifiants** : Les identifiants sensibles (tels que les identifiants git ou les clés de signature) ne sont jamais à l'intérieur du sandbox avec Claude Code. L'authentification est gérée via un proxy sécurisé utilisant des identifiants limités
- **Analyse sécurisée** : Le code est analysé et modifié dans des VM isolées avant la création de PR

Tarification et limites de débit

Claude Code sur le web partage les limites de débit avec tous les autres usages de Claude et Claude Code au sein de votre compte. L'exécution de plusieurs tâches en parallèle consommera proportionnellement plus de limites de débit.

Limitations

- **Authentification du référentiel** : Vous ne pouvez déplacer les sessions du web vers le local que lorsque vous êtes authentifié au même compte
- **Restrictions de plateforme** : Claude Code sur le web ne fonctionne qu'avec le code hébergé sur GitHub. Les référentiels non-GitHub comme GitLab ne peuvent pas être utilisés avec les sessions cloud

Meilleures pratiques

1. **Automatisez la configuration de l'environnement** : Utilisez les [scripts de configuration](#) pour installer les dépendances et configurer les outils avant le lancement de Claude Code. Pour les scénarios plus avancés, configurez les [hooks SessionStart](#).
2. **Documentez les exigences** : Spécifiez clairement les dépendances et les commandes dans votre fichier `CLAUDE.md`. Si vous avez un fichier `AGENTS.md`, vous pouvez le sourcer dans votre `CLAUDE.md` en utilisant `@AGENTS.md` pour maintenir une source unique de vérité.

Ressources connexes

- [Configuration des hooks](#)
- [Référence des paramètres](#)
- [Sécurité](#)
- [Utilisation des données](#)

Part 9: Security & Privacy

Sécurité

Découvrez les protections de sécurité de Claude Code et les meilleures pratiques pour une utilisation sûre.

Comment nous abordons la sécurité

Fondation de sécurité

La sécurité de votre code est primordiale. Claude Code est construit avec la sécurité au cœur, développé selon le programme de sécurité complet d'Anthropic. En savoir plus et accéder aux ressources (rapport SOC 2 Type 2, certificat ISO 27001, etc.) sur le [Centre de confiance Anthropic](#).

Architecture basée sur les permissions

Claude Code utilise des permissions strictes en lecture seule par défaut. Lorsque des actions supplémentaires sont nécessaires (édition de fichiers, exécution de tests, exécution de commandes), Claude Code demande une permission explicite. Les utilisateurs contrôlent s'il faut approuver les actions une seule fois ou les autoriser automatiquement.

Nous avons conçu Claude Code pour être transparent et sécurisé. Par exemple, nous exigeons une approbation pour les commandes bash avant de les exécuter, vous donnant un contrôle direct. Cette approche permet aux utilisateurs et aux organisations de configurer les permissions directement.

Pour une configuration détaillée des permissions, consultez [Permissions](#).

Protections intégrées

Pour atténuer les risques dans les systèmes agentiques :

- **Outil bash en sandbox** : [Sandbox](#) les commandes bash avec isolation du système de fichiers et du réseau, réduisant les invites de permission tout en maintenant la sécurité. Activez avec `/sandbox` pour définir les limites où Claude Code peut travailler de manière autonome
- **Restriction d'accès en écriture** : Claude Code ne peut écrire que dans le dossier où il a été démarré et ses sous-dossiers—il ne peut pas modifier les fichiers dans les

répertoires parents sans permission explicite. Bien que Claude Code puisse lire les fichiers en dehors du répertoire de travail (utile pour accéder aux bibliothèques système et aux dépendances), les opérations d'écriture sont strictement limitées à la portée du projet, créant une limite de sécurité claire

- **Atténuation de la fatigue des invites** : Support pour la liste blanche des commandes sûres fréquemment utilisées par utilisateur, par base de code ou par organisation
- **Mode Accepter les modifications** : Accepter par lot plusieurs modifications tout en maintenant les invites de permission pour les commandes avec effets secondaires

Responsabilité de l'utilisateur

Claude Code n'a que les permissions que vous lui accordez. Vous êtes responsable de l'examen du code et des commandes proposés pour la sécurité avant approbation.

Protégez-vous contre l'injection de prompt

L'injection de prompt est une technique où un attaquant tente de contourner ou de manipuler les instructions d'un assistant IA en insérant du texte malveillant. Claude Code inclut plusieurs protections contre ces attaques :

Protections principales

- **Système de permissions** : Les opérations sensibles nécessitent une approbation explicite
- **Analyse contextuelle** : Détecte les instructions potentiellement nuisibles en analysant la demande complète
- **Assainissement des entrées** : Préviend l'injection de commandes en traitant les entrées utilisateur
- **Liste noire de commandes** : Bloque les commandes risquées qui récupèrent du contenu arbitraire sur le web comme `curl` et `wget` par défaut. Lorsqu'elles sont explicitement autorisées, soyez conscient des [limitations des modèles de permission](#)

Protections de la vie privée

Nous avons mis en place plusieurs protections pour protéger vos données, notamment :

- Périodes de rétention limitées pour les informations sensibles (consultez le [Centre de confidentialité](#) pour en savoir plus)
- Accès restreint aux données de session utilisateur
- Contrôle utilisateur sur les préférences de formation des données. Les utilisateurs consommateurs peuvent modifier leurs [paramètres de confidentialité](#) à tout moment.

Pour plus de détails, veuillez consulter nos [Conditions commerciales](#) (pour les utilisateurs Team, Enterprise et API) ou [Conditions pour les consommateurs](#) (pour les utilisateurs Free, Pro et Max) et [Politique de confidentialité](#).

Protections supplémentaires

- **Approbation des demandes réseau** : Les outils qui effectuent des demandes réseau nécessitent une approbation utilisateur par défaut
- **Fenêtres de contexte isolées** : Web fetch utilise une fenêtre de contexte séparée pour éviter d'injecter des prompts potentiellement malveillants
- **Vérification de confiance** : Les premières exécutions de base de code et les nouveaux serveurs MCP nécessitent une vérification de confiance
 - Remarque : La vérification de confiance est désactivée lors de l'exécution non-interactive avec le drapeau `-p`
- **Détection d'injection de commande** : Les commandes bash suspectes nécessitent une approbation manuelle même si elles ont été précédemment autorisées
- **Correspondance en cas d'échec fermé** : Les commandes non appariées par défaut nécessitent une approbation manuelle
- **Descriptions en langage naturel** : Les commandes bash complexes incluent des explications pour la compréhension de l'utilisateur
- **Stockage sécurisé des identifiants** : Les clés API et les tokens sont chiffrés. Consultez [Gestion des identifiants](#)

Warning:

Risque de sécurité WebDAV Windows : Lors de l'exécution de Claude Code sur Windows, nous recommandons de ne pas activer WebDAV ou de permettre à Claude Code d'accéder à des chemins tels que `\\*` qui peuvent contenir des sous-répertoires WebDAV. [WebDAV a été déprécié par Microsoft](#) en raison de risques de sécurité. L'activation de WebDAV peut permettre à Claude Code de déclencher des demandes réseau vers des hôtes distants, contournant le système de permissions.

Meilleures pratiques pour travailler avec du contenu non fiable :

1. Examinez les commandes suggérées avant approbation
2. Évitez de diriger le contenu non fiable directement vers Claude
3. Vérifiez les modifications proposées aux fichiers critiques
4. Utilisez des machines virtuelles (VM) pour exécuter des scripts et effectuer des appels d'outils, en particulier lors de l'interaction avec des services web externes

5. Signalez les comportements suspects avec `/bug`

Warning:

Bien que ces protections réduisent considérablement les risques, aucun système n'est complètement immunisé contre toutes les attaques. Maintenez toujours de bonnes pratiques de sécurité lors du travail avec n'importe quel outil IA.

Sécurité MCP

Claude Code permet aux utilisateurs de configurer les serveurs Model Context Protocol (MCP). La liste des serveurs MCP autorisés est configurée dans votre code source, dans le cadre des paramètres Claude Code que les ingénieurs enregistrent dans le contrôle de source.

Nous vous encourageons à écrire vos propres serveurs MCP ou à utiliser des serveurs MCP de fournisseurs en qui vous avez confiance. Vous pouvez configurer les permissions Claude Code pour les serveurs MCP. Anthropic ne gère ni n'audite aucun serveur MCP.

Sécurité IDE

Consultez [Sécurité et confidentialité VS Code](#) pour plus d'informations sur l'exécution de Claude Code dans un IDE.

Sécurité de l'exécution cloud

Lors de l'utilisation de [Claude Code sur le web](#), des contrôles de sécurité supplémentaires sont en place :

- **Machines virtuelles isolées** : Chaque session cloud s'exécute dans une VM isolée gérée par Anthropic
- **Contrôles d'accès réseau** : L'accès réseau est limité par défaut et peut être configuré pour être désactivé ou autoriser uniquement des domaines spécifiques
- **Protection des identifiants** : L'authentification est gérée via un proxy sécurisé qui utilise un identifiant limité à l'intérieur du sandbox, qui est ensuite traduit en votre jeton d'authentification GitHub réel
- **Restrictions de branche** : Les opérations de push Git sont limitées à la branche de travail actuelle
- **Journalisation d'audit** : Toutes les opérations dans les environnements cloud sont enregistrées à des fins de conformité et d'audit

- **Nettoyage automatique** : Les environnements cloud sont automatiquement terminés après la fin de la session

Pour plus de détails sur l'exécution cloud, consultez [Claude Code sur le web](#).

Les sessions de [Contrôle à distance](#) fonctionnent différemment : l'interface web se connecte à un processus Claude Code s'exécutant sur votre machine locale. Toute l'exécution du code et l'accès aux fichiers restent locaux, et les mêmes données qui circulent lors de toute session Claude Code locale transitent par l'API Anthropic via TLS. Aucune VM cloud ou sandbox n'est impliquée. La connexion utilise plusieurs identifiants de courte durée et à portée étroite, chacun limité à un objectif spécifique et expirant indépendamment, pour limiter le rayon d'explosion de tout identifiant compromis unique.

Meilleures pratiques de sécurité

Travail avec du code sensible

- Examinez toutes les modifications suggérées avant approbation
- Utilisez les paramètres de permission spécifiques au projet pour les référentiels sensibles
- Envisagez d'utiliser les [devcontainers](#) pour une isolation supplémentaire
- Auditez régulièrement vos paramètres de permission avec `/permissions`

Sécurité d'équipe

- Utilisez les [paramètres gérés](#) pour appliquer les normes organisationnelles
- Partagez les configurations de permission approuvées via le contrôle de source
- Formez les membres de l'équipe aux meilleures pratiques de sécurité
- Surveillez l'utilisation de Claude Code via les [métriques OpenTelemetry](#)
- Auditez ou bloquez les modifications de paramètres pendant les sessions avec les [hooks ConfigChange](#)

Signalement des problèmes de sécurité

Si vous découvrez une vulnérabilité de sécurité dans Claude Code :

1. Ne la divulguez pas publiquement
2. Signalez-la via notre [programme HackerOne](#)
3. Incluez les étapes de reproduction détaillées
4. Accordez-nous du temps pour résoudre le problème avant la divulgation publique

Ressources connexes

- [Sandboxing](#) - Isolation du système de fichiers et du réseau pour les commandes bash
- [Permissions](#) - Configurer les permissions et les contrôles d'accès
- [Surveillance de l'utilisation](#) - Suivre et auditer l'activité Claude Code
- [Conteneurs de développement](#) - Environnements sécurisés et isolés
- [Centre de confiance Anthropic](#) - Certifications de sécurité et conformité

Sandboxing

Découvrez comment l'outil bash en sandbox de Claude Code fournit une isolation du système de fichiers et du réseau pour une exécution d'agent plus sûre et plus autonome.

Aperçu

Claude Code dispose d'un sandboxing natif pour fournir un environnement plus sécurisé pour l'exécution des agents tout en réduisant le besoin de demandes de permission constantes. Au lieu de demander une permission pour chaque commande bash, le sandboxing crée des limites définies à l'avance où Claude Code peut travailler plus librement avec un risque réduit.

L'outil bash en sandbox utilise des primitives au niveau du système d'exploitation pour appliquer à la fois l'isolation du système de fichiers et du réseau.

Pourquoi le sandboxing est important

La sécurité basée sur les permissions traditionnelles nécessite une approbation constante de l'utilisateur pour les commandes bash. Bien que cela offre un contrôle, cela peut entraîner :

- **Fatigue d'approbation** : Cliquer répétitivement sur « approuver » peut amener les utilisateurs à prêter moins attention à ce qu'ils approuvent
- **Productivité réduite** : Les interruptions constantes ralentissent les flux de travail de développement
- **Autonomie limitée** : Claude Code ne peut pas fonctionner aussi efficacement en attendant les approbations

Le sandboxing résout ces défis en :

1. **Définissant des limites claires** : Spécifiez exactement quels répertoires et hôtes réseau Claude Code peut accéder
2. **Réduisant les demandes de permission** : Les commandes sûres dans le sandbox ne nécessitent pas d'approbation
3. **Maintenant la sécurité** : Les tentatives d'accès aux ressources en dehors du sandbox déclenchent des notifications immédiates

4. **Permettant l'autonomie** : Claude Code peut fonctionner plus indépendamment dans les limites définies

Warning:

Un sandboxing efficace nécessite **à la fois** l'isolation du système de fichiers et du réseau. Sans isolation réseau, un agent compromis pourrait exfiltrer des fichiers sensibles comme les clés SSH. Sans isolation du système de fichiers, un agent compromis pourrait installer une porte dérobée sur les ressources système pour accéder au réseau. Lors de la configuration du sandboxing, il est important de s'assurer que vos paramètres configurés ne créent pas de contournements dans ces systèmes.

Comment ça marche

Isolation du système de fichiers

L'outil bash en sandbox restreint l'accès au système de fichiers à des répertoires spécifiques :

- **Comportement d'écriture par défaut** : Accès en lecture et écriture au répertoire de travail actuel et à ses sous-répertoires
- **Comportement de lecture par défaut** : Accès en lecture à l'ensemble de l'ordinateur, sauf certains répertoires refusés
- **Accès bloqué** : Impossible de modifier les fichiers en dehors du répertoire de travail actuel sans permission explicite
- **Configurable** : Définissez des chemins autorisés et refusés personnalisés via les paramètres

Vous pouvez accorder l'accès en écriture à des chemins supplémentaires en utilisant `sandbox.filesystem.allowWrite` dans vos paramètres. Ces restrictions sont appliquées au niveau du système d'exploitation (Seatbelt sur macOS, bubblewrap sur Linux), elles s'appliquent donc à toutes les commandes de sous-processus, y compris les outils comme `kubectl`, `terraform` et `npm`, pas seulement aux outils de fichiers de Claude.

Isolation réseau

L'accès réseau est contrôlé via un serveur proxy s'exécutant en dehors du sandbox :

- **Restrictions de domaine** : Seuls les domaines approuvés peuvent être accédés
- **Confirmation de l'utilisateur** : Les nouvelles demandes de domaine déclenchent des demandes de permission (sauf si `allowManagedDomainsOnly` est activé, ce qui bloque automatiquement les domaines non autorisés)

- **Support de proxy personnalisé** : Les utilisateurs avancés peuvent implémenter des règles personnalisées sur le trafic sortant
- **Couverture complète** : Les restrictions s'appliquent à tous les scripts, programmes et sous-processus générés par les commandes

Application au niveau du système d'exploitation

L'outil bash en sandbox exploite les primitives de sécurité du système d'exploitation :

- **macOS** : Utilise Seatbelt pour l'application du sandbox
- **Linux** : Utilise [bubblewrap](#) pour l'isolation
- **WSL2** : Utilise bubblewrap, comme Linux

WSL1 n'est pas supporté car bubblewrap nécessite des fonctionnalités du noyau uniquement disponibles dans WSL2.

Ces restrictions au niveau du système d'exploitation garantissent que tous les processus enfants générés par les commandes de Claude Code héritent des mêmes limites de sécurité.

Démarrage

Prérequis

Sur **macOS**, le sandboxing fonctionne directement en utilisant le framework Seatbelt intégré.

Sur **Linux et WSL2**, installez d'abord les packages requis :

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

Activer le sandboxing

Vous pouvez activer le sandboxing en exécutant la commande `/sandbox` :

```
/sandbox
```

Cela ouvre un menu où vous pouvez choisir entre les modes de sandbox. Si les dépendances requises sont manquantes (comme `bubblewrap` ou `socat` sur Linux), le menu affiche les instructions d'installation pour votre plateforme.

Modes de sandbox

Claude Code offre deux modes de sandbox :

Mode auto-allow : Les commandes Bash tenteront de s'exécuter dans le sandbox et sont automatiquement autorisées sans nécessiter de permission. Les commandes qui ne peuvent pas être sandboxées (comme celles nécessitant un accès réseau à des hôtes non autorisés) reviennent au flux de permission régulier. Les règles d'ask/deny explicites que vous avez configurées sont toujours respectées.

Mode permissions régulières : Toutes les commandes bash passent par le flux de permission standard, même lorsqu'elles sont sandboxées. Cela offre plus de contrôle mais nécessite plus d'approbations.

Dans les deux modes, le sandbox applique les mêmes restrictions de système de fichiers et de réseau. La différence réside uniquement dans le fait que les commandes sandboxées sont auto-approuvées ou nécessitent une permission explicite.

Info:

Le mode auto-allow fonctionne indépendamment de votre paramètre de mode de permission. Même si vous n'êtes pas en mode « accepter les modifications », les commandes bash sandboxées s'exécuteront automatiquement lorsque l'auto-allow est activé. Cela signifie que les commandes bash qui modifient les fichiers dans les limites du sandbox s'exécuteront sans invite, même lorsque les outils de modification de fichiers nécessiteraient normalement une approbation.

Configurer le sandboxing

Personnalisez le comportement du sandbox via votre fichier `settings.json`. Consultez [Paramètres](#) pour la référence de configuration complète.

Accorder l'accès en écriture des sous-processus à des chemins spécifiques

Par défaut, les commandes sandboxées ne peuvent écrire que dans le répertoire de travail actuel. Si les commandes de sous-processus comme `kubectl`, `terraform` ou `npm` doivent écrire en dehors du répertoire du projet, utilisez `sandbox.filesystem.allowWrite` pour accorder l'accès à des chemins spécifiques :

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "//tmp/build"]
    }
  }
}
```

Ces chemins sont appliqués au niveau du système d'exploitation, donc toutes les commandes s'exécutant dans le sandbox, y compris leurs processus enfants, les respectent. C'est l'approche recommandée lorsqu'un outil a besoin d'un accès en écriture à un emplacement spécifique, plutôt que d'exclure complètement l'outil du sandbox avec `excludedCommands`.

Lorsque `allowWrite` (ou `denyWrite` / `denyRead`) est défini dans plusieurs [portées de paramètres](#), les tableaux sont **fusionnés**, ce qui signifie que les chemins de chaque portée sont combinés, non remplacés. Par exemple, si les paramètres gérés autorisent les écritures à `//opt/company-tools` et qu'un utilisateur ajoute `~/.kube` dans ses paramètres personnels, les deux chemins sont inclus dans la configuration finale du sandbox. Cela signifie que les utilisateurs et les projets peuvent étendre la liste sans dupliquer ou remplacer les chemins définis par les portées de priorité plus élevée.

Les préfixes de chemin contrôlent la façon dont les chemins sont résolus :

Préfixe	Signification	Exemple
<code>//</code>	Chemin absolu à partir de la racine du système de fichiers	<code>//tmp/build</code> devient <code>/tmp/build</code>
<code>~/</code>	Relatif au répertoire personnel	<code>~/.kube</code> devient <code>\$HOME/.kube</code>

Préfixe	Signification	Exemple
/	Relatif au répertoire du fichier de paramètres	/build devient \$SETTINGSDIR/build
./ ou pas de préfixe	Chemin relatif (résolu par le runtime du sandbox)	./output

Vous pouvez également refuser l'accès en écriture ou en lecture en utilisant `sandbox.filesystem.denyWrite` et `sandbox.filesystem.denyRead`. Ceux-ci sont fusionnés avec tous les chemins des règles de permission `Edit(...)` et `Read(...)`.

Tip:

Toutes les commandes ne sont pas compatibles avec le sandboxing directement. Quelques notes qui peuvent vous aider à tirer le meilleur parti du sandbox :

- De nombreux outils CLI nécessitent d'accéder à certains hôtes. Au fur et à mesure que vous utilisez ces outils, ils demanderont la permission d'accéder à certains hôtes. Accorder la permission leur permettra d'accéder à ces hôtes maintenant et à l'avenir, leur permettant de s'exécuter en toute sécurité dans le sandbox.
- `watchman` est incompatible avec l'exécution dans le sandbox. Si vous exécutez `jest`, envisagez d'utiliser `jest --no-watchman`.
- `docker` est incompatible avec l'exécution dans le sandbox. Envisagez de spécifier `docker` dans `excludedCommands` pour le forcer à s'exécuter en dehors du sandbox.

Note:

Claude Code inclut un mécanisme d'échappatoire intentionnel qui permet aux commandes de s'exécuter en dehors du sandbox si nécessaire. Lorsqu'une commande échoue en raison des restrictions du sandbox (comme les problèmes de connectivité réseau ou les outils incompatibles), Claude est invité à analyser l'échec et peut réessayer la commande avec le paramètre `dangerouslyDisableSandbox`. Les commandes qui utilisent ce paramètre passent par le flux de permissions normal de Claude Code nécessitant une permission de l'utilisateur pour s'exécuter. Cela permet à Claude Code de gérer les cas limites où certains outils ou opérations réseau ne peuvent pas fonctionner dans les contraintes du sandbox.

Vous pouvez désactiver cet échappatoire en définissant `"allowUnsandboxedCommands": false` dans vos [paramètres de sandbox](#). Lorsqu'il est désactivé, le paramètre `dangerouslyDisableSandbox` est complètement ignoré et toutes les commandes doivent s'exécuter sandboxées ou être explicitement listées dans `excludedCommands`.

Avantages de sécurité

Protection contre l'injection de prompt

Même si un attaquant manipule avec succès le comportement de Claude Code par injection de prompt, le sandbox garantit que votre système reste sécurisé :

Protection du système de fichiers :

- Impossible de modifier les fichiers de configuration critiques tels que `~/.bashrc`
- Impossible de modifier les fichiers au niveau du système dans `/bin/`
- Impossible de lire les fichiers qui sont refusés dans vos [paramètres de permission Claude](#)

Protection réseau :

- Impossible d'exfiltrer les données vers des serveurs contrôlés par l'attaquant
- Impossible de télécharger des scripts malveillants à partir de domaines non autorisés
- Impossible de faire des appels API inattendus vers des services non approuvés
- Impossible de contacter des domaines non explicitement autorisés

Surveillance et contrôle :

- Toutes les tentatives d'accès en dehors du sandbox sont bloquées au niveau du système d'exploitation
- Vous recevez des notifications immédiates lorsque les limites sont testées
- Vous pouvez choisir de refuser, d'autoriser une fois ou de mettre à jour définitivement votre configuration

Surface d'attaque réduite

Le sandboxing limite les dommages potentiels causés par :

- **Dépendances malveillantes** : Packages NPM ou autres dépendances avec du code nuisible
- **Scripts compromis** : Scripts de construction ou outils avec des vulnérabilités de sécurité
- **Ingénierie sociale** : Attaques qui trompent les utilisateurs pour qu'ils exécutent des commandes dangereuses
- **Injection de prompt** : Attaques qui trompent Claude pour qu'il exécute des commandes dangereuses

Fonctionnement transparent

Lorsque Claude Code tente d'accéder à des ressources réseau en dehors du sandbox :

1. L'opération est bloquée au niveau du système d'exploitation
2. Vous recevez une notification immédiate
3. Vous pouvez choisir de :
 - Refuser la demande
 - L'autoriser une fois
 - Mettre à jour votre configuration de sandbox pour l'autoriser définitivement

Limitations de sécurité

- Limitations du sandboxing réseau : Le système de filtrage réseau fonctionne en restreignant les domaines auxquels les processus sont autorisés à se connecter. Il n'inspecte pas autrement le trafic passant par le proxy et les utilisateurs sont responsables de s'assurer qu'ils n'autorisent que les domaines de confiance dans leur politique.

Warning:

Les utilisateurs doivent être conscients des risques potentiels liés à l'autorisation de domaines larges comme `github.com` qui peuvent permettre l'exfiltration de données. De plus, dans certains cas, il peut être possible de contourner le filtrage réseau via [domain fronting](#).

- Escalade de privilèges via les sockets Unix : La configuration `allowUnixSockets` peut accorder involontairement l'accès à des services système puissants qui pourraient entraîner des contournements du sandbox. Par exemple, si elle est utilisée pour autoriser l'accès à `/var/run/docker.sock`, cela accorderait effectivement l'accès au système hôte en exploitant le socket docker. Les utilisateurs sont encouragés à examiner attentivement tous les sockets Unix qu'ils autorisent via le sandbox.
- Escalade de permissions du système de fichiers : Les permissions d'écriture du système de fichiers trop larges peuvent permettre des attaques d'escalade de privilèges. Autoriser les écritures dans les répertoires contenant des exécutables dans `$PATH`, les répertoires de configuration système ou les fichiers de configuration du shell utilisateur (`.bashrc` , `.zshrc`) peut entraîner l'exécution de code dans différents contextes de sécurité lorsque d'autres utilisateurs ou processus système accèdent à ces fichiers.
- Force du sandbox Linux : L'implémentation Linux fournit une isolation forte du système de fichiers et du réseau mais inclut un mode `enableWeakerNestedSandbox` qui lui permet de fonctionner à l'intérieur des environnements Docker sans espaces de

noms privilégiés. Cette option affaiblit considérablement la sécurité et ne doit être utilisée que dans les cas où une isolation supplémentaire est autrement appliquée.

Comment le sandboxing se rapporte aux permissions

Le sandboxing et les [permissions](#) sont des couches de sécurité complémentaires qui fonctionnent ensemble :

- **Les permissions** contrôlent quels outils Claude Code peut utiliser et sont évaluées avant l'exécution de tout outil. Elles s'appliquent à tous les outils : Bash, Read, Edit, WebFetch, MCP et autres.
- **Le sandboxing** fournit une application au niveau du système d'exploitation qui restreint ce que les commandes Bash peuvent accéder au niveau du système de fichiers et du réseau. Il s'applique uniquement aux commandes Bash et à leurs processus enfants.

Les restrictions du système de fichiers et du réseau sont configurées via les paramètres de sandbox et les règles de permission :

- Utilisez `sandbox.filesystem.allowWrite` pour accorder l'accès en écriture des sous-processus à des chemins en dehors du répertoire de travail
- Utilisez `sandbox.filesystem.denyWrite` et `sandbox.filesystem.denyRead` pour bloquer l'accès des sous-processus à des chemins spécifiques
- Utilisez les règles de refus `Read` et `Edit` pour bloquer l'accès à des fichiers ou répertoires spécifiques
- Utilisez les règles d'autorisation/refus `WebFetch` pour contrôler l'accès au domaine
- Utilisez les `allowedDomains` du sandbox pour contrôler quels domaines les commandes Bash peuvent atteindre

Les chemins des paramètres `sandbox.filesystem` et des règles de permission sont fusionnés dans la configuration finale du sandbox.

Ce [référentiel](#) inclut des configurations de paramètres de démarrage pour les scénarios de déploiement courants, y compris des exemples spécifiques au sandbox. Utilisez-les comme points de départ et ajustez-les selon vos besoins.

Utilisation avancée

Configuration de proxy personnalisée

Pour les organisations nécessitant une sécurité réseau avancée, vous pouvez implémenter un proxy personnalisé pour :

- Déchiffrer et inspecter le trafic HTTPS

- Appliquer des règles de filtrage personnalisées
- Enregistrer toutes les demandes réseau
- Intégrer avec l'infrastructure de sécurité existante

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

Intégration avec les outils de sécurité existants

L'outil bash en sandbox fonctionne aux côtés de :

- **Règles de permission** : Combinez avec les [paramètres de permission](#) pour une défense en profondeur
- **Conteneurs de développement** : Utilisez avec [devcontainers](#) pour une isolation supplémentaire
- **Politiques d'entreprise** : Appliquez les configurations de sandbox via les [paramètres gérés](#)

Meilleures pratiques

1. **Commencez restrictif** : Commencez avec des permissions minimales et développez selon les besoins
2. **Surveillez les journaux** : Examinez les tentatives de violation du sandbox pour comprendre les besoins de Claude Code
3. **Utilisez des configurations spécifiques à l'environnement** : Différentes règles de sandbox pour les contextes de développement par rapport à la production
4. **Combinez avec les permissions** : Utilisez le sandboxing aux côtés des politiques IAM pour une sécurité complète
5. **Testez les configurations** : Vérifiez que vos paramètres de sandbox ne bloquent pas les flux de travail légitimes

Open source

Le runtime du sandbox est disponible en tant que package npm open source pour une utilisation dans vos propres projets d'agent. Cela permet à la communauté plus large des agents IA de construire des systèmes autonomes plus sûrs et plus sécurisés. Cela peut également être utilisé pour sandboxer d'autres programmes que vous souhaitez exécuter. Par exemple, pour sandboxer un serveur MCP, vous pouvez exécuter :

```
npx @anthropic-ai/sandbox-runtime <command-to-sandbox>
```

Pour les détails d'implémentation et le code source, visitez le [référentiel GitHub](#).

Limitations

- **Surcharge de performance** : Minimale, mais certaines opérations du système de fichiers peuvent être légèrement plus lentes
- **Compatibilité** : Certains outils qui nécessitent des modèles d'accès système spécifiques peuvent nécessiter des ajustements de configuration, ou même devront être exécutés en dehors du sandbox
- **Support de plateforme** : Supporte macOS, Linux et WSL2. WSL1 n'est pas supporté. Le support natif de Windows est prévu.

Voir aussi

- [Sécurité](#) - Fonctionnalités de sécurité complètes et meilleures pratiques
- [Permissions](#) - Configuration des permissions et contrôle d'accès
- [Paramètres](#) - Référence de configuration complète
- [Référence CLI](#) - Options de ligne de commande

Checkpointing

Suivez automatiquement et annulez les modifications de Claude pour récupérer rapidement des changements indésirables.

Claude Code suit automatiquement les modifications de fichiers effectuées par Claude au fur et à mesure que vous travaillez, ce qui vous permet d'annuler rapidement les modifications et de revenir à des états antérieurs si quelque chose s'écarte de la trajectoire.

Fonctionnement des checkpoints

Au fur et à mesure que vous travaillez avec Claude, le checkpointing capture automatiquement l'état de votre code avant chaque modification. Ce filet de sécurité vous permet de poursuivre des tâches ambitieuses et à grande échelle en sachant que vous pouvez toujours revenir à un état de code antérieur.

Suivi automatique

Claude Code suit tous les changements effectués par ses outils d'édition de fichiers :

- Chaque message utilisateur crée un nouveau checkpoint
- Les checkpoints persistent entre les sessions, vous pouvez donc y accéder dans les conversations reprises
- Nettoyés automatiquement avec les sessions après 30 jours (configurable)

Annulation des modifications

Appuyez sur **Esc** deux fois (**Esc** + **Esc**) ou utilisez la commande **/rewind** pour ouvrir le menu d'annulation. Vous pouvez choisir de restaurer :

- **Conversation uniquement** : Annulez jusqu'à un message utilisateur tout en conservant les modifications de code
- **Code uniquement** : Annulez les modifications de fichiers tout en conservant la conversation
- **Code et conversation** : Restaurez les deux à un point antérieur de la session

Cas d'usage courants

Les checkpoints sont particulièrement utiles lorsque :

- **Explorer des alternatives** : Essayez différentes approches d'implémentation sans perdre votre point de départ
- **Récupérer après des erreurs** : Annulez rapidement les modifications qui ont introduit des bugs ou cassé des fonctionnalités
- **Itérer sur les fonctionnalités** : Expérimentez des variations en sachant que vous pouvez revenir à des états fonctionnels

Limitations

Les modifications de commandes Bash ne sont pas suivies

Le checkpointing ne suit pas les fichiers modifiés par les commandes bash. Par exemple, si Claude Code exécute :

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

Ces modifications de fichiers ne peuvent pas être annulées via l'annulation. Seules les modifications de fichiers directes effectuées via les outils d'édition de fichiers de Claude sont suivies.

Les modifications externes ne sont pas suivies

Le checkpointing suit uniquement les fichiers qui ont été modifiés au cours de la session actuelle. Les modifications manuelles que vous apportez aux fichiers en dehors de Claude Code et les modifications d'autres sessions concurrentes ne sont normalement pas capturées, sauf si elles modifient les mêmes fichiers que la session actuelle.

Pas un remplacement du contrôle de version

Les checkpoints sont conçus pour une récupération rapide au niveau de la session. Pour un historique de version permanent et la collaboration :

- Continuez à utiliser le contrôle de version (ex. Git) pour les commits, les branches et l'historique à long terme
- Les checkpoints complètent mais ne remplacent pas le contrôle de version approprié

- Pensez aux checkpoints comme « annulation locale » et à Git comme « historique permanent »

Voir aussi

- [Mode interactif](#) - Raccourcis clavier et contrôles de session
- [Commandes intégrées](#) - Accès aux checkpoints via `/rewind`
- [Référence CLI](#) - Options de ligne de commande

Utilisation des données

Découvrez les politiques d'utilisation des données d'Anthropic pour Claude

Politiques de données

Politique de formation aux données

Utilisateurs grand public (plans Free, Pro et Max) : Nous vous donnons le choix de permettre à vos données d'être utilisées pour améliorer les futurs modèles Claude. Nous formerons de nouveaux modèles en utilisant les données des comptes Free, Pro et Max lorsque ce paramètre est activé (y compris lorsque vous utilisez Claude Code à partir de ces comptes).

Utilisateurs commerciaux : (plans Team et Enterprise, API, plateformes tierces et Claude Gov) maintiennent les politiques existantes : Anthropic ne forme pas de modèles génératifs en utilisant le code ou les invites envoyés à Claude Code selon les conditions commerciales, sauf si le client a choisi de nous fournir ses données pour l'amélioration du modèle (par exemple, le [Programme de partenariat pour développeurs](#)).

Programme de partenariat pour développeurs

Si vous acceptez explicitement les méthodes pour nous fournir des matériaux à utiliser pour la formation, comme via le [Programme de partenariat pour développeurs](#), nous pouvons utiliser ces matériaux fournis pour former nos modèles. Un administrateur d'organisation peut accepter explicitement le Programme de partenariat pour développeurs pour son organisation. Notez que ce programme est disponible uniquement pour l'API Anthropic propriétaire, et non pour les utilisateurs de Bedrock ou Vertex.

Retours d'information à l'aide de la commande `/bug`

Si vous choisissez de nous envoyer des retours d'information sur Claude Code à l'aide de la commande `/bug`, nous pouvons utiliser vos retours d'information pour améliorer nos produits et services. Les transcriptions partagées via `/bug` sont conservées pendant 5 ans.

Sondages de qualité de session

Lorsque vous voyez l'invite « Comment Claude s'en sort-il cette session ? » dans Claude Code, répondre à ce sondage (y compris en sélectionnant « Ignorer »), seule votre note numérique (1, 2, 3 ou ignorer) est enregistrée. Nous ne collectons ni ne stockons de trans-

criptions de conversation, d'entrées, de sorties ou d'autres données de session dans le cadre de ce sondage. Contrairement aux retours d'information avec pouces vers le haut/bas ou aux rapports [/bug](#), ce sondage de qualité de session est une simple métrique de satisfaction du produit. Vos réponses à ce sondage n'affectent pas vos préférences de formation aux données et ne peuvent pas être utilisées pour former nos modèles d'IA.

Pour désactiver ces sondages, définissez `CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1`. Le sondage est également automatiquement désactivé lors de l'utilisation de fournisseurs tiers (Bedrock, Vertex, Foundry) ou lorsque la télémétrie est désactivée.

Conservation des données

Anthropic conserve les données de Claude Code en fonction de votre type de compte et de vos préférences.

Utilisateurs grand public (plans Free, Pro et Max) :

- Utilisateurs qui autorisent l'utilisation des données pour l'amélioration du modèle : période de conservation de 5 ans pour soutenir le développement et les améliorations de sécurité du modèle
- Utilisateurs qui n'autorisent pas l'utilisation des données pour l'amélioration du modèle : période de conservation de 30 jours
- Les paramètres de confidentialité peuvent être modifiés à tout moment sur claude.ai/settings/data-privacy-controls.

Utilisateurs commerciaux (Team, Enterprise et API) :

- Standard : période de conservation de 30 jours
- [Conservation zéro des données](#) : disponible pour Claude Code sur Claude for Enterprise. La conservation zéro des données est activée par organisation ; chaque nouvelle organisation doit avoir la conservation zéro des données activée séparément par votre équipe de compte
- Mise en cache locale : les clients Claude Code peuvent stocker les sessions localement pendant jusqu'à 30 jours pour permettre la reprise de session (configurable)

Vous pouvez supprimer les sessions individuelles de Claude Code sur le web à tout moment. La suppression d'une session supprime définitivement les données d'événement de la session. Pour obtenir des instructions sur la suppression des sessions, consultez [Gestion des sessions](#).

Découvrez-en plus sur les pratiques de conservation des données dans notre [Centre de confidentialité](#).

Pour plus de détails, veuillez consulter nos [Conditions commerciales](#) (pour les utilisateurs Team, Enterprise et API) ou [Conditions grand public](#) (pour les utilisateurs Free, Pro et Max) et [Politique de confidentialité](#).

Accès aux données

Pour tous les utilisateurs propriétaires, vous pouvez en savoir plus sur les données enregistrées pour [Claude Code local](#) et [Claude Code distant](#). Les sessions [Contrôle distant](#) suivent le flux de données local puisque toute l'exécution se fait sur votre machine. Notez que pour Claude Code distant, Claude accède au référentiel où vous lancez votre session Claude Code. Claude n'accède pas aux référentiels que vous avez connectés mais dans lesquels vous n'avez pas lancé de session.

Claude Code local : flux de données et dépendances

Le diagramme ci-dessous montre comment Claude Code se connecte aux services externes lors de l'installation et du fonctionnement normal. Les lignes pleines indiquent les connexions requises, tandis que les lignes pointillées représentent les flux de données optionnels ou initiés par l'utilisateur.

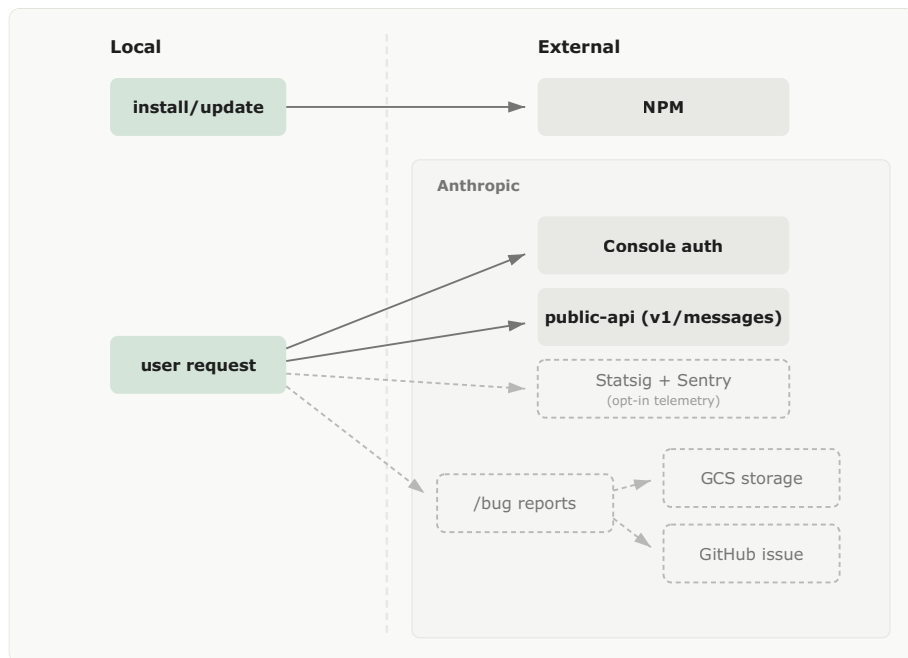


Diagramme montrant les connexions externes de Claude Code : l'installation/mise à jour se connecte à NPM, et les demandes des utilisateurs se connectent aux services Anthropic, y compris l'authentification Console, l'API publique, et optionnellement Statsig, Sentry et les rapports de bogues

Claude Code est installé à partir de [NPM](#). Claude Code s'exécute localement. Pour interagir avec le LLM, Claude Code envoie des données sur le réseau. Ces données incluent tous les invites utilisateur et les sorties du modèle. Les données sont chiffrées en transit via TLS et ne sont pas chiffrées au repos. Claude Code est compatible avec la plupart des VPN et proxies LLM populaires.

Claude Code est construit sur les API d'Anthropic. Pour plus de détails concernant les contrôles de sécurité de notre API, y compris nos procédures de journalisation des API, veuillez consulter les artefacts de conformité offerts dans le [Centre de confiance Anthropic](#).

Exécution cloud : flux de données et dépendances

Lors de l'utilisation de [Claude Code sur le web](#), les sessions s'exécutent dans des machines virtuelles gérées par Anthropic au lieu de s'exécuter localement. Dans les environnements cloud :

- **Stockage du code et des données :** Votre référentiel est cloné sur une VM isolée. Le code et les données de session sont soumis aux politiques de conservation et d'utilisation pour votre type de compte (voir la section Conservation des données ci-dessus)
- **Identifiants :** L'authentification GitHub est gérée via un proxy sécurisé ; vos identifiants GitHub n'entrent jamais dans le bac à sable
- **Trafic réseau :** Tout le trafic sortant passe par un proxy de sécurité pour la journalisation d'audit et la prévention des abus
- **Données de session :** Les invites, les modifications de code et les sorties suivent les mêmes politiques de données que l'utilisation locale de Claude Code

Pour plus de détails sur la sécurité de l'exécution cloud, consultez [Sécurité](#).

Services de télémétrie

Claude Code se connecte à partir des machines des utilisateurs au service Statsig pour enregistrer les métriques opérationnelles telles que la latence, la fiabilité et les modèles d'utilisation. Cet enregistrement n'inclut aucun code ni chemin de fichier. Les données sont chiffrées en transit à l'aide de TLS et au repos à l'aide du chiffrement AES 256 bits. Lisez-en plus dans la [documentation de sécurité Statsig](#). Pour refuser la télémétrie Statsig, définissez la variable d'environnement `DISABLE_TELEMETRY`.

Claude Code se connecte à partir des machines des utilisateurs à Sentry pour la journalisation des erreurs opérationnelles. Les données sont chiffrées en transit à l'aide de TLS et au repos à l'aide du chiffrement AES 256 bits. Lisez-en plus dans la [documentation de sécurité Sentry](#). Pour refuser la journalisation des erreurs, définissez la variable d'environnement `DISABLE_ERROR_REPORTING`.

Lorsque les utilisateurs exécutent la commande `/bug`, une copie de leur historique de conversation complet, y compris le code, est envoyée à Anthropic. Les données sont chiffrées en transit et au repos. Optionnellement, un problème Github est créé dans notre référentiel public. Pour refuser les rapports de bogues, définissez la variable d'environnement `DISABLE_BUG_COMMAND`.

Comportements par défaut par fournisseur d'API

Par défaut, nous désactivons tout le trafic non essentiel (y compris les rapports d'erreurs, la télémétrie, la fonctionnalité de rapport de bogues et les sondages de qualité de session) lors de l'utilisation de Bedrock, Vertex ou Foundry. Vous pouvez également refuser tous ces éléments à la fois en définissant la variable d'environnement

`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC`. Voici les comportements par défaut complets :

Service	Claude API	Vertex API	Bedrock API	Foundry API
Statsig (Métriques)	Activé par défaut. <code>DISABLE_TELEMETRY=1</code> pour désactiver.	Désactivé par défaut. <code>CLAUDE_CODE_USE_VERTEX</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_BEDROCK</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_FOUNDRY</code> doit être 1.
Sentry (Erreurs)	Activé par défaut. <code>DISABLE_ERROR_REPORTING=1</code> pour désactiver.	Désactivé par défaut. <code>CLAUDE_CODE_USE_VERTEX</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_BEDROCK</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_FOUNDRY</code> doit être 1.
Claude API (rapports / bug)	Activé par défaut. <code>DISABLE_BUG_COMMAND=1</code> pour désactiver.	Désactivé par défaut. <code>CLAUDE_CODE_USE_VERTEX</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_BEDROCK</code> doit être 1.	Désactivé par défaut. <code>CLAUDE_CODE_USE_FOUNDRY</code> doit être 1.

Ser-vice	Claude API	Vertex API	Bedrock API	Foundry API
Son-dages de qua-lité de session	Activé par défaut. CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 pour désactiver.	Désactivé par défaut. CLAUDE_CODE_USE_VERTEX doit être 1.	Désactivé par défaut. CLAUDE_CODE_USE_BEDROCK doit être 1.	Désactivé par défaut. CLAUDE_CODE_USE_FOUNDRY doit être 1.

Toutes les variables d’environnement peuvent être vérifiées dans `settings.json` ([en savoir plus](#)).

Légal et conformité

Accords juridiques, certifications de conformité et informations de sécurité pour Claude Code.

Accords juridiques

Licence

Votre utilisation de Claude Code est soumise à :

- [Conditions commerciales](#) - pour les utilisateurs Team, Enterprise et Claude API
- [Conditions de consommation](#) - pour les utilisateurs Free, Pro et Max

Accords commerciaux

Que vous utilisiez l'API Claude directement (1P) ou y accédiez via AWS Bedrock ou Google Vertex (3P), votre accord commercial existant s'appliquera à l'utilisation de Claude Code, sauf si nous avons convenu autrement.

Conformité

Conformité aux normes de santé (BAA)

Si un client a un accord d'associé commercial (BAA) avec nous et souhaite utiliser Claude Code, le BAA s'étendra automatiquement pour couvrir Claude Code si le client a exécuté un BAA et a activé la rétention zéro des données (ZDR). Le BAA s'appliquera au trafic API de ce client transitant par Claude Code.

Sécurité et confiance

Confiance et sécurité

Vous pouvez trouver plus d'informations dans le [Centre de confiance Anthropic](#) et le [Hub de transparence](#).

Signalement des vulnérabilités de sécurité

Anthropic gère notre programme de sécurité via HackerOne. [Utilisez ce formulaire pour signaler les vulnérabilités](#).

© Anthropic PBC. Tous droits réservés. L'utilisation est soumise aux conditions de service Anthropic applicables.

Zéro conservation des données

Découvrez la conservation zéro des données (ZDR) pour Claude Code sur Claude for Enterprise, y compris la portée, les fonctionnalités désactivées et comment demander l'activation.

La conservation zéro des données (ZDR) est disponible pour Claude Code lorsqu'il est utilisé via Claude for Enterprise. Lorsque ZDR est activée, les invites et les réponses du modèle générées lors des sessions Claude Code sont traitées en temps réel et ne sont pas conservées par Anthropic après le retour de la réponse, sauf si nécessaire pour se conformer à la loi ou combattre les abus.

ZDR sur Claude for Enterprise donne aux clients d'entreprise la possibilité d'utiliser Claude Code avec zéro conservation des données et d'accéder aux capacités administratives :

- Contrôles des coûts par utilisateur
- Tableau de bord [Analytics](#)
- [Paramètres gérés par le serveur](#)
- Journaux d'audit

ZDR pour Claude Code sur Claude for Enterprise s'applique uniquement à la plateforme directe d'Anthropic. Pour les déploiements de Claude sur AWS Bedrock, Google Vertex AI ou Microsoft Foundry, consultez les politiques de conservation des données de ces plateformes.

Portée de ZDR

ZDR couvre l'inférence Claude Code sur Claude for Enterprise.

Warning:

ZDR est activée sur la base de chaque organisation. Chaque nouvelle organisation nécessite que ZDR soit activée séparément par votre équipe de compte Anthropic. ZDR ne s'applique pas automatiquement aux nouvelles organisations créées sous le même compte. Contactez votre équipe de compte pour activer ZDR pour toute nouvelle organisation.

Ce que ZDR couvre

ZDR couvre les appels d’inférence du modèle effectués via Claude Code sur Claude for Enterprise. Lorsque vous utilisez Claude Code dans votre terminal, les invites que vous envoyez et les réponses que Claude génère ne sont pas conservées par Anthropic. Cela s’applique quel que soit le modèle Claude utilisé.

Ce que ZDR ne couvre pas

ZDR ne s’étend pas aux éléments suivants, même pour les organisations avec ZDR activée. Ces fonctionnalités suivent les [politiques standard de conservation des données](#) :

Fonctionnalité	Détails
Chat sur claude.ai	Les conversations de chat via l’interface web Claude for Enterprise ne sont pas couvertes par ZDR.
Cowork	Les sessions Cowork ne sont pas couvertes par ZDR.
Claude Code Analytics	Ne stocke pas les invites ou les réponses du modèle, mais collecte les métadonnées de productivité telles que les e-mails de compte et les statistiques d’utilisation. Les métriques de contribution ne sont pas disponibles pour les organisations ZDR ; le tableau de bord analytics affiche uniquement les métriques d’utilisation.
Gestion des utilisateurs et des sièges	Les données administratives telles que les e-mails de compte et les attributions de sièges sont conservées selon les politiques standard.
Intégrations tierces	Les données traitées par des outils tiers, des serveurs MCP ou d’autres intégrations externes ne sont pas couvertes par ZDR. Examinez indépendamment les pratiques de traitement des données de ces services.

Fonctionnalités désactivées sous ZDR

Lorsque ZDR est activée pour une organisation Claude Code sur Claude for Enterprise, certaines fonctionnalités qui nécessitent de stocker les invites ou les complétions sont automatiquement désactivées au niveau du backend :

Fonctionnalité	Raison
Claude Code sur le Web	Nécessite le stockage côté serveur de l'historique des conversations.
Sessions distantes de l'application Desktop	Nécessite des données de session persistantes qui incluent les invites et les complétions.
Soumission de commentaires (/feedback)	La soumission de commentaires envoie les données de conversation à Anthropic.

Ces fonctionnalités sont bloquées au niveau du backend quel que soit l’affichage côté client. Si vous voyez une fonctionnalité désactivée dans le terminal Claude Code au démarrage, toute tentative d’utilisation retourne une erreur indiquant que les politiques de l’organisation ne permettent pas cette action.

Les futures fonctionnalités peuvent également être désactivées si elles nécessitent de stocker les invites ou les complétions.

Conservation des données pour les violations de politique

Même avec ZDR activée, Anthropic peut conserver les données si la loi l’exige ou pour résoudre les violations de la politique d’utilisation. Si une session est signalée pour une violation de politique, Anthropic peut conserver les entrées et sorties associées pendant jusqu’à 2 ans, conformément à la politique ZDR standard d’Anthropic.

Demander ZDR

Pour demander ZDR pour Claude Code sur Claude for Enterprise, contactez votre équipe de compte Anthropic. Votre équipe de compte soumettra la demande en interne, et Anthropic examinera et activera ZDR sur votre organisation après avoir confirmé l’admissibilité. Toutes les actions d’activation sont enregistrées dans les journaux d’audit.

Si vous utilisez actuellement ZDR pour Claude Code via des clés API à l’usage, vous pouvez passer à Claude for Enterprise pour accéder aux fonctionnalités administratives tout en maintenant ZDR pour Claude Code. Contactez votre équipe de compte pour coordonner la migration.

Part 10: Enterprise & Monitoring

Suivre l'utilisation de l'équipe avec l'analytique

Consultez les métriques d'utilisation de Claude Code, suivez l'adoption et mesurez la vitesse d'ingénierie dans le tableau de bord analytique.

Claude Code fournit des tableaux de bord analytiques pour aider les organisations à comprendre les modèles d'utilisation des développeurs, suivre les métriques de contribution et mesurer l'impact de Claude Code sur la vitesse d'ingénierie. Accédez au tableau de bord pour votre plan :

Plan	URL du tableau de bord	Inclut	En savoir plus
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	Métriques d'utilisation, métriques de contribution avec intégration GitHub, classement, export de données	Détails
API (Claude Console)	platform.claude.com/claude-code	Métriques d'utilisation, suivi des dépenses, insights d'équipe	Détails

Accéder à l'analytique pour Teams et Enterprise

Accédez à claude.ai/analytics/claude-code. Les administrateurs et propriétaires peuvent consulter le tableau de bord.

Le tableau de bord Teams et Enterprise inclut :

- **Métriques d'utilisation** : lignes de code acceptées, taux d'acceptation des suggestions, utilisateurs actifs quotidiens et sessions
- **Métriques de contribution** : PRs et lignes de code livrées avec l'assistance de Claude Code, avec [intégration GitHub](#)
- **Classement** : principaux contributeurs classés par utilisation de Claude Code

- **Export de données** : téléchargez les données de contribution au format CSV pour des rapports personnalisés

Activer les métriques de contribution

Note:

Les métriques de contribution sont en bêta publique et disponibles sur les plans Claude for Teams et Claude for Enterprise. Ces métriques couvrent uniquement les utilisateurs au sein de votre organisation claude.ai. L'utilisation via l'API Claude Console ou les intégrations tierces n'est pas incluse.

Les données d'utilisation et d'adoption sont disponibles pour tous les comptes Claude for Teams et Claude for Enterprise. Les métriques de contribution nécessitent une configuration supplémentaire pour connecter votre organisation GitHub.

Vous devez avoir le rôle Propriétaire pour configurer les paramètres analytiques. Un administrateur GitHub doit installer l'application GitHub.

Warning:

Les métriques de contribution ne sont pas disponibles pour les organisations avec [Zero Data Retention](#) activé. Le tableau de bord analytique affichera uniquement les métriques d'utilisation.

Step 1: Installer l'application GitHub

Un administrateur GitHub installe l'application Claude GitHub sur le compte GitHub de votre organisation à github.com/apps/claude.

Step 2: Activer l'analytique Claude Code

Un propriétaire Claude accède à claude.ai/admin-settings/claude-code et active la fonctionnalité d'analytique Claude Code.

Step 3: Activer l'analytique GitHub

Sur la même page, activez le bouton bascule 'Analytique GitHub'.

Step 4: S'authentifier avec GitHub

Complétez le flux d'authentification GitHub et sélectionnez les organisations GitHub à inclure dans l'analyse.

Les données apparaissent généralement dans les 24 heures suivant l'activation, avec des mises à jour quotidiennes. Si aucune donnée n'apparaît, vous pouvez voir l'un de ces messages :

- « **Application GitHub requise** » : installez l'application GitHub pour consulter les métriques de contribution
- « **Traitement des données en cours** » : revenez dans quelques jours et confirmez que l'application GitHub est installée si les données n'apparaissent pas

Les métriques de contribution prennent en charge GitHub Cloud et GitHub Enterprise Server.

Examiner les métriques récapitulatives

Note:

Ces métriques sont délibérément conservatrices et représentent une sous-estimation de l'impact réel de Claude Code. Seules les lignes et les PRs où il y a une grande confiance dans l'implication de Claude Code sont comptabilisées.

Le tableau de bord affiche ces métriques récapitulatives en haut :

- **PRs avec CC** : nombre total de demandes de fusion qui contiennent au moins une ligne de code écrite avec Claude Code
- **Lignes de code avec CC** : nombre total de lignes de code dans toutes les PRs fusionnées qui ont été écrites avec l'assistance de Claude Code. Seules les « lignes effectives » sont comptabilisées : lignes avec plus de 3 caractères après normalisation, excluant les lignes vides et les lignes contenant uniquement des crochets ou une ponctuation triviale.
- **PRs avec Claude Code (%)** : pourcentage de toutes les PRs fusionnées qui contiennent du code assisté par Claude Code
- **Taux d'acceptation des suggestions** : pourcentage de fois où les utilisateurs acceptent les suggestions d'édition de code de Claude Code, y compris l'utilisation des outils Edit, Write et NotebookEdit
- **Lignes de code acceptées** : nombre total de lignes de code écrites par Claude Code que les utilisateurs ont acceptées dans leurs sessions. Cela exclut les suggestions rejetées et ne suit pas les suppressions ultérieures.

Explorer les graphiques

Le tableau de bord inclut plusieurs graphiques pour visualiser les tendances au fil du temps.

Suivre l'adoption

Le graphique Adoption affiche les tendances d'utilisation quotidiennes :

- **utilisateurs** : utilisateurs actifs quotidiens
- **sessions** : nombre de sessions Claude Code actives par jour

Mesurer les PRs par utilisateur

Ce graphique affiche l'activité des développeurs individuels au fil du temps :

- **PRs par utilisateur** : nombre total de PRs fusionnées par jour divisé par les utilisateurs actifs quotidiens
- **utilisateurs** : utilisateurs actifs quotidiens

Utilisez ceci pour comprendre comment la productivité individuelle change à mesure que l'adoption de Claude Code augmente.

Afficher la répartition des demandes de fusion

Le graphique Demandes de fusion affiche une répartition quotidienne des PRs fusionnées :

- **PRs avec CC** : demandes de fusion contenant du code assisté par Claude Code
- **PRs sans CC** : demandes de fusion sans code assisté par Claude Code

Basculez vers la vue **Lignes de code** pour voir la même répartition par lignes de code plutôt que par nombre de PR.

Trouver les principaux contributeurs

Le classement affiche les 10 meilleurs utilisateurs classés par volume de contribution.

Basculez entre :

- **Demandes de fusion** : affiche les PRs avec Claude Code par rapport à toutes les PRs pour chaque utilisateur
- **Lignes de code** : affiche les lignes avec Claude Code par rapport à toutes les lignes pour chaque utilisateur

Cliquez sur **Exporter tous les utilisateurs** pour télécharger les données de contribution complètes pour tous les utilisateurs au format CSV. L'export inclut tous les utilisateurs, pas seulement les 10 premiers affichés.

Attribution des PR

Lorsque les métriques de contribution sont activées, Claude Code analyse les demandes de fusion fusionnées pour déterminer quel code a été écrit avec l'assistance de Claude Code. Ceci est fait en mettant en correspondance l'activité de session Claude Code par rapport au code dans chaque PR.

Critères de balisage

Les PRs sont balisées comme « avec Claude Code » si elles contiennent au moins une ligne de code écrite lors d'une session Claude Code. Le système utilise une correspondance conservatrice : seul le code où il y a une grande confiance dans l'implication de Claude Code est compté comme assisté.

Processus d'attribution

Lorsqu'une demande de fusion est fusionnée :

1. Les lignes ajoutées sont extraites du diff de la PR
2. Les sessions Claude Code qui ont modifié les fichiers correspondants dans une fenêtre de temps sont identifiées
3. Les lignes de PR sont mises en correspondance avec la sortie de Claude Code en utilisant plusieurs stratégies
4. Les métriques sont calculées pour les lignes assistées par l'IA et le nombre total de lignes

Avant la comparaison, les lignes sont normalisées : les espaces blancs sont supprimés, les espaces multiples sont réduits, les guillemets sont standardisés et le texte est converti en minuscules.

Les demandes de fusion fusionnées contenant des lignes assistées par Claude Code sont étiquetées comme `claude-code-assisted` dans GitHub.

Fenêtre de temps

Les sessions de 21 jours avant à 2 jours après la date de fusion de la PR sont considérées pour la correspondance d'attribution.

Fichiers exclus

Certains fichiers sont automatiquement exclus de l'analyse car ils sont générés automatiquement :

- Fichiers de verrouillage : `package-lock.json`, `yarn.lock`, `Cargo.lock` et similaires
- Code généré : sorties Protobuf, artefacts de construction, fichiers minifiés

- Répertoires de construction : dist/, build/, node_modules/, target/
- Fixtures de test : snapshots, cassettes, données fictives
- Lignes de plus de 1 000 caractères, qui sont probablement minifiées ou générées

Notes d'attribution

Gardez ces détails supplémentaires à l'esprit lors de l'interprétation des données d'attribution :

- Le code considérablement réécrit par les développeurs, avec une différence de plus de 20 %, n'est pas attribué à Claude Code
- Les sessions en dehors de la fenêtre de 21 jours ne sont pas considérées
- L'algorithme ne considère pas la branche source ou destination de la PR lors de l'attribution

Tirer le meilleur parti de l'analytique

Utilisez les métriques de contribution pour démontrer le ROI, identifier les modèles d'adoption et trouver les membres de l'équipe qui peuvent aider les autres à démarrer.

Surveiller l'adoption

Suivez le graphique Adoption et les nombres d'utilisateurs pour identifier :

- Les utilisateurs actifs qui peuvent partager les meilleures pratiques
- Les tendances globales d'adoption dans votre organisation
- Les baisses d'utilisation qui peuvent indiquer des frictions ou des problèmes

Mesurer le ROI

Les métriques de contribution aident à répondre à « Cet outil vaut-il l'investissement ? » avec des données de votre propre base de code :

- Suivez les changements dans les PRs par utilisateur au fil du temps à mesure que l'adoption augmente
- Comparez les PRs et les lignes de code livrées avec et sans Claude Code
- Utilisez aux côtés des [métriques DORA](#), de la vitesse des sprints ou d'autres KPIs d'ingénierie pour comprendre les changements résultant de l'adoption de Claude Code

Identifier les utilisateurs avancés

Le classement vous aide à trouver les membres de l'équipe avec une adoption élevée de Claude Code qui peuvent :

- Partager les techniques de prompting et les flux de travail avec l'équipe

- Fournir des commentaires sur ce qui fonctionne bien
- Aider à intégrer les nouveaux utilisateurs

Accéder aux données par programmation

Pour interroger ces données via GitHub, recherchez les PRs étiquetées avec `claude-code-assisted`.

Accéder à l'analytique pour les clients API

Les clients API utilisant Claude Console peuvent accéder à l'analytique à platform.claude.com/claude-code. Vous devez avoir la permission UsageView pour accéder au tableau de bord, qui est accordée aux rôles Developer, Billing, Admin, Owner et Primary Owner.

Note:

Les métriques de contribution avec intégration GitHub ne sont actuellement pas disponibles pour les clients API. Le tableau de bord Console affiche uniquement les métriques d'utilisation et de dépenses.

Le tableau de bord Console affiche :

- **Lignes de code acceptées** : nombre total de lignes de code écrites par Claude Code que les utilisateurs ont acceptées dans leurs sessions. Cela exclut les suggestions rejetées et ne suit pas les suppressions ultérieures.
- **Taux d'acceptation des suggestions** : pourcentage de fois où les utilisateurs acceptent l'utilisation de l'outil d'édition de code, y compris les outils Edit, Write et NotebookEdit.
- **Activité** : utilisateurs actifs quotidiens et sessions affichés sur un graphique.
- **Dépenses** : coûts quotidiens de l'API en dollars aux côtés du nombre d'utilisateurs.

Afficher les insights d'équipe

Le tableau des insights d'équipe affiche les métriques par utilisateur :

- **Membres** : tous les utilisateurs qui se sont authentifiés à Claude Code. Les utilisateurs de clé API s'affichent par identifiant de clé, les utilisateurs OAuth s'affichent par adresse e-mail.
- **Dépenses ce mois** : coûts totaux de l'API par utilisateur pour le mois en cours.
- **Lignes ce mois** : total par utilisateur des lignes de code acceptées pour le mois en cours.

Note:

Les chiffres de dépenses dans le tableau de bord Console sont des estimations à des fins analytiques. Pour les coûts réels, consultez votre page de facturation.

Ressources connexes

- [Surveillance avec OpenTelemetry](#) : exportez les métriques et événements en temps réel vers votre pile d'observabilité
- [Gérer les coûts efficacement](#) : définissez les limites de dépenses et optimisez l'utilisation des tokens
- [Permissions](#) : configurez les rôles et les permissions

Surveillance

Découvrez comment activer et configurer OpenTelemetry pour Claude Code.

Claude Code prend en charge les métriques et événements OpenTelemetry (OTel) pour la surveillance et l'observabilité.

Toutes les métriques sont des données de séries chronologiques exportées via le protocole de métriques standard d'OpenTelemetry, et les événements sont exportés via le protocole de journaux/événements d'OpenTelemetry. Il est de la responsabilité de l'utilisateur de s'assurer que ses backends de métriques et de journaux sont correctement configurés et que la granularité d'agrégation répond à ses exigences de surveillance.

Démarrage rapide

Configurez OpenTelemetry à l'aide de variables d'environnement :

```
## 1. Activer la télémétrie
export CLAUDE_CODE_ENABLE_TELEMETRY=1

## 2. Choisir les exportateurs (les deux sont facultatifs - configurez uniquement
ce dont vous avez besoin)
export OTEL_METRICS_EXPORTER=otlp      # Options : otlp, prometheus, console
export OTEL_LOGS_EXPORTER=otlp        # Options : otlp, console

## 3. Configurer le point de terminaison OTLP (pour l'exportateur OTLP)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

## 4. Définir l'authentification (si nécessaire)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

## 5. Pour le débogage : réduire les intervalles d'export
export OTEL_METRIC_EXPORT_INTERVAL=10000 # 10 secondes (par défaut : 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000    # 5 secondes (par défaut : 5000ms)

## 6. Exécuter Claude Code
claude
```

Note:

Les intervalles d'export par défaut sont de 60 secondes pour les métriques et de 5 secondes pour les journaux. Lors de la configuration, vous pouvez utiliser des intervalles plus courts à des fins de débogage. N'oubliez pas de les réinitialiser pour une utilisation en production.

Pour les options de configuration complètes, consultez la [spécification OpenTelemetry](#).

Configuration de l'administrateur

Les administrateurs peuvent configurer les paramètres OpenTelemetry pour tous les utilisateurs via le [fichier de paramètres gérés](#). Cela permet un contrôle centralisé des paramètres de télémétrie dans toute une organisation. Consultez la [précédence des paramètres](#) pour plus d'informations sur la façon dont les paramètres sont appliqués.

Exemple de configuration des paramètres gérés :

```
{
  "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:

Les paramètres gérés peuvent être distribués via MDM (Mobile Device Management) ou d'autres solutions de gestion d'appareils. Les variables d'environnement définies dans le fichier de paramètres gérés ont une haute priorité et ne peuvent pas être remplacées par les utilisateurs.

Détails de la configuration

Variables de configuration courantes

Variable d'environnement	Description	Exemples de valeurs
<code>CLAUDE_CODE_ENABLE_TELEMETRY</code>	Active la collecte de télémétrie (obligatoire)	<code>1</code>
<code>OTEL_METRICS_EXPORTER</code>	Type(s) d'exportateur de métriques (séparés par des virgules)	<code>console</code> , <code>otlp</code> , <code>prometheus</code>
<code>OTEL_LOGS_EXPORTER</code>	Type(s) d'exportateur de journaux/événements (séparés par des virgules)	<code>console</code> , <code>otlp</code>
<code>OTEL_EXPORTER_OTLP_PROTOCOL</code>	Protocole pour l'exportateur OTLP (tous les signaux)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_ENDPOINT</code>	Point de terminaison du collecteur OTLP (tous les signaux)	<code>http://localhost:4317</code>

Variable d'environnement	Description	Exemples de valeurs
<code>OTEL_EXPORTER_OTLP_METRICS_PROTOCOL</code>	Protocole pour les métriques (remplace le général)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_METRICS_ENDPOINT</code>	Point de terminaison des métriques OTLP (remplace le général)	<code>http://localhost:4318/v1/metrics</code>
<code>OTEL_EXPORTER_OTLP_LOGS_PROTOCOL</code>	Protocole pour les journaux (remplace le général)	<code>grpc</code> , <code>http/json</code> , <code>http/protobuf</code>
<code>OTEL_EXPORTER_OTLP_LOGS_ENDPOINT</code>	Point de terminaison des journaux OTLP (remplace le général)	<code>http://localhost:4318/v1/logs</code>
<code>OTEL_EXPORTER_OTLP_HEADERS</code>	En-têtes d'authentification pour OTLP	<code>Authorization=Bearer token</code>
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY</code>	Clé client pour l'authentification mTLS	Chemin vers le fichier de clé client
<code>OTEL_EXPORTER_OTLP_METRICS_CLIENT_CERTIFICATE</code>	Certificat client pour l'authentification mTLS	Chemin vers le fichier de certificat client
<code>OTEL_METRIC_EXPORT_INTERVAL</code>	Intervalle d'export en millisecondes (par défaut : 60000)	<code>5000</code> , <code>60000</code>
<code>OTEL_LOGS_EXPORT_INTERVAL</code>	Intervalle d'export des journaux en millisecondes (par défaut : 5000)	<code>1000</code> , <code>10000</code>
<code>OTEL_LOG_USER_PROMPTS</code>	Activer la journalisation du contenu des invites utilisateur (par défaut : désactivé)	<code>1</code> pour activer
<code>CLAUDE_CODE_OTEL_HELPERS_DEBOUNCE_MS</code>	Intervalle d'actualisation des en-têtes dynamiques (par défaut : 1740000ms / 29 minutes)	<code>900000</code>

Contrôle de la cardinalité des métriques

Les variables d'environnement suivantes contrôlent les attributs inclus dans les métriques pour gérer la cardinalité :

Variable d'environnement	Description	Valeur par défaut	Exemple pour désactiver
<code>OTEL_METRICS_INCLUDE_SESSION_ID</code>	Inclure l'attribut session.id dans les métriques	<code>true</code>	<code>false</code>
<code>OTEL_METRICS_INCLUDE_VERSION</code>	Inclure l'attribut app.version dans les métriques	<code>false</code>	<code>true</code>
<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code>	Inclure l'attribut user.account_uuid dans les métriques	<code>true</code>	<code>false</code>

Ces variables aident à contrôler la cardinalité des métriques, ce qui affecte les exigences de stockage et les performances des requêtes dans votre backend de métriques. Une cardinalité plus faible signifie généralement de meilleures performances et des coûts de stockage plus bas, mais des données moins granulaires pour l'analyse.

En-têtes dynamiques

Pour les environnements d'entreprise qui nécessitent une authentification dynamique, vous pouvez configurer un script pour générer des en-têtes dynamiquement :

Configuration des paramètres

Ajoutez à votre `.claude/settings.json` :

```
{
  "otelHeadersHelper": "/bin/generate_opentelemetry_headers.sh"
}
```

Exigences du script

Le script doit générer du JSON valide avec des paires clé-valeur de chaînes représentant les en-têtes HTTP :

```
#!/bin/bash
## Exemple : plusieurs en-têtes
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.sh)\"}"
```

Comportement d'actualisation

Le script d'aide des en-têtes s'exécute au démarrage et périodiquement par la suite pour prendre en charge l'actualisation des jetons. Par défaut, le script s'exécute toutes les 29 minutes. Personnalisez l'intervalle avec la variable d'environnement

`CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS`.

Support des organisations multi-équipes

Les organisations avec plusieurs équipes ou départements peuvent ajouter des attributs personnalisés pour distinguer les différents groupes à l'aide de la variable d'environnement `OTEL_RESOURCE_ATTRIBUTES` :

```
## Ajouter des attributs personnalisés pour l'identification de l'équipe
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_center=eng-123"
```

Ces attributs personnalisés seront inclus dans toutes les métriques et tous les événements, ce qui vous permet de :

- Filtrer les métriques par équipe ou département
- Suivre les coûts par centre de coûts
- Créer des tableaux de bord spécifiques à l'équipe
- Configurer des alertes pour des équipes spécifiques

Warning:

Exigences importantes de formatage pour `OTEL_RESOURCE_ATTRIBUTES` :

La variable d'environnement `OTEL_RESOURCE_ATTRIBUTES` suit la [spécification W3C Baggage](#), qui a des exigences de formatage strictes :

- **Aucun espace autorisé** : Les valeurs ne peuvent pas contenir d'espaces. Par exemple, `user.organizationName=My Company` est invalide
- **Format** : Doit être des paires clé=valeur séparées par des virgules :
`key1=value1,key2=value2`

- **Caractères autorisés** : Uniquement les caractères US-ASCII à l'exclusion des caractères de contrôle, des espaces, des guillemets doubles, des virgules, des points-virgules et des barres obliques inverses
- **Caractères spéciaux** : Les caractères en dehors de la plage autorisée doivent être codés en pourcentage

Exemples :

```
## ❌ Invalide - contient des espaces
export OTEL_RESOURCE_ATTRIBUTES="org.name=John's Organization"

## ✅ Valide - utiliser des traits de soulignement ou camelCase à la place
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"

## ✅ Valide - coder en pourcentage les caractères spéciaux si nécessaire
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

Remarque : entourer les valeurs de guillemets n'échappe pas aux espaces. Par exemple, `org.name="My Company"` donne la valeur littérale `"My Company"` (avec guillemets inclus), pas `My Company`.

Exemples de configurations


```

## Débogage de console (intervalles de 1 seconde)
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

## Plusieurs exportateurs
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json

## Points de terminaison/backends différents pour les métriques et les journaux
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

## Métriques uniquement (pas d'événements/journaux)
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

## Événements/journaux uniquement (pas de métriques)
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
```

Métriques et événements disponibles

Attributs standard

Toutes les métriques et tous les événements partagent ces attributs standard :

Attribut	Description	Contrôlé par
<code>session.id</code>	Identifiant de session unique	<code>OTEL_METRICS_INCLUDE_SESSION_ID</code> (par défaut : <code>true</code>)
<code>app.version</code>	Version actuelle de Claude Code	<code>OTEL_METRICS_INCLUDE_VERSION</code> (par défaut : <code>false</code>)
<code>organization.id</code>	UUID de l'organisation (si authentifié)	Toujours inclus si disponible
<code>user.account.uuid</code>	UUID du compte (si authentifié)	<code>OTEL_METRICS_INCLUDE_ACCOUNT_UUID</code> (par défaut : <code>true</code>)
<code>terminal.type</code>	Type de terminal (par exemple, <code>iTerm.app</code> , <code>vscode</code> , <code>cursor</code> , <code>tmux</code>)	Toujours inclus si détecté

Métriques

Claude Code exporte les métriques suivantes :

Nom de la métrique	Description	Unité
<code>claude_code.session.count</code>	Nombre de sessions CLI démarrées	count
<code>claude_code.lines_of_code.count</code>	Nombre de lignes de code modifiées	count
<code>claude_code.pull_request.count</code>	Nombre de demandes de tirage créées	count

Nom de la métrique	Description	Unité
<code>claude_code.commit.count</code>	Nombre de commits git créés	count
<code>claude_code.cost.usage</code>	Coût de la session Claude Code	USD
<code>claude_code.token.usage</code>	Nombre de jetons utilisés	tokens
<code>claude_code.code_edit_tool.decision</code>	Nombre de décisions de permission de l'outil d'édition de code	count
<code>claude_code.active_time.total</code>	Temps actif total en secondes	s

Détails des métriques

Compteur de sessions

Incrémenté au début de chaque session.

Attributs :

- Tous les [attributs standard](#)

Compteur de lignes de code

Incrémenté lorsque du code est ajouté ou supprimé.

Attributs :

- Tous les [attributs standard](#)
- `type` : (`"added"` , `"removed"`)

Compteur de demandes de tirage

Incrémenté lors de la création de demandes de tirage via Claude Code.

Attributs :

- Tous les [attributs standard](#)

Compteur de commits

Incrémenté lors de la création de commits git via Claude Code.

Attributs :

- Tous les [attributs standard](#)

Compteur de coûts

Incrémenté après chaque demande d'API.

Attributs :

- Tous les [attributs standard](#)
- `model` : Identifiant du modèle (par exemple, "claude-sonnet-4-5-20250929")

Compteur de jetons

Incrémenté après chaque demande d'API.

Attributs :

- Tous les [attributs standard](#)
- `type` : ("input", "output", "cacheRead", "cacheCreation")
- `model` : Identifiant du modèle (par exemple, "claude-sonnet-4-5-20250929")

Compteur de décisions de l'outil d'édition de code

Incrémenté lorsque l'utilisateur accepte ou rejette l'utilisation de l'outil Edit, Write ou NotebookEdit.

Attributs :

- Tous les [attributs standard](#)
- `tool` : Nom de l'outil ("Edit", "Write", "NotebookEdit")
- `decision` : Décision de l'utilisateur ("accept", "reject")
- `language` : Langage de programmation du fichier édité (par exemple, "TypeScript", "Python", "JavaScript", "Markdown"). Retourne "unknown" pour les extensions de fichier non reconnues.

Compteur de temps actif

Suit le temps réel passé à utiliser activement Claude Code (pas le temps d'inactivité). Cette métrique est incrémentée lors des interactions utilisateur telles que la saisie d'invites ou la réception de réponses.

Attributs :

- Tous les [attributs standard](#)

Événements

Claude Code exporte les événements suivants via les journaux/événements OpenTelemetry (lorsque `OTEL_LOGS_EXPORTER` est configuré) :

Événement d'invite utilisateur

Enregistré lorsqu'un utilisateur soumet une invite.

Nom de l'événement : `claude_code.user_prompt`

Attributs :

- Tous les [attributs standard](#)
- `event.name` : `"user_prompt"`
- `event.timestamp` : Horodatage ISO 8601
- `prompt_length` : Longueur de l'invite
- `prompt` : Contenu de l'invite (masqué par défaut, activez avec `OTEL_LOG_USER_PROMPTS=1`)

Événement de résultat d'outil

Enregistré lorsqu'un outil termine son exécution.

Nom de l'événement : `claude_code.tool_result`

Attributs :

- Tous les [attributs standard](#)
- `event.name` : `"tool_result"`
- `event.timestamp` : Horodatage ISO 8601
- `tool_name` : Nom de l'outil
- `success` : `"true"` ou `"false"`
- `duration_ms` : Temps d'exécution en millisecondes
- `error` : Message d'erreur (en cas d'échec)
- `decision` : Soit `"accept"` soit `"reject"`
- `source` : Source de la décision - `"config"`, `"user_permanent"`, `"user_temporary"`, `"user_abort"`, ou `"user_reject"`
- `tool_parameters` : Chaîne JSON contenant les paramètres spécifiques à l'outil (si disponible)
 - Pour l'outil Bash : inclut `bash_command`, `full_command`, `timeout`, `description`, `sandbox`

Événement de demande d'API

Enregistré pour chaque demande d'API à Claude.

Nom de l'événement : `claude_code.api_request`

Attributs :

- Tous les [attributs standard](#)
- `event.name` : `"api_request"`
- `event.timestamp` : Horodatage ISO 8601
- `model` : Modèle utilisé (par exemple, "claude-sonnet-4-5-20250929")
- `cost_usd` : Coût estimé en USD
- `duration_ms` : Durée de la demande en millisecondes
- `input_tokens` : Nombre de jetons d'entrée
- `output_tokens` : Nombre de jetons de sortie
- `cache_read_tokens` : Nombre de jetons lus à partir du cache
- `cache_creation_tokens` : Nombre de jetons utilisés pour la création du cache

Événement d'erreur d'API

Enregistré lorsqu'une demande d'API à Claude échoue.

Nom de l'événement : `claude_code.api_error`

Attributs :

- Tous les [attributs standard](#)
- `event.name` : `"api_error"`
- `event.timestamp` : Horodatage ISO 8601
- `model` : Modèle utilisé (par exemple, "claude-sonnet-4-5-20250929")
- `error` : Message d'erreur
- `status_code` : Code de statut HTTP (si applicable)
- `duration_ms` : Durée de la demande en millisecondes
- `attempt` : Numéro de tentative (pour les demandes réessayées)

Événement de décision d'outil

Enregistré lorsqu'une décision de permission d'outil est prise (accepter/rejeter).

Nom de l'événement : `claude_code.tool_decision`

Attributs :

- Tous les [attributs standard](#)
- `event.name` : "tool_decision"
- `event.timestamp` : Horodatage ISO 8601
- `tool_name` : Nom de l'outil (par exemple, "Read", "Edit", "Write", "NotebookEdit")
- `decision` : Soit "accept" soit "reject"
- `source` : Source de la décision - "config", "user_permanent", "user_temporary", "user_abort", ou "user_reject"

Interprétation des données de métriques et d'événements

Les métriques exportées par Claude Code fournissent des informations précieuses sur les modèles d'utilisation et la productivité. Voici quelques visualisations et analyses courantes que vous pouvez créer :

Surveillance de l'utilisation

Métrique	Opportunité d'analyse
<code>claude_code.token.usage</code>	Ventiler par <code>type</code> (entrée/sortie), utilisateur, équipe ou modèle
<code>claude_code.session.count</code>	Suivre l'adoption et l'engagement au fil du temps
<code>claude_code.lines_of_code.count</code>	Mesurer la productivité en suivant les ajouts/suppressions de code
<code>claude_code.commit.count</code> & <code>claude_code.pull_request.count</code>	Comprendre l'impact sur les flux de travail de développement

Surveillance des coûts

La métrique `claude_code.cost.usage` aide à :

- Suivre les tendances d'utilisation entre les équipes ou les individus
- Identifier les sessions à utilisation élevée pour l'optimisation

Note:

Les métriques de coûts sont des approximations. Pour les données de facturation officielles, consultez votre fournisseur d'API (Claude Console, AWS Bedrock ou Google Cloud Vertex).

Alertes et segmentation

Les alertes courantes à considérer :

- Pics de coûts
- Consommation de jetons inhabituelle
- Volume de session élevé d'utilisateurs spécifiques

Toutes les métriques peuvent être segmentées par `user.account_uuid`, `organization.id`, `session.id`, `model` et `app.version`.

Analyse des événements

Les données d'événements fournissent des informations détaillées sur les interactions de Claude Code :

Modèles d'utilisation des outils : analyser les événements de résultat d'outil pour identifier :

- Les outils les plus fréquemment utilisés
- Les taux de réussite des outils
- Les temps d'exécution moyens des outils
- Les modèles d'erreur par type d'outil

Surveillance des performances : suivre les durées des demandes d'API et les temps d'exécution des outils pour identifier les goulots d'étranglement de performance.

Considérations relatives aux backends

Votre choix de backends de métriques et de journaux détermine les types d'analyses que vous pouvez effectuer :

Pour les métriques

- **Bases de données de séries chronologiques (par exemple, Prometheus)** : Calculs de taux, métriques agrégées
- **Magasins colonnaires (par exemple, ClickHouse)** : Requêtes complexes, analyse d'utilisateurs uniques
- **Plates-formes d'observabilité complètes (par exemple, Honeycomb, Datadog)** : Requêtes avancées, visualisation, alertes

Pour les événements/journaux

- **Systèmes d'agrégation de journaux (par exemple, Elasticsearch, Loki)** : Recherche en texte intégral, analyse de journaux

- **Magasins colonnaires (par exemple, ClickHouse)** : Analyse d'événements structurés
- **Plates-formes d'observabilité complètes (par exemple, Honeycomb, Data-dog)** : Corrélation entre les métriques et les événements

Pour les organisations nécessitant des métriques d'utilisateurs actifs quotidiens/hebdomadaires/mensuels (DAU/WAU/MAU), envisagez des backends qui prennent en charge les requêtes de valeurs uniques efficaces.

Informations sur le service

Toutes les métriques et tous les événements sont exportés avec les attributs de ressource suivants :

- `service.name` : `claude-code`
- `service.version` : Version actuelle de Claude Code
- `os.type` : Type de système d'exploitation (par exemple, `linux`, `darwin`, `windows`)
- `os.version` : Chaîne de version du système d'exploitation
- `host.arch` : Architecture de l'hôte (par exemple, `amd64`, `arm64`)
- `wsl.version` : Numéro de version WSL (présent uniquement lors de l'exécution sur Windows Subsystem for Linux)
- Nom du compteur : `com.anthropic.claude_code`

Ressources de mesure du ROI

Pour un guide complet sur la mesure du retour sur investissement pour Claude Code, y compris la configuration de la télémétrie, l'analyse des coûts, les métriques de productivité et les rapports automatisés, consultez le [Guide de mesure du ROI de Claude Code](#). Ce référentiel fournit des configurations Docker Compose prêtes à l'emploi, des configurations Prometheus et OpenTelemetry, et des modèles pour générer des rapports de productivité intégrés à des outils comme Linear.

Considérations de sécurité/confidentialité

- La télémétrie est opt-in et nécessite une configuration explicite
- Les informations sensibles comme les clés d'API ou le contenu des fichiers ne sont jamais incluses dans les métriques ou les événements
- Le contenu des invites utilisateur est masqué par défaut - seule la longueur de l'invite est enregistrée. Pour activer la journalisation des invites utilisateur, définissez

`OTEL_LOG_USER_PROMPTS=1`

Surveillance de Claude Code sur Amazon Bedrock

Pour des conseils détaillés sur la surveillance de l'utilisation de Claude Code pour Amazon Bedrock, consultez [Implémentation de la surveillance de Claude Code \(Bedrock\)](#).

Aperçu du déploiement en entreprise

Découvrez comment Claude Code peut s'intégrer à divers services tiers et infrastructures pour répondre aux exigences de déploiement en entreprise.

Cette page fournit un aperçu des options de déploiement disponibles et vous aide à choisir la configuration appropriée pour votre organisation.

Comparaison des fournisseurs

Fonctionnalité Anthropic Amazon Bedrock Google Vertex AI Microsoft Foundry

Régions [Pays](#) pris en charge Plusieurs [régions](#) AWS Plusieurs [régions](#) GCP Plusieurs [régions](#) Azure

Mise en cache des invites Activée par défaut Activée par défaut Activée par défaut Activée par défaut

Authentification Clé API Clé API ou identifiants AWS Identifiants GCP Clé API ou Microsoft Entra ID

Suivi des coûts Tableau de bord AWS Cost Explorer Facturation GCP Gestion des coûts Azure

Fonctionnalités d'entreprise Équipes, surveillance de l'utilisation Politiques IAM, CloudTrail Rôles IAM, Journaux d'audit cloud Politiques RBAC, Azure Monitor

Fournisseurs de cloud

- [Amazon Bedrock](#) Utilisez les modèles Claude via l'infrastructure AWS avec authentification par clé API ou basée sur IAM et surveillance native AWS
- [Google Vertex AI](#) Accédez aux modèles Claude via Google Cloud Platform avec sécurité et conformité de niveau entreprise
- [Microsoft Foundry](#) Accédez à Claude via Azure avec authentification par clé API ou Microsoft Entra ID et facturation Azure

Infrastructure d'entreprise

- [Réseau d'entreprise](#) Configurez Claude Code pour fonctionner avec les serveurs proxy de votre organisation et les exigences SSL/TLS

- [Passerelle LLM](#) Déployez un accès centralisé aux modèles avec suivi de l'utilisation, budgétisation et journalisation d'audit

Aperçu de la configuration

Claude Code prend en charge des options de configuration flexibles qui vous permettent de combiner différents fournisseurs et infrastructures :

Note:

Comprenez la différence entre :

- **Proxy d'entreprise** : Un proxy HTTP/HTTPS pour acheminer le trafic (défini via `HTTPS_PROXY` ou `HTTP_PROXY`)
- **Passerelle LLM** : Un service qui gère l'authentification et fournit des points de terminaison compatibles avec le fournisseur (défini via `ANTHROPIC_BASE_URL` , `ANTHROPIC_BEDROCK_BASE_URL` ou `ANTHROPIC_VERTEX_BASE_URL`)

Les deux configurations peuvent être utilisées ensemble.

Utilisation de Bedrock avec proxy d'entreprise

Acheminez le trafic Bedrock via un proxy HTTP/HTTPS d'entreprise :

```
## Activer Bedrock
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1

## Configurer le proxy d'entreprise
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilisation de Bedrock avec passerelle LLM

Utilisez un service de passerelle qui fournit des points de terminaison compatibles avec Bedrock :

```
## Activer Bedrock
export CLAUDE_CODE_USE_BEDROCK=1

## Configurer la passerelle LLM
export ANTHROPIC_BEDROCK_BASE_URL='https://your-llm-gateway.com/bedrock'
export CLAUDE_CODE_SKIP_BEDROCK_AUTH=1
# Si la passerelle gère l'authentification AWS
```

Utilisation de Foundry avec proxy d'entreprise

Acheminez le trafic Azure via un proxy HTTP/HTTPS d'entreprise :

```
## Activer Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1
export ANTHROPIC_FOUNDRY_RESOURCE=your-resource
export ANTHROPIC_FOUNDRY_API_KEY=your-api-key # Ou omettez pour
l'authentification Entra ID

## Configurer le proxy d'entreprise
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilisation de Foundry avec passerelle LLM

Utilisez un service de passerelle qui fournit des points de terminaison compatibles avec Azure :

```
## Activer Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## Configurer la passerelle LLM
export ANTHROPIC_FOUNDRY_BASE_URL='https://your-llm-gateway.com'
export CLAUDE_CODE_SKIP_FOUNDRY_AUTH=1
# Si la passerelle gère l'authentification Azure
```

Utilisation de Vertex AI avec proxy d'entreprise

Acheminez le trafic Vertex AI via un proxy HTTP/HTTPS d'entreprise :

```
## Activer Vertex
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=us-east5
export ANTHROPIC_VERTEX_PROJECT_ID=your-project-id

## Configurer le proxy d'entreprise
export HTTPS_PROXY='https://proxy.example.com:8080'
```

Utilisation de Vertex AI avec passerelle LLM

Combinez les modèles Google Vertex AI avec une passerelle LLM pour une gestion centralisée :

```
## Activer Vertex
export CLAUDE_CODE_USE_VERTEX=1

## Configurer la passerelle LLM
export ANTHROPIC_VERTEX_BASE_URL='https://your-llm-gateway.com/vertex'
export CLAUDE_CODE_SKIP_VERTEX_AUTH=1 # Si la passerelle gère l'authentification
GCP
```

Configuration de l'authentification

Claude Code utilise `ANTHROPIC_AUTH_TOKEN` pour l'en-tête `Authorization` si nécessaire. Les drapeaux `SKIP_AUTH` (`CLAUDE_CODE_SKIP_BEDROCK_AUTH` , `CLAUDE_CODE_SKIP_VERTEX_AUTH`) sont utilisés dans les scénarios de passerelle LLM où la passerelle gère l'authentification du fournisseur.

Choisir la bonne configuration de déploiement

Considérez ces facteurs lors de la sélection de votre approche de déploiement :

Accès direct au fournisseur

Idéal pour les organisations qui :

- Veulent la configuration la plus simple
- Disposent d'une infrastructure AWS ou GCP existante
- Ont besoin de surveillance et de conformité natives du fournisseur

Proxy d'entreprise

Idéal pour les organisations qui :

- Ont des exigences de proxy d'entreprise existantes
- Ont besoin de surveillance du trafic et de conformité
- Doivent acheminer tout le trafic via des chemins réseau spécifiques

Passerelle LLM

Idéal pour les organisations qui :

- Ont besoin de suivi de l'utilisation entre les équipes
- Veulent basculer dynamiquement entre les modèles
- Nécessitent une limitation de débit personnalisée ou des budgets
- Ont besoin d'une gestion centralisée de l'authentification

Débogage

Lors du débogage de votre déploiement :

- Utilisez la `commande slash` `claude /status`. Cette commande fournit une observabilité dans les paramètres d'authentification, de proxy et d'URL appliqués.
- Définissez la variable d'environnement `export ANTHROPIC_LOG=debug` pour enregistrer les requêtes.

Meilleures pratiques pour les organisations

1. Investir dans la documentation et la mémoire

Nous recommandons vivement d'investir dans la documentation afin que Claude Code comprenne votre base de code. Les organisations peuvent déployer des fichiers CLAUDE.md à plusieurs niveaux :

- **À l'échelle de l'organisation** : Déployez dans les répertoires système comme `/Library/Application Support/ClaudeCode/CLAUDE.md` (macOS) pour les normes à l'échelle de l'entreprise
- **Au niveau du référentiel** : Créez des fichiers `CLAUDE.md` dans les racines des référentiels contenant l'architecture du projet, les commandes de compilation et les directives de contribution. Archivez-les dans le contrôle de source afin que tous les utilisateurs en bénéficient

[En savoir plus.](#)

2. Simplifier le déploiement

Si vous avez un environnement de développement personnalisé, nous constatons que créer un moyen « en un clic » d'installer Claude Code est essentiel pour augmenter l'adoption dans une organisation.

3. Commencer par une utilisation guidée

Encouragez les nouveaux utilisateurs à essayer Claude Code pour les questions sur la base de code, ou sur les corrections de bogues plus petites ou les demandes de fonctionnalités. Demandez à Claude Code de faire un plan. Vérifiez les suggestions de Claude et donnez un retour d'information si c'est hors piste. Au fil du temps, à mesure que les utilisateurs comprendront mieux ce nouveau paradigme, ils seront plus efficaces pour laisser Claude Code fonctionner de manière plus agentique.

4. Configurer les politiques de sécurité

Les équipes de sécurité peuvent configurer des autorisations gérées pour ce que Claude Code est et n'est pas autorisé à faire, ce qui ne peut pas être remplacé par la configuration locale. [En savoir plus](#).

5. Exploiter MCP pour les intégrations

MCP est un excellent moyen de donner à Claude Code plus d'informations, comme la connexion à des systèmes de gestion de tickets ou des journaux d'erreurs. Nous recommandons qu'une équipe centrale configure les serveurs MCP et archive une configuration `.mcp.json` dans la base de code afin que tous les utilisateurs en bénéficient. [En savoir plus](#).

Chez Anthropic, nous faisons confiance à Claude Code pour alimenter le développement dans chaque base de code Anthropic. Nous espérons que vous apprécierez d'utiliser Claude Code autant que nous.

Étapes suivantes

- [Configurer Amazon Bedrock](#) pour le déploiement natif AWS
- [Configurer Google Vertex AI](#) pour le déploiement GCP
- [Configurer Microsoft Foundry](#) pour le déploiement Azure
- [Configurer le réseau d'entreprise](#) pour les exigences réseau
- [Déployer la passerelle LLM](#) pour la gestion en entreprise
- [Paramètres](#) pour les options de configuration et les variables d'environnement

Configurer les paramètres gérés par le serveur (bêta publique)

Configurez centralement Claude Code pour votre organisation via des paramètres livrés par le serveur, sans nécessiter d'infrastructure de gestion des appareils.

Les paramètres gérés par le serveur permettent aux administrateurs de configurer centralement Claude Code via une interface web sur Claude.ai. Les clients Claude Code reçoivent automatiquement ces paramètres lorsque les utilisateurs s'authentifient avec leurs identifiants d'organisation.

Cette approche est conçue pour les organisations qui n'ont pas d'infrastructure de gestion des appareils en place, ou qui ont besoin de gérer les paramètres pour les utilisateurs sur des appareils non gérés.

Note:

Les paramètres gérés par le serveur sont en bêta publique et disponibles pour les clients [Claude for Teams](#) et [Claude for Enterprise](#). Les fonctionnalités peuvent évoluer avant la disponibilité générale.

Conditions requises

Pour utiliser les paramètres gérés par le serveur, vous avez besoin de :

- Un plan Claude for Teams ou Claude for Enterprise
- Claude Code version 2.1.38 ou ultérieure pour Claude for Teams, ou version 2.1.30 ou ultérieure pour Claude for Enterprise
- Accès réseau à api.anthropic.com

Choisir entre les paramètres gérés par le serveur et gérés par le point de terminaison

Claude Code prend en charge deux approches pour la configuration centralisée. Les paramètres gérés par le serveur livrent la configuration à partir des serveurs d'Anthropic. Les [paramètres gérés par le point de terminaison](#) sont déployés directement sur les appareils via des stratégies natives du système d'exploitation (préférences gérées macOS, registre Windows) ou des fichiers de paramètres gérés.

Approche	Idéale pour	Modèle de sécurité
Paramètres gérés par le serveur	Organisations sans MDM, ou utilisateurs sur des appareils non gérés	Paramètres livrés à partir des serveurs d'Anthropic au moment de l'authentification
Paramètres gérés par le point de terminaison	Organisations avec MDM ou gestion des points de terminaison	Paramètres déployés sur les appareils via des profils de configuration MDM, des stratégies de registre ou des fichiers de paramètres gérés

Si vos appareils sont inscrits dans une solution MDM ou de gestion des points de terminaison, les paramètres gérés par le point de terminaison offrent des garanties de sécurité plus fortes car le fichier de paramètres peut être protégé contre les modifications de l'utilisateur au niveau du système d'exploitation.

Configurer les paramètres gérés par le serveur

Step 1: Ouvrir la console d'administration

Dans [Claude.ai](#), accédez à **Admin Settings > Claude Code > Managed settings**.

Step 2: Définir vos paramètres

Ajoutez votre configuration en JSON. Tous les [paramètres disponibles dans settings.json](#) sont pris en charge, y compris les [paramètres réservés à la gestion](#) comme `disableBypassPermissionsMode`.

Cet exemple applique une liste de refus de permissions et empêche les utilisateurs de contourner les permissions :

```
{
  "permissions": {
    "deny": [
      "Bash(curl *)",
      "Read(./env)",
      "Read(./env.*)",
      "Read(./secrets/**)"
    ]
  },
  "disableBypassPermissionsMode": "disable"
}
```

Step 3: Enregistrer et déployer

Enregistrez vos modifications. Les clients Claude Code reçoivent les paramètres mis à jour au prochain démarrage ou lors du cycle d'interrogation horaire.

Vérifier la livraison des paramètres

Pour confirmer que les paramètres sont appliqués, demandez à un utilisateur de redémarrer Claude Code. Si la configuration inclut des paramètres qui déclenchent la [boîte de dialogue d'approbation de sécurité](#), l'utilisateur voit une invite décrivant les paramètres gérés au démarrage. Vous pouvez également vérifier que les règles de permission gérées sont actives en demandant à un utilisateur d'exécuter `/permissions` pour afficher ses règles de permission effectives.

Contrôle d'accès

Les rôles suivants peuvent gérer les paramètres gérés par le serveur :

- **Propriétaire principal**
- **Propriétaire**

Limitez l'accès au personnel de confiance, car les modifications de paramètres s'appliquent à tous les utilisateurs de l'organisation.

Limitations actuelles

Les paramètres gérés par le serveur ont les limitations suivantes pendant la période bêta :

- Les paramètres s'appliquent uniformément à tous les utilisateurs de l'organisation. Les configurations par groupe ne sont pas encore prises en charge.

- Les [configurations de serveur MCP](#) ne peuvent pas être distribuées via les paramètres gérés par le serveur.

Livraison des paramètres

Précédence des paramètres

Les paramètres gérés par le serveur et les [paramètres gérés par le point de terminaison](#) occupent tous deux le niveau le plus élevé dans la [hiérarchie des paramètres](#) de Claude Code. Aucun autre niveau de paramètres ne peut les remplacer, y compris les arguments de ligne de commande. Lorsque les deux sont présents, les paramètres gérés par le serveur ont la priorité et les paramètres gérés par le point de terminaison ne sont pas utilisés.

Comportement de récupération et de mise en cache

Claude Code récupère les paramètres à partir des serveurs d'Anthropic au démarrage et interroge les mises à jour toutes les heures pendant les sessions actives.

Premier lancement sans paramètres en cache :

- Claude Code récupère les paramètres de manière asynchrone
- Si la récupération échoue, Claude Code continue sans paramètres gérés
- Il y a une brève fenêtre avant le chargement des paramètres où les restrictions ne sont pas encore appliquées

Lancements ultérieurs avec paramètres en cache :

- Les paramètres en cache s'appliquent immédiatement au démarrage
- Claude Code récupère les paramètres actualisés en arrière-plan
- Les paramètres en cache persistent en cas de défaillance réseau

Claude Code applique les mises à jour des paramètres automatiquement sans redémarrage, sauf pour les paramètres avancés comme la configuration OpenTelemetry, qui nécessitent un redémarrage complet pour prendre effet.

Boîtes de dialogue d'approbation de sécurité

Certains paramètres qui pourraient présenter des risques de sécurité nécessitent une approbation explicite de l'utilisateur avant d'être appliqués :

- **Paramètres de commande shell** : paramètres qui exécutent des commandes shell
- **Variables d'environnement personnalisées** : variables ne figurant pas dans la liste de sécurité connue
- **Configurations de hook** : toute définition de hook

Lorsque ces paramètres sont présents, les utilisateurs voient une boîte de dialogue de sécurité expliquant ce qui est configuré. Les utilisateurs doivent approuver pour continuer. Si un utilisateur rejette les paramètres, Claude Code se ferme.

Note:

En mode non interactif avec l'indicateur `-p`, Claude Code ignore les boîtes de dialogue de sécurité et applique les paramètres sans approbation de l'utilisateur.

Disponibilité de la plateforme

Les paramètres gérés par le serveur nécessitent une connexion directe à `api.anthropic.com` et ne sont pas disponibles lors de l'utilisation de fournisseurs de modèles tiers :

- Amazon Bedrock
- Google Vertex AI
- Microsoft Foundry
- Points de terminaison API personnalisés via `ANTHROPIC_BASE_URL` ou [passerelles LLM](#)

Journalisation d'audit

Les événements du journal d'audit pour les modifications de paramètres sont disponibles via l'API de conformité ou l'export du journal d'audit. Contactez votre équipe de compte Anthropic pour accéder.

Les événements d'audit incluent le type d'action effectuée, le compte et l'appareil qui ont effectué l'action, et les références aux valeurs précédentes et nouvelles.

Considérations de sécurité

Les paramètres gérés par le serveur offrent une application centralisée des stratégies, mais ils fonctionnent comme un contrôle côté client. Sur les appareils non gérés, les utilisateurs ayant un accès administrateur ou sudo peuvent modifier le binaire Claude Code, le système de fichiers ou la configuration réseau.

Scénario	Comportement
L'utilisateur modifie le fichier de paramètres en cache	Le fichier falsifié s'applique au démarrage, mais les paramètres corrects se restaurent lors de la prochaine récupération du serveur

Scénario	Comportement
L'utilisateur supprime le fichier de paramètres en cache	Le comportement du premier lancement se produit : les paramètres sont récupérés de manière asynchrone avec une brève fenêtre non appliquée
L'API est indisponible	Les paramètres en cache s'appliquent s'ils sont disponibles, sinon les paramètres gérés ne sont pas appliqués jusqu'à la prochaine récupération réussie
L'utilisateur s'authentifie avec une organisation différente	Les paramètres ne sont pas livrés pour les comptes en dehors de l'organisation gérée
L'utilisateur définit un <code>ANTHROPIC_BASE_URL</code> non défini par défaut	Les paramètres gérés par le serveur sont contournés lors de l'utilisation de fournisseurs d'API tiers

Pour détecter les modifications de configuration au moment de l'exécution, utilisez les [hooks](#) `ConfigChange` pour enregistrer les modifications ou bloquer les modifications non autorisées avant qu'elles ne prennent effet.

Pour des garanties d'application plus fortes, utilisez les [paramètres gérés par le point de terminaison](#) sur les appareils inscrits dans une solution MDM.

Voir aussi

Pages connexes pour gérer la configuration de Claude Code :

- [Paramètres](#) : référence de configuration complète incluant tous les paramètres disponibles
- [Paramètres gérés par le point de terminaison](#) : paramètres gérés déployés sur les appareils par l'informatique
- [Authentification](#) : configurer l'accès des utilisateurs à Claude Code
- [Sécurité](#) : garanties de sécurité et meilleures pratiques

Part 11: Cloud Providers

Claude Code sur Amazon Bedrock

Découvrez comment configurer Claude Code via Amazon Bedrock, y compris la configuration, la configuration IAM et le dépannage.

Prérequis

Avant de configurer Claude Code avec Bedrock, assurez-vous que vous disposez de :

- Un compte AWS avec accès à Bedrock activé
- Accès aux modèles Claude souhaités (par exemple, Claude Sonnet 4.6) dans Bedrock
- AWS CLI installé et configuré (facultatif - nécessaire uniquement si vous n'avez pas d'autre mécanisme pour obtenir les identifiants)
- Autorisations IAM appropriées

Note:

Si vous déployez Claude Code pour plusieurs utilisateurs, [épinglez vos versions de modèle](#) pour éviter les ruptures lors de la publication de nouveaux modèles par Anthropic.

Configuration

1. Soumettre les détails du cas d'usage

Les utilisateurs pour la première fois des modèles Anthropic doivent soumettre les détails du cas d'usage avant d'invoquer un modèle. Ceci est fait une fois par compte.

1. Assurez-vous que vous disposez des bonnes autorisations IAM (voir plus à ce sujet ci-dessous)
2. Accédez à la [console Amazon Bedrock](#)
3. Sélectionnez **Chat/Text playground**
4. Choisissez n'importe quel modèle Anthropic et vous serez invité à remplir le formulaire de cas d'usage

2. Configurer les identifiants AWS

Claude Code utilise la chaîne d'identifiants par défaut du SDK AWS. Configurez vos identifiants en utilisant l'une de ces méthodes :

Option A : Configuration AWS CLI

```
aws configure
```

Option B : Variables d'environnement (clé d'accès)

```
export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-access-key
export AWS_SESSION_TOKEN=your-session-token
```

Option C : Variables d'environnement (profil SSO)

```
aws sso login --profile=<your-profile-name>

export AWS_PROFILE=your-profile-name
```

Option D : Identifiants de la console de gestion AWS

```
aws login
```

[En savoir plus](#) sur `aws login`.

Option E : Clés API Bedrock

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Les clés API Bedrock offrent une méthode d'authentification plus simple sans avoir besoin d'identifiants AWS complets. [En savoir plus sur les clés API Bedrock](#).

Configuration avancée des identifiants

Claude Code prend en charge l'actualisation automatique des identifiants pour AWS SSO et les fournisseurs d'identité d'entreprise. Ajoutez ces paramètres à votre fichier de paramètres Claude Code (voir [Paramètres](#) pour les emplacements des fichiers).

Lorsque Claude Code détecte que vos identifiants AWS ont expiré (soit localement en fonction de leur horodatage, soit lorsque Bedrock retourne une erreur d'identifiants), il exécutera automatiquement vos commandes `awsAuthRefresh` et/ou `awsCredentialExport` configurées pour obtenir de nouveaux identifiants avant de réessayer la demande.

Exemple de configuration

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

Paramètres de configuration expliqués

awsAuthRefresh : Utilisez ceci pour les commandes qui modifient le répertoire `.aws`, comme la mise à jour des identifiants, du cache SSO ou des fichiers de configuration. La sortie de la commande s'affiche à l'utilisateur, mais l'entrée interactive n'est pas prise en charge. Cela fonctionne bien pour les flux SSO basés sur un navigateur où l'interface de ligne de commande affiche une URL ou un code et vous complétez l'authentification dans le navigateur.

awsCredentialExport : Utilisez ceci uniquement si vous ne pouvez pas modifier `.aws` et devez retourner directement les identifiants. La sortie est capturée silencieusement et non affichée à l'utilisateur. La commande doit générer du JSON dans ce format :

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value"
  }
}
```

3. Configurer Claude Code

Définissez les variables d'environnement suivantes pour activer Bedrock :

```
## Enable Bedrock integration
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1 # or your preferred region

## Optional: Override the region for the small/fast model (Haiku)
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2
```

Lors de l'activation de Bedrock pour Claude Code, gardez à l'esprit les points suivants :

- `AWS_REGION` est une variable d'environnement requise. Claude Code ne lit pas à partir du fichier de configuration `.aws` pour ce paramètre.
- Lors de l'utilisation de Bedrock, les commandes `/login` et `/logout` sont désactivées car l'authentification est gérée via les identifiants AWS.
- Vous pouvez utiliser des fichiers de paramètres pour les variables d'environnement comme `AWS_PROFILE` que vous ne voulez pas divulguer à d'autres processus. Voir [Paramètres](#) pour plus d'informations.

4. Épingler les versions de modèle

Warning:

Épinglez les versions de modèle spécifiques pour chaque déploiement. Si vous utilisez des alias de modèle (`sonnet` , `opus` , `haiku`) sans épinglage, Claude Code peut tenter d'utiliser une version de modèle plus récente qui n'est pas disponible dans votre compte Bedrock, ce qui casse les utilisateurs existants lors de la publication de mises à jour par Anthropic.

Définissez ces variables d'environnement sur des ID de modèle Bedrock spécifiques :

```
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'
```

Ces variables utilisent des ID de profil d'inférence inter-régions (avec le préfixe `us.`). Si vous utilisez un préfixe de région différent ou des profils d'inférence d'application, ajustez en conséquence. Pour les ID de modèle actuels et hérités, voir [Aperçu des modèles](#). Voir [Configuration du modèle](#) pour la liste complète des variables d'environnement.

Claude Code utilise ces modèles par défaut lorsqu'aucune variable d'épinglage n'est définie :

Type de modèle	Valeur par défaut
Modèle principal	<code>global.anthropic.claude-sonnet-4-6</code>
Modèle petit/rapide	<code>us.anthropic.claude-haiku-4-5-20251001-v1:0</code>

Pour personnaliser davantage les modèles, utilisez l’une de ces méthodes :

```
## Using inference profile ID
export ANTHROPIC_MODEL='global.anthropic.claude-sonnet-4-6'
export ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

## Using application inference profile ARN
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
inference-profile/your-model-id'

## Optional: Disable prompt caching if needed
export DISABLE_PROMPT_CACHING=1
```

Note:
[La mise en cache des invites](#) peut ne pas être disponible dans toutes les régions.

Mapper chaque version de modèle à un profil d’inférence

Les variables d’environnement `ANTHROPIC_DEFAULT_*_MODEL` configurent un profil d’inférence par famille de modèles. Si votre organisation doit exposer plusieurs versions de la même famille dans le sélecteur `/model`, chacune acheminée vers son propre ARN de profil d’inférence d’application, utilisez plutôt le paramètre `modelOverrides` dans votre [fichier de paramètres](#).

Cet exemple mappe trois versions d’Opus à des ARN distincts afin que les utilisateurs puissent basculer entre elles sans contourner les profils d’inférence de votre organisation :

```
{
  "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"
  }
}
```

Lorsqu'un utilisateur sélectionne l'une de ces versions dans `/model`, Claude Code appelle Bedrock avec l'ARN mappé. Les versions sans remplacement reviennent à l'ID de modèle Bedrock intégré ou à tout profil d'inférence correspondant découvert au démarrage. Voir [Remplacer les ID de modèle par version](#) pour plus de détails sur la façon dont les remplacements interagissent avec `availableModels` et d'autres paramètres de modèle.

Configuration IAM

Créez une politique IAM avec les autorisations requises pour 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"
        }
      }
    }
  ]
}

```

Pour des autorisations plus restrictives, vous pouvez limiter la ressource à des ARN de profil d'inférence spécifiques.

Pour plus de détails, voir [Documentation IAM Bedrock](#).

Note:

Créez un compte AWS dédié pour Claude Code pour simplifier le suivi des coûts et le contrôle d'accès.

Garde-fous AWS

Les [garde-fous Amazon Bedrock](#) vous permettent de mettre en œuvre le filtrage du contenu pour Claude Code. Créez un garde-fou dans la [console Amazon Bedrock](#), publiez une version, puis ajoutez les en-têtes du garde-fou à votre [fichier de paramètres](#). Activez l'inférence inter-régions sur votre garde-fou si vous utilisez des profils d'inférence inter-régions.

Exemple de configuration :

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

Dépannage

Si vous rencontrez des problèmes de région :

- Vérifiez la disponibilité du modèle : `aws bedrock list-inference-profiles --region your-region`
- Basculez vers une région prise en charge : `export AWS_REGION=us-east-1`
- Envisagez d'utiliser des profils d'inférence pour l'accès inter-régions

Si vous recevez une erreur « on-demand throughput isn't supported » :

- Spécifiez le modèle comme ID de [profil d'inférence](#)

Claude Code utilise l'API Bedrock [Invoke](#) et ne prend pas en charge l'API Converse.

Ressources supplémentaires

- [Documentation Bedrock](#)
- [Tarification Bedrock](#)

- [Profils d'inférence Bedrock](#)
- [Claude Code sur Amazon Bedrock : Guide de configuration rapide](#)
- [Implémentation de la surveillance de Claude Code \(Bedrock\)](#)

Claude Code sur Google Vertex AI

Découvrez comment configurer Claude Code via Google Vertex AI, y compris la configuration, la configuration IAM et le débogage.

Prérequis

Avant de configurer Claude Code avec Vertex AI, assurez-vous que vous disposez de :

- Un compte Google Cloud Platform (GCP) avec facturation activée
- Un projet GCP avec l'API Vertex AI activée
- Accès aux modèles Claude souhaités (par exemple, Claude Sonnet 4.5)
- Google Cloud SDK (`gcloud`) installé et configuré
- Quota alloué dans la région GCP souhaitée

Configuration de la région

Claude Code peut être utilisé avec les points de terminaison Vertex AI [globaux](#) et régionaux.

Note:

Vertex AI peut ne pas supporter les modèles par défaut de Claude Code dans toutes les régions. Vous devrez peut-être basculer vers une [région ou un modèle supporté](#).

Note:

Vertex AI peut ne pas supporter les modèles par défaut de Claude Code sur les points de terminaison globaux. Vous devrez peut-être basculer vers un point de terminaison régional ou un [modèle supporté](#).

Configuration

1. Activer l'API Vertex AI

Activez l'API Vertex AI dans votre projet GCP :


```
## Définissez votre ID de projet
gcloud config set project YOUR-PROJECT-ID

## Activez l'API Vertex AI
gcloud services enable aiplatform.googleapis.com
```

2. Demander l'accès au modèle

Demandez l'accès aux modèles Claude dans Vertex AI :

1. Accédez au [Vertex AI Model Garden](#)
2. Recherchez les modèles « Claude »
3. Demandez l'accès aux modèles Claude souhaités (par exemple, Claude Sonnet 4.5)
4. Attendez l'approbation (peut prendre 24 à 48 heures)

3. Configurer les identifiants GCP

Claude Code utilise l'authentification Google Cloud standard.

Pour plus d'informations, consultez la [documentation d'authentification Google Cloud](#).

Note:

Lors de l'authentification, Claude Code utilisera automatiquement l'ID de projet de la variable d'environnement `ANTHROPIC_VERTEX_PROJECT_ID` . Pour remplacer cela, définissez l'une de ces variables d'environnement : `GCPLOUD_PROJECT` , `GOOGLE_CLOUD_PROJECT` , ou `GOOGLE_APPLICATION_CREDENTIALS` .

4. Configurer Claude Code

Définissez les variables d'environnement suivantes :

```
## Activez l'intégration Vertex AI
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

## Optionnel : Désactivez la mise en cache des invites si nécessaire
export DISABLE_PROMPT_CACHING=1

## Quand CLOUD_ML_REGION=global, remplacez la région pour les modèles non
supportés
export VERTEX_REGION_CLAUDE_3_5_HAIKU=us-east5

## Optionnel : Remplacez les régions pour d'autres modèles spécifiques
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:

La [mise en cache des invites](#) est automatiquement supportée lorsque vous spécifiez l'indicateur éphémère `cache_control`. Pour la désactiver, définissez `DISABLE_PROMPT_CACHING=1`. Pour des limites de débit accrues, contactez le support Google Cloud.

Note:

Lors de l'utilisation de Vertex AI, les commandes `/login` et `/logout` sont désactivées puisque l'authentification est gérée via les identifiants Google Cloud.

5. Configuration du modèle

Claude Code utilise ces modèles par défaut pour Vertex AI :

Type de modèle	Valeur par défaut
Modèle principal	claude-sonnet-4-5@20250929
Modèle petit/rapide	claude-haiku-4-5@20251001

Note:

Pour les utilisateurs de Vertex AI, Claude Code ne sera pas automatiquement mis à niveau de Haiku 3.5 vers Haiku 4.5. Pour basculer manuellement vers un modèle Haiku plus récent, définissez la variable d'environnement `ANTHROPIC_DEFAULT_HAIKU_MODEL` avec le nom complet du modèle (par exemple, `claude-haiku-4-5@20251001`).

Pour personnaliser les modèles :

```
export ANTHROPIC_MODEL='claude-opus-4-1@20250805'  
export ANTHROPIC_SMALL_FAST_MODEL='claude-haiku-4-5@20251001'
```

Configuration IAM

Attribuez les autorisations IAM requises :

Le rôle `roles/aiplatform.user` inclut les autorisations requises :

- `aiplatform.endpoints.predict` - Requis pour l'invocation de modèle et le comptage de jetons

Pour des autorisations plus restrictives, créez un rôle personnalisé avec uniquement les autorisations ci-dessus.

Pour plus de détails, consultez la [documentation IAM de Vertex](#).

Note:

Nous recommandons de créer un projet GCP dédié pour Claude Code afin de simplifier le suivi des coûts et le contrôle d'accès.

Fenêtre de contexte de 1 million de jetons

Claude Sonnet 4 et Sonnet 4.5 supportent la [fenêtre de contexte de 1 million de jetons](#) sur Vertex AI.

Note:

La fenêtre de contexte de 1 million de jetons est actuellement en version bêta. Pour utiliser la fenêtre de contexte étendue, incluez l'en-tête bêta `context-1m-2025-08-07` dans vos demandes Vertex AI.

Dépannage

Si vous rencontrez des problèmes de quota :

- Vérifiez les quotas actuels ou demandez une augmentation de quota via la [Console Cloud](#)

Si vous rencontrez des erreurs « modèle non trouvé » 404 :

- Confirmez que le modèle est activé dans [Model Garden](#)
- Vérifiez que vous avez accès à la région spécifiée
- Si vous utilisez `CLOUD_ML_REGION=global`, vérifiez que vos modèles supportent les points de terminaison globaux dans [Model Garden](#) sous « Fonctionnalités supportées ». Pour les modèles qui ne supportent pas les points de terminaison globaux, soit :
 - Spécifiez un modèle supporté via `ANTHROPIC_MODEL` ou `ANTHROPIC_SMALL_FAST_MODEL`, soit
 - Définissez un point de terminaison régional en utilisant les variables d'environnement `VERTEX_REGION_<MODEL_NAME>`

Si vous rencontrez des erreurs 429 :

- Pour les points de terminaison régionaux, assurez-vous que le modèle principal et le modèle petit/rapide sont supportés dans votre région sélectionnée
- Envisagez de basculer vers `CLOUD_ML_REGION=global` pour une meilleure disponibilité

Ressources supplémentaires

- [Documentation Vertex AI](#)
- [Tarification Vertex AI](#)
- [Quotas et limites Vertex AI](#)

Claude Code sur Microsoft Foundry

Découvrez comment configurer Claude Code via Microsoft Foundry, y compris la configuration, les paramètres et la résolution des problèmes.

Conditions préalables

Avant de configurer Claude Code avec Microsoft Foundry, assurez-vous que vous disposez de :

- Un abonnement Azure avec accès à Microsoft Foundry
- Des autorisations RBAC pour créer des ressources et des déploiements Microsoft Foundry
- Azure CLI installé et configuré (facultatif - nécessaire uniquement si vous n'avez pas d'autre mécanisme pour obtenir les identifiants)

Configuration

1. Provisionner la ressource Microsoft Foundry

Tout d'abord, créez une ressource Claude dans Azure :

1. Accédez au [portail Microsoft Foundry](#)
2. Créez une nouvelle ressource, en notant le nom de votre ressource
3. Créez des déploiements pour les modèles Claude :
 - Claude Opus
 - Claude Sonnet
 - Claude Haiku

2. Configurer les identifiants Azure

Claude Code prend en charge deux méthodes d'authentification pour Microsoft Foundry. Choisissez la méthode qui correspond le mieux à vos exigences de sécurité.

Option A : Authentification par clé API

1. Accédez à votre ressource dans le portail Microsoft Foundry
2. Allez à la section **Points de terminaison et clés**
3. Copiez la **Clé API**

4. Définissez la variable d'environnement :

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

Option B : Authentification Microsoft Entra ID

Lorsque `ANTHROPIC_FOUNDRY_API_KEY` n'est pas défini, Claude Code utilise automatiquement la [chaîne d'identifiants par défaut](#) du SDK Azure. Cela prend en charge une variété de méthodes pour authentifier les charges de travail locales et distantes.

Dans les environnements locaux, vous pouvez généralement utiliser Azure CLI :

```
az login
```

Note:

Lors de l'utilisation de Microsoft Foundry, les commandes `/login` et `/logout` sont désactivées car l'authentification est gérée via les identifiants Azure.

3. Configurer Claude Code

Définissez les variables d'environnement suivantes pour activer Microsoft Foundry. Notez que les noms de vos déploiements sont définis comme identifiants de modèle dans Claude Code (peut être facultatif si vous utilisez les noms de déploiement suggérés).

```
## Activer l'intégration Microsoft Foundry
export CLAUDE_CODE_USE_FOUNDRY=1

## Nom de la ressource Azure (remplacez {resource} par le nom de votre ressource)
export ANTHROPIC_FOUNDRY_RESOURCE={resource}

## Ou fournissez l'URL de base complète :
## export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com

## Définissez les modèles sur les noms de déploiement de votre ressource
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'
```

Pour plus de détails sur les options de configuration des modèles, consultez [Configuration des modèles](#).

Configuration Azure RBAC

Les rôles par défaut **Azure AI User** et **Cognitive Services User** incluent toutes les autorisations requises pour invoquer les modèles Claude.

Pour des autorisations plus restrictives, créez un rôle personnalisé avec les éléments suivants :

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

Pour plus de détails, consultez la [documentation RBAC de Microsoft Foundry](#).

Résolution des problèmes

Si vous recevez une erreur « Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed » :

- Configurez Entra ID sur l'environnement, ou définissez **ANTHROPIC_FOUNDRY_API_KEY**.

Ressources supplémentaires

- [Documentation Microsoft Foundry](#)
- [Modèles Microsoft Foundry](#)
- [Tarification Microsoft Foundry](#)

Configuration de la passerelle LLM

Découvrez comment configurer Claude Code pour fonctionner avec des solutions de passerelle LLM. Couvre les exigences de la passerelle, la configuration de l'authentification, la sélection du modèle et la configuration des points de terminaison spécifiques aux fournisseurs.

Les passerelles LLM fournissent une couche proxy centralisée entre Claude Code et les fournisseurs de modèles, offrant souvent :

- **Authentification centralisée** - Point unique pour la gestion des clés API
- **Suivi de l'utilisation** - Surveiller l'utilisation entre les équipes et les projets
- **Contrôles des coûts** - Implémenter des budgets et des limites de débit
- **Journalisation d'audit** - Suivre toutes les interactions de modèle pour la conformité
- **Routage des modèles** - Basculer entre les fournisseurs sans modifications de code

Exigences de la passerelle

Pour qu'une passerelle LLM fonctionne avec Claude Code, elle doit répondre aux exigences suivantes :

Format API

La passerelle doit exposer aux clients au moins l'un des formats API suivants :

1. **Anthropic Messages**: `/v1/messages` , `/v1/messages/count_tokens`
 - Doit transférer les en-têtes de requête : `anthropic-beta` , `anthropic-version`
2. **Bedrock InvokeModel**: `/invoke` , `/invoke-with-response-stream`
 - Doit préserver les champs du corps de la requête : `anthropic_beta` , `anthropic_version`
3. **Vertex rawPredict**: `:rawPredict` , `:streamRawPredict` , `/count-tokens:rawPredict`
 - Doit transférer les en-têtes de requête : `anthropic-beta` , `anthropic-version`

L'absence de transfert d'en-têtes ou la non-préservation des champs du corps peut entraîner une réduction des fonctionnalités ou l'impossibilité d'utiliser les fonctionnalités de Claude Code.

Note:

Claude Code détermine les fonctionnalités à activer en fonction du format API. Lors de l'utilisation du format Anthropic Messages avec Bedrock ou Vertex, vous devrez peut-être définir la variable d'environnement `CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS=1`.

Configuration

Sélection du modèle

Par défaut, Claude Code utilisera les noms de modèles standard pour le format API sélectionné.

Si vous avez configuré des noms de modèles personnalisés dans votre passerelle, utilisez les variables d'environnement documentées dans [Configuration du modèle](#) pour correspondre à vos noms personnalisés.

Configuration de LiteLLM

Note:

LiteLLM est un service proxy tiers. Anthropic n'approuve pas, ne maintient pas et n'audite pas la sécurité ou les fonctionnalités de LiteLLM. Ce guide est fourni à titre informatif et peut devenir obsolète. À utiliser à votre discrétion.

Conditions préalables

- Claude Code mis à jour vers la dernière version
- Serveur proxy LiteLLM déployé et accessible
- Accès aux modèles Claude via votre fournisseur choisi

Configuration de base de LiteLLM

Configurer Claude Code :

Méthodes d'authentification

Clé API statique

Méthode la plus simple utilisant une clé API fixe :

```
## Définir dans l'environnement
export ANTHROPIC_AUTH_TOKEN=sk-litellm-static-key

## Ou dans les paramètres de Claude Code
{
  "env": {
    "ANTHROPIC_AUTH_TOKEN": "sk-litellm-static-key"
  }
}
```

Cette valeur sera envoyée en tant qu'en-tête **Authorization**.

Clé API dynamique avec assistant

Pour les clés rotatives ou l'authentification par utilisateur :

1. Créez un script d'assistant de clé API :

```
#!/bin/bash
## ~/bin/get-litellm-key.sh

## Exemple : Récupérer la clé du coffre-fort
vault kv get -field=api_key secret/litellm/claude-code

## Exemple : Générer un jeton JWT
jwt encode \
  --secret="${JWT_SECRET}" \
  --exp="+1h" \
  '{"user":"'${USER}'","team":"engineering"}'
```

1. Configurez les paramètres de Claude Code pour utiliser l'assistant :

```
{
  "apiKeyHelper": "~/bin/get-litellm-key.sh"
}
```

1. Définissez l'intervalle d'actualisation du jeton :

```
## Actualiser toutes les heures (3600000 ms)
export CLAUDE_CODE_API_KEY_HELPER_TTL_MS=3600000
```

Cette valeur sera envoyée en tant qu'en-têtes `Authorization` et `X-API-Key`.
L' `apiKeyHelper` a une priorité inférieure à `ANTHROPIC_AUTH_TOKEN` ou `ANTHROPIC_API_KEY`.

Point de terminaison unifié (recommandé)

Utilisant le [point de terminaison au format Anthropic](#) de LiteLLM :

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000
```

Avantages du point de terminaison unifié par rapport aux points de terminaison de transmission directe :

- Équilibrage de charge
- Basculements
- Support cohérent du suivi des coûts et du suivi des utilisateurs finaux

Points de terminaison de transmission directe spécifiques aux fournisseurs (alternative)

API Claude via LiteLLM

Utilisant le [point de terminaison de transmission directe](#) :

```
export ANTHROPIC_BASE_URL=https://litellm-server:4000/anthropic
```

Amazon Bedrock via LiteLLM

Utilisant le [point de terminaison de transmission directe](#) :

```
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 via LiteLLM

Utilisant le [point de terminaison de transmission directe](#) :

```
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
```

Pour plus d'informations détaillées, consultez la [documentation de LiteLLM](#).

Ressources supplémentaires

- [Documentation de LiteLLM](#)
- [Paramètres de Claude Code](#)
- [Configuration du réseau d'entreprise](#)
- [Aperçu des intégrations tierces](#)

Configuration réseau d'entreprise

Configurez Claude Code pour les environnements d'entreprise avec des serveurs proxy, des autorités de certification (CA) personnalisées et l'authentification mutuelle Transport Layer Security (mTLS).

Claude Code prend en charge diverses configurations réseau et de sécurité d'entreprise via des variables d'environnement. Cela inclut le routage du trafic via des serveurs proxy d'entreprise, la confiance envers des autorités de certification (CA) personnalisées et l'authentification avec des certificats Transport Layer Security (mTLS) mutuels pour une sécurité renforcée.

Note:

Toutes les variables d'environnement affichées sur cette page peuvent également être configurées dans `settings.json`.

Configuration du proxy

Variables d'environnement

Claude Code respecte les variables d'environnement proxy standard :

```
## Proxy HTTPS (recommandé)
export HTTPS_PROXY=https://proxy.example.com:8080

## Proxy HTTP (si HTTPS non disponible)
export HTTP_PROXY=http://proxy.example.com:8080

## Contourner le proxy pour des requêtes spécifiques - format séparé par des
espaces
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
## Contourner le proxy pour des requêtes spécifiques - format séparé par des
virgules
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
## Contourner le proxy pour toutes les requêtes
export NO_PROXY="*"
```

Note:

Claude Code ne prend pas en charge les proxies SOCKS.

Authentification de base

Si votre proxy nécessite une authentification de base, incluez les identifiants dans l'URL du proxy :

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

Warning:

Évitez de coder en dur les mots de passe dans les scripts. Utilisez plutôt des variables d'environnement ou un stockage sécurisé des identifiants.

Tip:

Pour les proxies nécessitant une authentification avancée (NTLM, Kerberos, etc.), envisagez d'utiliser un service LLM Gateway qui prend en charge votre méthode d'authentification.

Certificats CA personnalisés

Si votre environnement d'entreprise utilise des CA personnalisées pour les connexions HTTPS (que ce soit via un proxy ou un accès direct à l'API), configurez Claude Code pour les approuver :

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

Authentification mTLS

Pour les environnements d'entreprise nécessitant une authentification par certificat client :

```
## Certificat client pour l'authentification
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

## Clé privée du client
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

## Optionnel : Phrase de passe pour la clé privée chiffrée
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

Exigences d'accès réseau

Claude Code nécessite l'accès aux URL suivantes :

- api.anthropic.com : Points de terminaison de l'API Claude
- claude.ai : authentification pour les comptes claude.ai
- platform.claude.com : authentification pour les comptes Anthropic Console

Assurez-vous que ces URL sont autorisées dans votre configuration proxy et vos règles de pare-feu. Ceci est particulièrement important lors de l'utilisation de Claude Code dans des environnements réseau conteneurisés ou restreints.

Ressources supplémentaires

- [Paramètres Claude Code](#)
- [Référence des variables d'environnement](#)
- [Guide de dépannage](#)

Part 12: Environment Setup

Conteneurs de développement

Découvrez le conteneur de développement Claude Code pour les équipes qui ont besoin d'environnements cohérents et sécurisés.

La [configuration devcontainer](#) de référence et le [Dockerfile](#) associé offrent un conteneur de développement préconfiguré que vous pouvez utiliser tel quel ou personnaliser selon vos besoins. Ce devcontainer fonctionne avec l'extension [Dev Containers](#) de Visual Studio Code et des outils similaires.

Les mesures de sécurité renforcées du conteneur (isolation et règles de pare-feu) vous permettent d'exécuter `claude --dangerously-skip-permissions` pour contourner les invites de permission pour un fonctionnement sans surveillance.

Warning:

Bien que le devcontainer offre des protections substantielles, aucun système n'est complètement immunisé contre toutes les attaques. Lorsqu'il est exécuté avec `--dangerously-skip-permissions`, les devcontainers n'empêchent pas un projet malveillant d'exfiltrer quoi que ce soit d'accessible dans le devcontainer, y compris les identifiants Claude Code. Nous recommandons d'utiliser les devcontainers uniquement lors du développement avec des référentiels de confiance. Maintenez toujours de bonnes pratiques de sécurité et surveillez les activités de Claude.

Caractéristiques principales

- **Node.js prêt pour la production** : Basé sur Node.js 20 avec les dépendances de développement essentielles
- **Sécurité par conception** : Pare-feu personnalisé limitant l'accès réseau aux seuls services nécessaires
- **Outils conviviaux pour les développeurs** : Inclut git, ZSH avec améliorations de productivité, fzf, et plus
- **Intégration transparente de VS Code** : Extensions préconfigurées et paramètres optimisés

- **Persistance de session** : Préserve l'historique des commandes et les configurations entre les redémarrages du conteneur
- **Fonctionne partout** : Compatible avec les environnements de développement macOS, Windows et Linux

Démarrage en 4 étapes

1. Installez VS Code et l'extension Remote - Containers
2. Clonez le référentiel de l'[implémentation de référence Claude Code](#)
3. Ouvrez le référentiel dans VS Code
4. Lorsque vous y êtes invité, cliquez sur « Rouvrir dans le conteneur » (ou utilisez la Palette de commandes : Cmd+Shift+P → « Remote-Containers: Reopen in Container »)

Analyse de la configuration

La configuration devcontainer se compose de trois composants principaux :

- **[devcontainer.json](#)** : Contrôle les paramètres du conteneur, les extensions et les montages de volume
- **[Dockerfile](#)** : Définit l'image du conteneur et les outils installés
- **[init-firewall.sh](#)** : Établit les règles de sécurité réseau

Fonctionnalités de sécurité

Le conteneur implémente une approche de sécurité multicouche avec sa configuration de pare-feu :

- **Contrôle d'accès précis** : Limite les connexions sortantes aux domaines autorisés uniquement (registre npm, GitHub, API Claude, etc.)
- **Connexions sortantes autorisées** : Le pare-feu autorise les connexions DNS et SSH sortantes
- **Politique de refus par défaut** : Bloque tous les autres accès réseau externes
- **Vérification au démarrage** : Valide les règles de pare-feu lors de l'initialisation du conteneur
- **Isolation** : Crée un environnement de développement sécurisé séparé de votre système principal

Options de personnalisation

La configuration devcontainer est conçue pour être adaptable à vos besoins :

- Ajoutez ou supprimez des extensions VS Code en fonction de votre flux de travail
- Modifiez les allocations de ressources pour différents environnements matériels
- Ajustez les permissions d'accès réseau
- Personnalisez les configurations de shell et les outils de développement

Exemples de cas d'utilisation

Travail client sécurisé

Utilisez les devcontainers pour isoler différents projets clients, en veillant à ce que le code et les identifiants ne se mélangent jamais entre les environnements.

Intégration d'équipe

Les nouveaux membres de l'équipe peuvent obtenir un environnement de développement entièrement configuré en quelques minutes, avec tous les outils et paramètres nécessaires préinstallés.

Environnements CI/CD cohérents

Reflétez votre configuration devcontainer dans les pipelines CI/CD pour assurer que les environnements de développement et de production correspondent.

Ressources connexes

- [Documentation VS Code devcontainers](#)
- [Meilleures pratiques de sécurité Claude Code](#)
- [Configuration réseau d'entreprise](#)

Optimisez votre configuration de terminal

Claude Code fonctionne mieux lorsque votre terminal est correctement configuré. Suivez ces directives pour optimiser votre expérience.

Thèmes et apparence

Claude ne peut pas contrôler le thème de votre terminal. C'est géré par votre application de terminal. Vous pouvez faire correspondre le thème de Claude Code à votre terminal à tout moment via la commande `/config`.

Pour une personnalisation supplémentaire de l'interface Claude Code elle-même, vous pouvez configurer une [ligne d'état personnalisée](#) pour afficher des informations contextuelles comme le modèle actuel, le répertoire de travail ou la branche git en bas de votre terminal.

Sauts de ligne

Vous avez plusieurs options pour entrer des sauts de ligne dans Claude Code :

- **Échappement rapide** : Tapez `\` suivi d'Entrée pour créer une nouvelle ligne
- **Maj+Entrée** : Fonctionne directement dans iTerm2, WezTerm, Ghostty et Kitty
- **Raccourci clavier** : Configurez une liaison de clavier pour insérer une nouvelle ligne dans d'autres terminaux

Configurez Maj+Entrée pour d'autres terminaux

Exécutez `/terminal-setup` dans Claude Code pour configurer automatiquement Maj+Entrée pour VS Code, Alacritty, Zed et Warp.

Note:

La commande `/terminal-setup` n'est visible que dans les terminaux qui nécessitent une configuration manuelle. Si vous utilisez iTerm2, WezTerm, Ghostty ou Kitty, vous ne verrez pas cette commande car Maj+Entrée fonctionne déjà nativement.

Configurez Option+Entrée (VS Code, iTerm2 ou macOS Terminal.app)

Pour Mac Terminal.app :

1. Ouvrez Paramètres → Profils → Clavier

2. Cochez « Utiliser Option comme touche Meta »

Pour le terminal iTerm2 et VS Code :

1. Ouvrez Paramètres → Profils → Touches
2. Sous Général, définissez la touche Option gauche/droite sur « Esc+ »

Configuration des notifications

Ne manquez jamais quand Claude termine une tâche avec une configuration de notification appropriée :

Notifications système iTerm 2

Pour les alertes iTerm 2 lorsque les tâches se terminent :

1. Ouvrez les Préférences iTerm 2
2. Accédez à Profils → Terminal
3. Activez « Silence bell » et Filtrer les alertes → « Envoyer les alertes générées par la séquence d'échappement »
4. Définissez votre délai de notification préféré

Notez que ces notifications sont spécifiques à iTerm 2 et ne sont pas disponibles dans le Terminal macOS par défaut.

Hooks de notification personnalisés

Pour une gestion avancée des notifications, vous pouvez créer des [hooks de notification](#) pour exécuter votre propre logique.

Gestion des entrées volumineuses

Lorsque vous travaillez avec du code étendu ou des instructions longues :

- **Évitez le collage direct** : Claude Code peut avoir du mal avec du contenu très long collé
- **Utilisez des flux basés sur des fichiers** : Écrivez le contenu dans un fichier et demandez à Claude de le lire
- **Soyez conscient des limitations de VS Code** : Le terminal VS Code est particulièrement sujet à la troncature des longs collages

Mode Vim

Claude Code supporte un sous-ensemble de liaisons de clavier Vim qui peut être activé avec `/vim` ou configuré via `/config`.

Le sous-ensemble supporté inclut :

- Changement de mode : **Esc** (vers NORMAL), **i / I**, **a / A**, **o / O** (vers INSERT)
- Navigation : **h / j / k / l**, **w / e / b**, **0 / \$ / ^**, **gg / G**, **f / F / t / T** avec répétition **;** / **,**
- Édition : **x**, **dw / de / db / dd / D**, **cw / ce / cb / cc / C**, **.** (répétition)
- Copie/collage : **yy / Y**, **yw / ye / yb**, **p / P**
- Objets texte : **iw / aw**, **iW / aW**, **i" / a"**, **i' / a'**, **i(/ a(**, **i[/ a[**, **i{ / a{**
- Indentation : **>>** / **<<**
- Opérations de ligne : **J** (fusionner les lignes)

Consultez [Mode interactif](#) pour la référence complète.

Personnalisez votre barre de statut

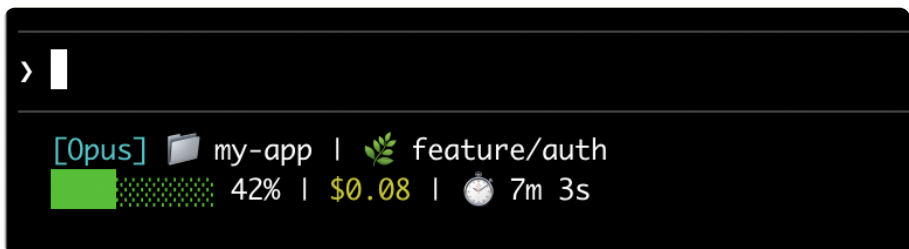
Configurez une barre de statut personnalisée pour surveiller l'utilisation de la fenêtre de contexte, les coûts et l'état git dans Claude Code

La barre de statut est une barre personnalisable en bas de Claude Code qui exécute n'importe quel script shell que vous configurez. Elle reçoit les données de session JSON sur stdin et affiche tout ce que votre script imprime, vous donnant une vue persistante et en un coup d'œil de l'utilisation du contexte, des coûts, de l'état git, ou de tout ce que vous voulez suivre.

Les barres de statut sont utiles quand vous :

- Voulez surveiller l'utilisation de la fenêtre de contexte pendant que vous travaillez
- Avez besoin de suivre les coûts de session
- Travaillez sur plusieurs sessions et avez besoin de les distinguer
- Voulez que la branche git et l'état soient toujours visibles

Voici un exemple d'une [barre de statut multi-lignes](#) qui affiche les informations git sur la première ligne et une barre de contexte codée par couleur sur la deuxième.



Une barre de statut multi-lignes affichant le nom du modèle, le répertoire, la branche git sur la première ligne, et une barre de progression d'utilisation du contexte avec le coût et la durée sur la deuxième ligne

Cette page vous guide à travers [la configuration d'une barre de statut basique](#), explique [comment les données circulent](#) de Claude Code vers votre script, liste [tous les champs que vous pouvez afficher](#), et fournit [des exemples prêts à l'emploi](#) pour les modèles courants comme l'état git, le suivi des coûts et les barres de progression.

Configurer une barre de statut

Utilisez la [commande `/statusline`](#) pour que Claude Code génère un script pour vous, ou [créez manuellement un script](#) et ajoutez-le à vos paramètres.

Utiliser la commande `/statusline`

La commande `/statusline` accepte des instructions en langage naturel décrivant ce que vous voulez afficher. Claude Code génère un fichier script dans `~/.claude/` et met à jour vos paramètres automatiquement :

```
/statusline show model name and context percentage with a progress bar
```

Configurer manuellement une barre de statut

Ajoutez un champ `statusLine` à vos paramètres utilisateur (`~/.claude/settings.json` , où `~` est votre répertoire personnel) ou [paramètres de projet](#). Définissez `type` sur `"command"` et pointez `command` vers un chemin de script ou une commande shell en ligne. Pour une procédure complète de création d'un script, voir [Construire une barre de statut étape par étape](#).

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

Le champ `command` s'exécute dans un shell, vous pouvez donc aussi utiliser des commandes en ligne au lieu d'un fichier script. Cet exemple utilise `jq` pour analyser l'entrée JSON et afficher le nom du modèle et le pourcentage de contexte :

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\[\\(.model.display_name)] \\(.context_window.used_percent
age // 0)% context\''
  }
}
```

Le champ optionnel `padding` ajoute un espacement horizontal supplémentaire (en caractères) au contenu de la barre de statut. Par défaut `0`. Cet espacement s'ajoute à l'espacement intégré de l'interface, il contrôle donc l'indentation relative plutôt que la distance absolue du bord du terminal.

Désactiver la barre de statut

Exécutez `/statusline` et demandez-lui de supprimer ou d'effacer votre barre de statut (par exemple, `/statusline delete`, `/statusline clear`, `/statusline remove it`). Vous pouvez aussi supprimer manuellement le champ `statusLine` de votre `settings.json`.

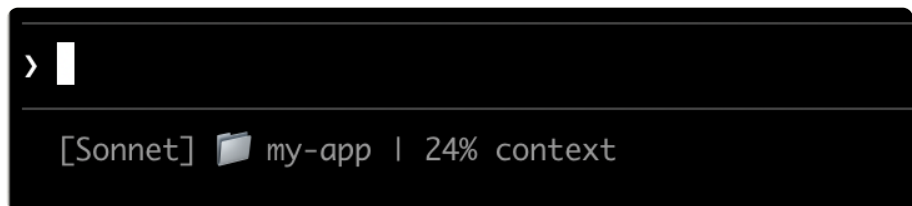
Construire une barre de statut étape par étape

Cette procédure montre ce qui se passe sous le capot en créant manuellement une barre de statut qui affiche le modèle actuel, le répertoire de travail et le pourcentage d'utilisation de la fenêtre de contexte.

Note:

L'exécution de `/statusline` avec une description de ce que vous voulez configurer tout cela automatiquement pour vous.

Ces exemples utilisent des scripts Bash, qui fonctionnent sur macOS et Linux. Sur Windows, voir [Configuration Windows](#) pour des exemples PowerShell et Git Bash.



Une barre de statut affichant le nom du modèle, le répertoire et le pourcentage de contexte

Step 1: Créer un script qui lit JSON et imprime la sortie

Claude Code envoie les données JSON à votre script via stdin. Ce script utilise `jq`, un analyseur JSON en ligne de commande que vous devrez peut-être installer, pour extraire le nom du modèle, le répertoire et le pourcentage de contexte, puis imprime une ligne formattée.

Enregistrez ceci dans `~/.claude/statusline.sh` (où `~` est votre répertoire personnel, tel que `/Users/username` sur macOS ou `/home/username` sur 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: Le rendre exécutable

Marquez le script comme exécutable pour que votre shell puisse l'exécuter :

```
chmod +x ~/.claude/statusline.sh
```

Step 3: Ajouter aux paramètres

Dites à Claude Code d'exécuter votre script comme barre de statut. Ajoutez cette configuration à `~/.claude/settings.json`, qui définit `type` sur `"command"` (ce qui signifie « exécuter cette commande shell ») et pointe `command` vers votre script :

```
{
  "statusLine": {
    "type": "command",
    "command": "~/ .claude/statusline.sh"
  }
}
```

Votre barre de statut apparaît en bas de l'interface. Les paramètres se rechargent automatiquement, mais les modifications n'apparaîtront pas avant votre prochaine interaction avec Claude Code.

Comment fonctionnent les barres de statut

Claude Code exécute votre script et envoie les [données de session JSON](#) via stdin. Votre script lit le JSON, extrait ce dont il a besoin et imprime du texte sur stdout. Claude Code affiche tout ce que votre script imprime.

Quand elle se met à jour

Votre script s'exécute après chaque nouveau message d'assistant, quand le mode de permission change, ou quand le mode vim bascule. Les mises à jour sont débogées à 300 ms, ce qui signifie que les changements rapides se regroupent et votre script s'exécute une fois que les choses se stabilisent. Si une nouvelle mise à jour se déclenche pendant que votre script s'exécute toujours, l'exécution en cours est annulée. Si vous modifiez votre script, les modifications n'apparaîtront pas avant que votre prochaine interaction avec Claude Code ne déclenche une mise à jour.

Ce que votre script peut afficher

- **Plusieurs lignes** : chaque instruction `echo` ou `print` s'affiche comme une ligne séparée. Voir l'[exemple multi-lignes](#).
- **Couleurs** : utilisez les [codes d'échappement ANSI](#) comme `\033[32m` pour le vert (le terminal doit les supporter). Voir l'[exemple d'état git](#).
- **Liens** : utilisez les [séquences d'échappement OSC 8](#) pour rendre le texte cliquable (Cmd+clic sur macOS, Ctrl+clic sur Windows/Linux). Nécessite un terminal qui supporte les hyperliens comme iTerm2, Kitty ou WezTerm. Voir l'[exemple de liens cliquables](#).

Note:

La barre de statut s'exécute localement et ne consomme pas de jetons API. Elle se cache temporairement pendant certaines interactions UI, y compris les suggestions d'autocomplétion, le menu d'aide et les invites de permission.

Données disponibles

Claude Code envoie les champs JSON suivants à votre script via stdin :

Champ	Description
<code>model.id</code> , <code>model.display_name</code>	Identifiant du modèle actuel et nom d’affichage
<code>cwd</code> , <code>workspace.current_dir</code>	Répertoire de travail actuel. Les deux champs contiennent la même valeur ; <code>workspace.current_dir</code> est préféré pour la cohérence avec <code>workspace.project_dir</code> .
<code>workspace.project_dir</code>	Répertoire où Claude Code a été lancé, qui peut différer de <code>cwd</code> si le répertoire de travail change pendant une session
<code>cost.total_cost_usd</code>	Coût total de la session en USD
<code>cost.total_duration_ms</code>	Temps écoulé total depuis le début de la session, en millisecondes
<code>cost.total_api_duration_ms</code>	Temps total passé à attendre les réponses API en millisecondes
<code>cost.total_lines_added</code> , <code>cost.total_lines_removed</code>	Lignes de code modifiées
<code>context_window.total_input_tokens</code> , <code>context_window.total_output_tokens</code>	Comptages de jetons cumulatifs dans la session
<code>context_window.context_window_size</code>	Taille maximale de la fenêtre de contexte en jetons. 200 000 par défaut, ou 1 000 000 pour les modèles avec contexte étendu.
<code>context_window.used_percentage</code>	Pourcentage pré-calculé de fenêtre de contexte utilisée
<code>context_window.remaining_percentage</code>	Pourcentage pré-calculé de fenêtre de contexte restante

Champ	Description
<code>context_window.current_usage</code>	Comptages de jetons du dernier appel API, décrits dans champs de fenêtre de contexte
<code>exceeds_200k_tokens</code>	Si le comptage total de jetons (jetons d'entrée, de cache et de sortie combinés) de la réponse API la plus récente dépasse 200 k. C'est un seuil fixe indépendamment de la taille réelle de la fenêtre de contexte.
<code>session_id</code>	Identifiant de session unique
<code>transcript_path</code>	Chemin vers le fichier de transcription de conversation
<code>version</code>	Version de Claude Code
<code>output_style.name</code>	Nom du style de sortie actuel
<code>vim.mode</code>	Mode vim actuel (<code>NORMAL</code> ou <code>INSERT</code>) quand le mode vim est activé
<code>agent.name</code>	Nom de l'agent lors de l'exécution avec l'indicateur <code>--agent</code> ou les paramètres d'agent configurés
<code>worktree.name</code>	Nom du worktree actif. Présent uniquement pendant les sessions <code>--worktree</code>
<code>worktree.path</code>	Chemin absolu vers le répertoire du worktree
<code>worktree.branch</code>	Nom de la branche git pour le worktree (par exemple, <code>"worktree-my-feature"</code>). Absent pour les worktrees basés sur des hooks
<code>worktree.original_cwd</code>	Le répertoire dans lequel Claude se trouvait avant d'entrer dans le worktree
<code>worktree.original_branch</code>	Branche git extraite avant d'entrer dans le worktree. Absent pour les worktrees basés sur des hooks

Votre commande de barre de statut reçoit cette structure JSON via 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"
  }
}

```

Champs qui peuvent être absents (non présents dans JSON) :

- **vim** : apparaît uniquement quand le mode vim est activé
- **agent** : apparaît uniquement lors de l'exécution avec l'indicateur `--agent` ou les paramètres d'agent configurés
- **worktree** : apparaît uniquement pendant les sessions `--worktree`. Quand présent, **branch** et **original_branch** peuvent aussi être absents pour les worktrees basés sur des hooks

Champs qui peuvent être `null` :

- `context_window.current_usage` : `null` avant le premier appel API dans une session
- `context_window.used_percentage`, `context_window.remaining_percentage` : peuvent être `null` au début de la session

Gérez les champs manquants avec un accès conditionnel et les valeurs null avec des valeurs par défaut de secours dans vos scripts.

Champs de fenêtre de contexte

L'objet `context_window` fournit deux façons de suivre l'utilisation du contexte :

- **Totaux cumulatifs** (`total_input_tokens`, `total_output_tokens`) : somme de tous les jetons dans toute la session, utile pour suivre la consommation totale
- **Utilisation actuelle** (`current_usage`) : comptages de jetons du dernier appel API, utilisez ceci pour un pourcentage de contexte précis car il reflète l'état réel du contexte

L'objet `current_usage` contient :

- `input_tokens` : jetons d'entrée dans le contexte actuel
- `output_tokens` : jetons de sortie générés
- `cache_creation_input_tokens` : jetons écrits dans le cache
- `cache_read_input_tokens` : jetons lus du cache

Le champ `used_percentage` est calculé à partir des jetons d'entrée uniquement : `input_tokens + cache_creation_input_tokens + cache_read_input_tokens`. Il n'inclut pas `output_tokens`.

Si vous calculez le pourcentage de contexte manuellement à partir de `current_usage`, utilisez la même formule d'entrée uniquement pour correspondre à `used_percentage`.

L'objet `current_usage` est `null` avant le premier appel API dans une session.

Exemples

Ces exemples montrent les modèles courants de barre de statut. Pour utiliser n'importe quel exemple :

1. Enregistrez le script dans un fichier comme `~/.claude/statusline.sh` (ou `.py` / `.js`)
2. Le rendre exécutable : `chmod +x ~/.claude/statusline.sh`
3. Ajouter le chemin à vos [paramètres](#)

Les exemples Bash utilisent `jq` pour analyser JSON. Python et Node.js ont l'analyse JSON intégrée.

Utilisation de la fenêtre de contexte

Affiche le modèle actuel et l'utilisation de la fenêtre de contexte avec une barre de progression visuelle. Chaque script lit JSON depuis stdin, extrait le champ `used_percentage` et construit une barre de 10 caractères où les blocs remplis (█) représentent l'utilisation :



Une barre de statut affichant le nom du modèle et une barre de progression avec pourcentage

```
#!/bin/bash
## Read all of stdin into a variable
input=$(cat)

## Extract fields with jq, "// 0" provides fallback for null
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

## Build progress bar: printf creates spaces, tr replaces with blocks
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""
[ "$FILLED" -gt 0 ] && BAR=$(printf "%${FILLED}s" | tr ' ' '█')
[ "$EMPTY" -gt 0 ] && BAR="${BAR}${(printf "%${EMPTY}s" | tr ' ' '░')}"

echo "[ $MODEL ] $BAR $PCT%"
```



```
#!/usr/bin/env python3
import json, sys

## json.load reads and parses stdin in one step
data = json.load(sys.stdin)
model = data['model']['display_name']
## "or 0" handles null values
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

## String multiplication builds the bar
filled = pct * 10 // 100
bar = '█' * filled + '░' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

```
#!/usr/bin/env node
// Node.js reads stdin asynchronously with events
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  // Optional chaining (?.) safely handles null fields
  const pct = Math.floor(data.context_window?.used_percentage || 0);

  // String.repeat() builds the bar
  const filled = Math.floor(pct * 10 / 100);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

  console.log(`[${model}] ${bar} ${pct}%`);
});
```

État git avec couleurs

Affiche la branche git avec des indicateurs codés par couleur pour les fichiers en attente et modifiés. Ce script utilise les [codes d'échappement ANSI](#) pour les couleurs de terminal :

\033[32m est vert, \033[33m est jaune, et \033[0m réinitialise à la valeur par défaut.



[Sonnet]  my-app |  feature/auth +3~2

Une barre de statut affichant le modèle, le répertoire, la branche git et des indicateurs colorés pour les fichiers en attente et modifiés

Chaque script vérifie si le répertoire actuel est un dépôt git, compte les fichiers en attente et modifiés, et affiche des indicateurs codés par couleur :

```
#!/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;

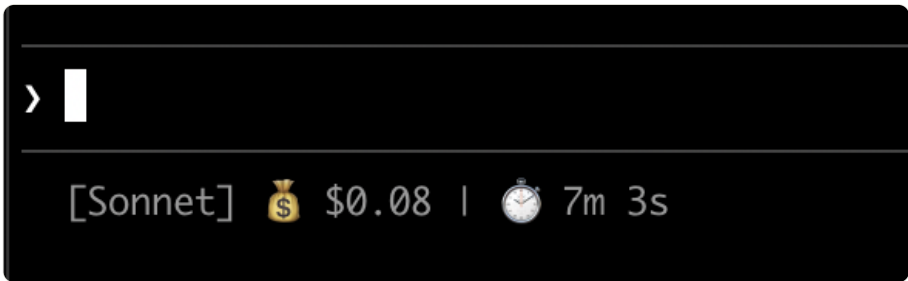
    let gitStatus = staged ? `${GREEN}+${staged}${RESET}` : '';
    gitStatus += modified ? `${YELLOW}~${modified}${RESET}` : '';

    console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} ${gitStatus}`);
  } catch {
    console.log(`[${model}] 📁 ${dir}`);
  }
});
```

Suivi des coûts et de la durée

Suivez les coûts API de votre session et le temps écoulé. Le champ `cost.total_cost_usd` accumule le coût de tous les appels API dans la session actuelle. Le champ `cost.total_duration_ms` mesure le temps écoulé total depuis le début de la session, tandis que `cost.total_api_duration_ms` suit uniquement le temps passé à attendre les réponses API.

Chaque script formate le coût en devise et convertit les millisecondes en minutes et secondes :



Une barre de statut affichant le nom du modèle, le coût de session et la durée

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

COST_FMT=$(printf '%.2f' "$COST")
DURATION_SEC=$((DURATION_MS / 1000))
MINS=$((DURATION_SEC / 60))
SECS=$((DURATION_SEC % 60))

echo "[$MODEL] 💰 $COST_FMT | 🕒 ${MINS}m ${SECS}s"
```

```
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

duration_sec = duration_ms // 1000
mins, secs = duration_sec // 60, duration_sec % 60

print(f"[{model}] 💰 ${cost:.2f} | ⌚ {mins}m {secs}s")
```

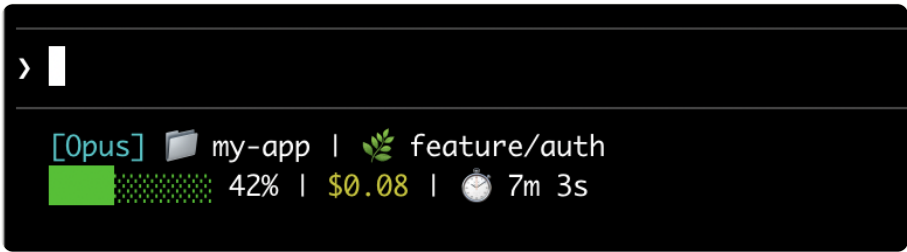
```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const cost = data.cost?.total_cost_usd || 0;
  const durationMs = data.cost?.total_duration_ms || 0;

  const durationSec = Math.floor(durationMs / 1000);
  const mins = Math.floor(durationSec / 60);
  const secs = durationSec % 60;

  console.log(`[${model}] 💰 $$${cost.toFixed(2)} | ⌚ ${mins}m ${secs}s`);
});
```

Afficher plusieurs lignes

Votre script peut afficher plusieurs lignes pour créer un affichage plus riche. Chaque instruction `echo` produit une ligne séparée dans la zone de statut.



Une barre de statut multi-lignes affichant le nom du modèle, le répertoire, la branche git sur la première ligne, et une barre de progression d'utilisation du contexte avec le coût et la durée sur la deuxième ligne

Cet exemple combine plusieurs techniques : couleurs basées sur des seuils (vert sous 70 %, jaune 70-89 %, rouge 90 %+), une barre de progression et des informations de branche git. Chaque instruction `print` ou `echo` crée une ligne séparée :

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[
0m'

## Pick bar color based on context usage
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
BAR=$(printf "%${FILLED}s" | tr ' ' '█')$(printf "%${EMPTY}s" | tr ' ' '░')

MINS=$((DURATION_MS / 60000)); SECS=$((DURATION_MS % 60000 / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | 🌿 $(git branch --show-
current 2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} 📁 ${DIR##*/}${BRANCH}"
COST_FMT=$(printf '=%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⌚ $
{MINS}m ${SECS}s"
```



```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m',
'\033[0m'

bar_color = RED if pct ≥ 90 else YELLOW if pct ≥ 70 else GREEN
filled = pct // 10
bar = '█' * filled + '░' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=True,
stderr=subprocess.DEVNULL).strip()
    branch = f" | 🌿 {branch}" if branch else ""
except:
    branch = ""

print(f"{CYAN}[{model}]{RESET} 📁 {directory}{branch}")
print(f"{bar_color}{bar}{RESET} {pct}% | {YELLOW}${cost:.2f}{RESET} | 🕒 {mins}m
{secs}s")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);
  const cost = data.cost?.total_cost_usd || 0;
  const pct = Math.floor(data.context_window?.used_percentage || 0);
  const durationMs = data.cost?.total_duration_ms || 0;

  const CYAN = '\x1b[36m', GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RED = '\x1b[31m', RESET = '\x1b[0m';

  const barColor = pct ≥ 90 ? RED : pct ≥ 70 ? YELLOW : GREEN;
  const filled = Math.floor(pct / 10);
  const bar = '█'.repeat(filled) + '░'.repeat(10 - filled);

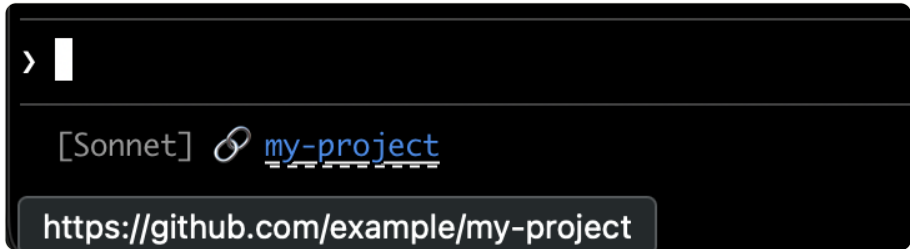
  const mins = Math.floor(durationMs / 60000);
  const secs = Math.floor((durationMs % 60000) / 1000);

  let branch = '';
  try {
    branch = execSync('git branch --show-current', { encoding: 'utf8', stdio:
['pipe', 'pipe', 'ignore'] }).trim();
    branch = branch ? ` | 🌿 ${branch}` : '';
  } catch {}

  console.log(`${CYAN}${model}${RESET} 📁 ${dir}${branch}`);
  console.log(`${barColor}${bar}${RESET} ${pct}% | ${YELLOW}${cost.toFixed(2)}$
${RESET} | 🕒 ${mins}m ${secs}s`);
});
```

Liens cliquables

Cet exemple crée un lien cliquable vers votre dépôt GitHub. Il lit l'URL du dépôt distant, convertit le format SSH en HTTPS avec `sed` et enveloppe le nom du dépôt dans les codes d'échappement OSC 8. Maintenez Cmd (macOS) ou Ctrl (Windows/Linux) et cliquez pour ouvrir le lien dans votre navigateur.



Une barre de statut affichant un lien cliquable vers un dépôt GitHub

Chaque script obtient l'URL du dépôt distant, convertit le format SSH en HTTPS et enveloppe le nom du dépôt dans les codes d'échappement OSC 8. La version Bash utilise `printf '%b'` qui interprète les échappements de barre oblique inverse de manière plus fiable que `echo -e` sur différents shells :

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

## Convert git SSH URL to HTTPS
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/github.com\/' | sed 's\/.git$\/')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 format: \e]8;;URL\a then TEXT then \e]8;;\a
    # printf %b interprets escape sequences reliably across shells
    printf '%b' "[$MODEL]  \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[$MODEL]"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

## Get git remote URL
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # Convert SSH to HTTPS format
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 escape sequences
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] link")
except:
    print(f"[{model}]")
```

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;

  try {
    let remote = execSync('git remote get-url origin', { encoding: 'utf8', stdio: ['pipe', 'pipe', 'ignore'] }).trim();
    // Convert SSH to HTTPS format
    remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replace(/\.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}]`);
  }
});
```

Mettre en cache les opérations coûteuses

Votre script de barre de statut s'exécute fréquemment pendant les sessions actives. Les commandes comme `git status` ou `git diff` peuvent être lentes, surtout dans les grands dépôts. Cet exemple met en cache les informations git dans un fichier temporaire et ne les actualise que toutes les 5 secondes.

Utilisez un nom de fichier stable et fixe pour le fichier de cache comme `/tmp/statusline-git-cache`. Chaque invocation de barre de statut s'exécute en tant que nouveau processus, donc les identifiants basés sur les processus comme `$$`, `os.getpid()` ou `process.pid` produisent une valeur différente à chaque fois et le cache n'est jamais réutilisé.

Chaque script vérifie si le fichier de cache est manquant ou plus ancien que 5 secondes avant d'exécuter les commandes 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 "$CACHE_FILE" 2>/dev/null || echo 0))) -gt $CACHE_MAX_AGE )
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[$MODEL] 📁 ${DIR##*/} | 🌿 $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[$MODEL] 📁 ${DIR##*/}"
fi
```

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

CACHE_FILE = "/tmp/statusline-git-cache"
CACHE_MAX_AGE = 5 # seconds

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],
text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--numstat'],
text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'],
text=True).strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
```

```
print(f"[{model}] 📁 {directory} | 🌿 {branch} +{staged} ~{modified}")  
else:  
    print(f"[{model}] 📁 {directory}")
```



```
#!/usr/bin/env node
const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
  const data = JSON.parse(input);
  const model = data.model.display_name;
  const dir = path.basename(data.workspace.current_dir);

  const CACHE_FILE = '/tmp/statusline-git-cache';
  const CACHE_MAX_AGE = 5; // seconds

  const cacheIsStale = () => {
    if (!fs.existsSync(CACHE_FILE)) return true;
    return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 >
    CACHE_MAX_AGE;
  };

  if (cacheIsStale()) {
    try {
      execSync('git rev-parse --git-dir', { stdio: 'ignore' });
      const branch = execSync('git branch --show-current', { encoding: 'utf8'
' }).trim();
      const staged = execSync('git diff --cached --numstat', { encoding: 'ut
f8' }).trim().split('\n').filter(Boolean).length;
      const modified = execSync('git diff --numstat', { encoding:
'utf8' }).trim().split('\n').filter(Boolean).length;
      fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
    } catch {
      fs.writeFileSync(CACHE_FILE, '||');
    }
  }

  const [branch, staged, modified] = fs.readFileSync(CACHE_FILE,
'utf8').trim().split('|');
```

```

    if (branch) {
        console.log(`[${model}] 📁 ${dir} | 🌿 ${branch} +${staged} ~${modified}
`);
    } else {
        console.log(`[${model}] 📁 ${dir}`);
    }
}
});

```

Configuration Windows

Sur Windows, Claude Code exécute les commandes de barre de statut via Git Bash. Vous pouvez invoquer PowerShell à partir de ce 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]"
}

```

Ou exécutez un script Bash directement :

```
{
  "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]"
```

Conseils

- **Tester avec une entrée fictive** : `echo '{"model": {"display_name": "0pus"}, "context_window": {"used_percentage": 25}}' | ./statusline.sh`
- **Garder la sortie courte** : la barre de statut a une largeur limitée, donc une sortie longue peut être tronquée ou s'enrouler maladroitement
- **Mettre en cache les opérations lentes** : votre script s'exécute fréquemment pendant les sessions actives, donc les commandes comme `git status` peuvent causer des ralentissements. Voir l'[exemple de mise en cache](#) pour savoir comment gérer cela.

Les projets communautaires comme [cstatusline](#) et [starship-claude](#) fournissent des configurations pré-construites avec des thèmes et des fonctionnalités supplémentaires.

Dépannage

La barre de statut n'apparaît pas

- Vérifiez que votre script est exécutable : `chmod +x ~/.claude/statusline.sh`
- Vérifiez que votre script affiche sur stdout, pas stderr
- Exécutez votre script manuellement pour vérifier qu'il produit une sortie
- Si `disableAllHooks` est défini sur `true` dans vos paramètres, la barre de statut est également désactivée. Supprimez ce paramètre ou définissez-le sur `false` pour le ré-activer.

- Exécutez `claude --debug` pour enregistrer le code de sortie et stderr de la première invocation de barre de statut dans une session
- Demandez à Claude de lire votre fichier de paramètres et d'exécuter la commande `statusLine` directement pour faire surface aux erreurs

La barre de statut affiche `--` ou des valeurs vides

- Les champs peuvent être `null` avant la fin de la première réponse API
- Gérez les valeurs null dans votre script avec des valeurs par défaut de secours telles que `// 0` dans jq
- Redémarrez Claude Code si les valeurs restent vides après plusieurs messages

Le pourcentage de contexte affiche des valeurs inattendues

- Utilisez `used_percentage` pour un état de contexte précis plutôt que les totaux cumulatifs
- Les `total_input_tokens` et `total_output_tokens` sont cumulatifs dans la session et peuvent dépasser la taille de la fenêtre de contexte
- Le pourcentage de contexte peut différer de la sortie `/context` en raison du moment où chacun est calculé

Les liens OSC 8 ne sont pas cliquables

- Vérifiez que votre terminal supporte les hyperliens OSC 8 (iTerm2, Kitty, WezTerm)
- Terminal.app ne supporte pas les liens cliquables
- Les sessions SSH et tmux peuvent supprimer les séquences OSC selon la configuration
- Si les séquences d'échappement apparaissent comme du texte littéral comme `\e]8;;`, utilisez `printf '%b'` au lieu de `echo -e` pour une gestion plus fiable des échappements

Problèmes d'affichage avec les séquences d'échappement

- Les séquences d'échappement complexes (couleurs ANSI, liens OSC 8) peuvent occasionnellement causer une sortie corrompue si elles chevauchent d'autres mises à jour UI
- Si vous voyez du texte corrompu, essayez de simplifier votre script en sortie en texte brut
- Les barres de statut multi-lignes avec codes d'échappement sont plus sujettes aux problèmes de rendu que le texte brut sur une seule ligne

Erreurs de script ou blocages

- Les scripts qui se terminent avec des codes non nuls ou ne produisent aucune sortie font que la barre de statut devient vide
- Les scripts lents bloquent la barre de statut de la mise à jour jusqu'à ce qu'ils se terminent. Gardez les scripts rapides pour éviter une sortie obsolète.
- Si une nouvelle mise à jour se déclenche pendant qu'un script lent s'exécute, le script en cours est annulé
- Testez votre script indépendamment avec une entrée fictive avant de le configurer

Les notifications partagent la ligne de la barre de statut

- Les notifications système comme les erreurs du serveur MCP, les mises à jour automatiques et les avertissements de jetons s'affichent sur le côté droit de la même ligne que votre barre de statut
- L'activation du mode verbeux ajoute un compteur de jetons à cette zone
- Sur les terminaux étroits, ces notifications peuvent tronquer votre sortie de barre de statut

Part 13: Troubleshooting & Changelog

Dépannage

Découvrez les solutions aux problèmes courants liés à l'installation et l'utilisation de Claude Code.

Dépanner les problèmes d'installation

Tip:

Si vous préférez éviter le terminal, l'[application Claude Code Desktop](#) vous permet d'installer et d'utiliser Claude Code via une interface graphique. Téléchargez-la pour [macOS](#) ou [Windows](#) et commencez à coder sans aucune configuration en ligne de commande.

Trouvez le message d'erreur ou le symptôme que vous rencontrez :

Ce que vous voyez	Solution
<code>command not found: claude</code> ou <code>'claude' is not recognized</code>	Corriger votre PATH
<code>syntax error near unexpected token '<'</code>	Le script d'installation retourne du HTML
<code>curl: (56) Failure writing output to destination</code>	Télécharger le script d'abord, puis l'exécuter
<code>Killed</code> pendant l'installation sur Linux	Ajouter de l'espace d'échange pour les serveurs à faible mémoire
<code>TLS connect error</code> ou <code>SSL/TLS secure channel</code>	Mettre à jour les certificats CA
<code>Failed to fetch version</code> ou impossible d'atteindre le serveur de téléchargement	Vérifier la connectivité réseau et les paramètres proxy

Ce que vous voyez	Solution
<code>irm is not recognized</code> ou <code>&& is not valid</code>	Utiliser la bonne commande pour votre shell
<code>Claude Code on Windows requires git-bash</code>	Installer ou configurer Git Bash
<code>Error loading shared library</code>	Mauvaise variante binaire pour votre système
<code>Illegal instruction</code> sur Linux	Incompatibilité d'architecture
<code>dyld: cannot load</code> ou <code>Abort trap</code> sur macOS	Incompatibilité binaire
<code>Invoke-Expression: Missing argument in parameter list</code>	Le script d'installation retourne du HTML
<code>App unavailable in region</code>	Claude Code n'est pas disponible dans votre pays. Voir pa ys supportés .
<code>unable to get local issuer certificate</code>	Configurer les certificats CA d'entreprise
<code>OAuth error</code> ou <code>403 Forbidden</code>	Corriger l'authentification

Si votre problème n'est pas listé, suivez ces étapes de diagnostic.

Déboguer les problèmes d'installation

Vérifier la connectivité réseau

L'installateur télécharge depuis `storage.googleapis.com`. Vérifiez que vous pouvez l'atteindre :

```
curl -sI https://storage.googleapis.com
```

Si cela échoue, votre réseau bloque peut-être la connexion. Les causes courantes incluent :

- Les pare-feu d'entreprise ou les proxies bloquant Google Cloud Storage
- Les restrictions réseau régionales : essayez un VPN ou un réseau alternatif
- Les problèmes TLS/SSL : mettez à jour les certificats CA de votre système, ou vérifiez si `HTTPS_PROXY` est configuré

Si vous êtes derrière un proxy d'entreprise, définissez `HTTPS_PROXY` et `HTTP_PROXY` à l'adresse de votre proxy avant d'installer. Demandez à votre équipe informatique l'URL du proxy si vous ne la connaissez pas, ou vérifiez les paramètres proxy de votre navigateur.

Cet exemple définit les deux variables proxy, puis exécute l'installateur via votre 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
```

Vérifier votre PATH

Si l'installation a réussi mais que vous obtenez une erreur `command not found` ou `not recognized` lors de l'exécution de `claude`, le répertoire d'installation n'est pas dans votre PATH. Votre shell recherche les programmes dans les répertoires listés dans PATH, et l'installateur place `claude` à `~/.local/bin/claude` sur macOS/Linux ou `%USERPROFILE%\local\bin\claude.exe` sur Windows.

Vérifiez si le répertoire d'installation est dans votre PATH en listant vos entrées PATH et en filtrant pour `local/bin` :

macOS/Linux

```
echo $PATH | tr ':' '\n' | grep local/bin
```

S'il n'y a pas de sortie, le répertoire est manquant. Ajoutez-le à votre configuration shell :

```
## Zsh (macOS par défaut)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

## Bash (Linux par défaut)
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Sinon, fermez et rouvrez votre terminal.

Vérifiez que la correction a fonctionné :


```
claude --version
```

Windows PowerShell

```
$env:PATH -split ';' | Select-String 'local\bin'
```

S'il n'y a pas de sortie, ajoutez le répertoire d'installation à votre PATH utilisateur :

```
$currentPath = [Environment]::GetEnvironmentVariable('PATH', 'User')
[Environment]::SetEnvironmentVariable('PATH', "$currentPath;$env:USERPROFILE\local\bin", 'User')
```

Redémarrez votre terminal pour que la modification prenne effet.

Vérifiez que la correction a fonctionné :

```
claude --version
```

Windows CMD

```
echo %PATH% | findstr /i "local\bin"
```

S'il n'y a pas de sortie, ouvrez Paramètres système, allez à Variables d'environnement, et ajoutez `%USERPROFILE%\local\bin` à votre variable PATH utilisateur. Redémarrez votre terminal.

Vérifiez que la correction a fonctionné :

```
claude --version
```

Vérifier les installations conflictuelles

Plusieurs installations de Claude Code peuvent causer des incompatibilités de version ou un comportement inattendu. Vérifiez ce qui est installé :

macOS/Linux

Listez tous les binaires `claude` trouvés dans votre PATH :

```
which -a claude
```

Vérifiez si les versions de l'installateur natif et npm sont présentes :

```
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"
```

Si vous trouvez plusieurs installations, conservez-en une seule. L'installation native à `~/.local/bin/claude` est recommandée. Supprimez les installations supplémentaires :

Désinstallez une installation npm globale :

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

Supprimez une installation Homebrew sur macOS :

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

Vérifier les permissions des répertoires

L'installateur a besoin d'accès en écriture à `~/.local/bin/` et `~/.claude/`. Si l'installation échoue avec des erreurs de permission, vérifiez si ces répertoires sont accessibles en écriture :

```
test -w ~/.local/bin && echo "writable" || echo "not writable"
test -w ~/.claude && echo "writable" || echo "not writable"
```

Si l'un des répertoires n'est pas accessible en écriture, créez le répertoire d'installation et définissez votre utilisateur comme propriétaire :

```
sudo mkdir -p ~/.local/bin
sudo chown -R $(whoami) ~/.local
```

Vérifier que le binaire fonctionne

Si **claude** est installé mais plante ou se fige au démarrage, exécutez ces vérifications pour réduire la cause.

Confirmez que le binaire existe et est exécutable :

```
ls -la $(which claude)
```

Sur Linux, vérifiez les bibliothèques partagées manquantes. Si **ldd** affiche des bibliothèques manquantes, vous devrez peut-être installer des paquets système. Sur Alpine Linux et autres distributions basées sur musl, voir [Configuration Alpine Linux](#).

```
ldd $(which claude) | grep "not found"
```

Exécutez une vérification rapide que le binaire peut s'exécuter :

```
claude --version
```

Problèmes d'installation courants

Ce sont les problèmes d'installation les plus fréquemment rencontrés et leurs solutions.

Le script d'installation retourne du HTML au lieu d'un script shell

Lors de l'exécution de la commande d'installation, vous pouvez voir l'une de ces erreurs :

```
bash: line 1: syntax error near unexpected token `<'
bash: line 1: `<!DOCTYPE html>'
```

Sur PowerShell, le même problème apparaît comme :

```
Invoke-Expression: Missing argument in parameter list.
```

Cela signifie que l'URL d'installation a retourné une page HTML au lieu du script d'installation. Si la page HTML dit « App unavailable in region », Claude Code n'est pas disponible dans votre pays. Voir [pays supportés](#).

Sinon, cela peut se produire en raison de problèmes réseau, de routage régional, ou d'une interruption de service temporaire.

Solutions :

1. Utiliser une méthode d'installation alternative :

Sur macOS ou Linux, installez via Homebrew :

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

Sur Windows, installez via WinGet :

```
winget install Anthropic.ClaudeCode
```

- Réessayer après quelques minutes** : le problème est souvent temporaire. Attendez et réessayez la commande originale.

command not found: claude après l'installation

L'installation s'est terminée mais `claude` ne fonctionne pas. L'erreur exacte varie selon la plateforme :

Plate-forme	Message d'erreur
macOS	<code>zsh: command not found: claude</code>
Linux	<code>bash: claude: command not found</code>

Plate-forme	Message d'erreur
Windows CMD	'claude' is not recognized as an internal or external command
Power-Shell	claude : The term 'claude' is not recognized as the name of a cmdlet

Cela signifie que le répertoire d'installation n'est pas dans le chemin de recherche de votre shell. Voir [Vérifier votre PATH](#) pour la correction sur chaque plateforme.

curl: (56) Failure writing output to destination

La commande `curl ... | bash` télécharge le script et le transmet directement à Bash pour exécution via un pipe (`|`). Cette erreur signifie que la connexion s'est interrompue avant que le script ne soit complètement téléchargé. Les causes courantes incluent les interruptions réseau, le téléchargement étant bloqué en cours de flux, ou les limites de ressources système.

Solutions :

1. **Vérifier la stabilité du réseau** : les binaires Claude Code sont hébergés sur Google Cloud Storage. Testez que vous pouvez l'atteindre :

```
curl -fsSL https://storage.googleapis.com -o /dev/null
```

Si la commande se termine silencieusement, votre connexion est correcte et le problème est probablement intermittent. Réessayez la commande d'installation. Si vous voyez une erreur, votre réseau bloque peut-être le téléchargement.

1. **Essayer une méthode d'installation alternative :**

Sur macOS ou Linux :

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

Sur Windows :

```
winget install Anthropic.ClaudeCode
```

Erreurs de connexion TLS ou SSL

Les erreurs comme `curl: (35) TLS connect error`, `schannel: next InitializeSecurityContext failed`, ou le `Could not establish trust relationship for the SSL/TLS secure channel` de PowerShell indiquent des échecs de négociation TLS.

Solutions :

1. Mettre à jour vos certificats CA système :

Sur Ubuntu/Debian :

```
sudo apt-get update && sudo apt-get install ca-certificates
```

Sur macOS via Homebrew :

```
brew install ca-certificates
```

1. Sur Windows, activez TLS 1.2 dans PowerShell avant d'exécuter l'installateur :

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12  
irm https://claude.ai/install.ps1 | iex
```

1. Vérifier l'interférence du proxy ou du pare-feu : les proxies d'entreprise qui effectuent l'inspection TLS peuvent causer ces erreurs, y compris `unable to get local issuer certificate`. Définissez `NODE_EXTRA_CA_CERTS` sur votre bundle de certificat CA d'entreprise :

```
export NODE_EXTRA_CA_CERTS=/path/to/corporate-ca.pem
```

Demandez à votre équipe informatique le fichier de certificat si vous ne l'avez pas. Vous pouvez également essayer sur une connexion directe pour confirmer que le proxy est la cause.

`Failed to fetch version from storage.googleapis.com`

L'installateur n'a pas pu atteindre le serveur de téléchargement. Cela signifie généralement que `storage.googleapis.com` est bloqué sur votre réseau.

Solutions :

1. Tester la connectivité directement :

```
curl -sI https://storage.googleapis.com
```

1. **Si derrière un proxy**, définissez `HTTPS_PROXY` pour que l'installateur puisse le router. Voir [configuration du proxy](#) pour les détails.

```
export HTTPS_PROXY=http://proxy.example.com:8080
curl -fsSL https://claude.ai/install.sh | bash
```

1. **Si sur un réseau restreint**, essayez un réseau différent ou un VPN, ou utilisez une méthode d'installation alternative :

Sur macOS ou Linux :

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

Sur Windows :

```
winget install Anthropic.ClaudeCode
```

Windows : `irm` ou `&&` non reconnu

Si vous voyez `'irm' is not recognized` ou `The token '&&' is not valid`, vous exécutez la mauvaise commande pour votre shell.

- **`irm` non reconnu** : vous êtes dans CMD, pas PowerShell. Vous avez deux options :
Ouvrez PowerShell en recherchant « PowerShell » dans le menu Démarrer, puis exécutez la commande d'installation originale :

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

Ou restez dans CMD et utilisez l'installateur CMD à la place :

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

- **&& non valide** : vous êtes dans PowerShell mais avez exécuté la commande d'installateur CMD. Utilisez l'installateur PowerShell :

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

Installation tuée sur les serveurs Linux à faible mémoire

Si vous voyez **Killed** pendant l'installation sur un VPS ou une instance cloud :

```
Setting up Claude Code...
Installing Claude Code native build latest...
bash: line 142: 34803 Killed      "$binary_path" install ${TARGET:+"$TARGET"}
```

Le tueur OOM Linux a terminé le processus car le système a manqué de mémoire. Claude Code nécessite au moins 4 Go de RAM disponible.

Solutions :

1. **Ajouter de l'espace d'échange** si votre serveur a une RAM limitée. L'échange utilise l'espace disque comme mémoire de débordement, permettant à l'installation de se terminer même avec une RAM physique faible.

Créez un fichier d'échange de 2 Go et activez-le :

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

Puis réessayez l'installation :

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

1. **Fermer d'autres processus** pour libérer de la mémoire avant d'installer.

2. **Utiliser une instance plus grande** si possible. Claude Code nécessite au moins 4 Go de RAM.

L'installation se fige dans Docker

Lors de l'installation de Claude Code dans un conteneur Docker, l'installation en tant que root dans `/` peut causer des blocages.

Solutions :

1. **Définir un répertoire de travail** avant d'exécuter l'installateur. Lorsqu'il est exécuté depuis `/`, l'installateur analyse l'ensemble du système de fichiers, ce qui provoque une utilisation excessive de la mémoire. Définir `WORKDIR` limite l'analyse à un petit répertoire :

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

1. **Augmenter les limites de mémoire Docker** si vous utilisez Docker Desktop :

```
docker build --memory=4g .
```

Windows : Claude Desktop remplace la commande CLI `claude`

Si vous avez installé une version antérieure de Claude Desktop, elle peut enregistrer un `Claude.exe` dans le répertoire `WindowsApps` qui prend la priorité PATH sur Claude Code CLI. L'exécution de `claude` ouvre l'application Desktop au lieu de la CLI.

Mettez à jour Claude Desktop vers la dernière version pour corriger ce problème.

Windows : « Claude Code on Windows requires git-bash »

Claude Code sur Windows natif a besoin de [Git for Windows](#), qui inclut Git Bash.

Si Git n'est pas installé, téléchargez et installez-le depuis git-scm.com/downloads/win. Pendant la configuration, sélectionnez « Add to PATH ». Redémarrez votre terminal après l'installation.

Si Git est déjà installé mais Claude Code ne peut toujours pas le trouver, définissez le chemin dans votre [fichier settings.json](#) :

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Si votre Git est installé ailleurs, trouvez le chemin en exécutant `where.exe git` dans PowerShell et utilisez le chemin `bin\\bash.exe` de ce répertoire.

Linux : mauvaise variante binaire installée (incompatibilité musl/glibc)

Si vous voyez des erreurs concernant des bibliothèques partagées manquantes comme `libstdc++.so.6` ou `libgcc_s.so.1` après l'installation, l'installateur a peut-être téléchargé la mauvaise variante binaire pour votre système.

```
Error loading shared library libstdc++.so.6: No such file or directory
```

Cela peut se produire sur les systèmes basés sur glibc qui ont des paquets de compilation croisée musl installés, ce qui amène l'installateur à mal détecter le système comme musl.

Solutions :

1. Vérifier quelle libc votre système utilise :

```
ldd /bin/ls | head -1
```

S'il affiche `linux-vdso.so` ou des références à `/lib/x86_64-linux-gnu/`, vous êtes sur glibc. S'il affiche `musl`, vous êtes sur musl.

1. **Si vous êtes sur glibc mais avez obtenu le binaire musl**, supprimez l'installation et réinstallez. Vous pouvez également télécharger manuellement le binaire correct depuis le bucket GCS à <https://storage.googleapis.com/claude-code-dist-86c565f3-f756-42ad-8dfa-d59b1c096819/claude-code-releases/{VERSION}/manifest.json>. Déposez un [problème GitHub](#) avec la sortie de `ldd /bin/ls` et `ls /lib/libc.musl*`.
2. **Si vous êtes réellement sur musl** (Alpine Linux), installez les paquets requis :

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

Illegal instruction sur Linux

Si l'installateur affiche **Illegal instruction** au lieu du message **Killed** OOM, le binaire téléchargé ne correspond pas à l'architecture de votre CPU. Cela se produit couramment sur les serveurs ARM qui reçoivent un binaire x86, ou sur les anciens CPU qui manquent d'ensembles d'instructions requis.

```
bash: line 142: 2238232 Illegal instruction   "$binary_path" install ${TARGET:
+ "$TARGET"}
```

Solutions :

1. Vérifier votre architecture :

```
uname -m
```

x86_64 signifie 64 bits Intel/AMD, **aarch64** signifie ARM64. Si le binaire ne correspond pas, [déposez un problème GitHub](#) avec la sortie.

1. Essayer une méthode d'installation alternative pendant que le problème d'architecture est résolu :

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

dyld: cannot load sur macOS

Si vous voyez **dyld: cannot load** ou **Abort trap: 6** pendant l'installation, le binaire est incompatible avec votre version ou matériel macOS.

```
dyld: cannot load 'claude-2.1.42-darwin-x64' (load command 0x80000034 is unknown)
Abort trap: 6
```

Solutions :

1. Vérifier votre version macOS : Claude Code nécessite macOS 13.0 ou ultérieur.

Ouvrez le menu Apple et sélectionnez À propos de ce Mac pour vérifier votre version.

2. **Mettre à jour macOS** si vous êtes sur une version antérieure. Le binaire utilise des commandes de chargement que les versions macOS antérieures ne supportent pas.
3. **Essayer Homebrew** comme méthode d'installation alternative :

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

Problèmes d'installation Windows : erreurs dans WSL

Vous pourriez rencontrer les problèmes suivants dans WSL :

Problèmes de détection d'OS/plateforme : si vous recevez une erreur pendant l'installation, WSL utilise peut-être npm Windows. Essayez :

- Exécutez `npm config set os linux` avant l'installation
- Installez avec `npm install -g @anthropic-ai/claude-code --force --no-os-check`. N'utilisez pas `sudo`.

Erreurs Node non trouvé : si vous voyez `exec: node: not found` lors de l'exécution de `claude`, votre environnement WSL utilise peut-être une installation Windows de Node.js. Vous pouvez confirmer cela avec `which npm` et `which node`, qui devraient pointer vers des chemins Linux commençant par `/usr/` plutôt que `/mnt/c/`. Pour corriger cela, essayez d'installer Node via le gestionnaire de paquets de votre distribution Linux ou via `nvm`.

Conflits de version nvm : si vous avez nvm installé à la fois dans WSL et Windows, vous pouvez rencontrer des conflits de version lors du changement de versions Node dans WSL. Cela se produit car WSL importe le PATH Windows par défaut, ce qui amène Windows nvm/npm à prendre la priorité sur l'installation WSL.

Vous pouvez identifier ce problème en :

- Exécutant `which npm` et `which node` - s'ils pointent vers des chemins Windows (commençant par `/mnt/c/`), les versions Windows sont utilisées
- Rencontrant une fonctionnalité cassée après le changement de versions Node avec nvm dans WSL

Pour résoudre ce problème, corrigez votre PATH Linux pour assurer que les versions Linux node/npm prennent la priorité :

Solution principale : Assurer que nvm est correctement chargé dans votre shell

La cause la plus courante est que nvm n'est pas chargé dans les shells non-interactifs. Ajoutez ce qui suit à votre fichier de configuration shell (`~/.bashrc` , `~/.zshrc` , etc.) :

```
## Load nvm if it exists
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

Ou exécutez directement dans votre session actuelle :

```
source ~/.nvm/nvm.sh
```

Alternative : Ajuster l'ordre du PATH

Si nvm est correctement chargé mais les chemins Windows prennent toujours la priorité, vous pouvez explicitement préfixer vos chemins Linux au PATH dans votre configuration shell :

```
export PATH="$HOME/.nvm/versions/node/$(node -v)/bin:$PATH"
```

Warning:

Évitez de désactiver l'importation du PATH Windows via `appendWindowsPath = false` car cela casse la capacité d'appeler les exécutables Windows depuis WSL. De même, évitez de désinstaller Node.js de Windows si vous l'utilisez pour le développement Windows.

Configuration du sandbox WSL2

Le [sandboxing](#) est supporté sur WSL2 mais nécessite l'installation de paquets supplémentaires. Si vous voyez une erreur comme « Sandbox requires socat and bubblewrap » lors de l'exécution de `/sandbox` , installez les dépendances :

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

WSL1 ne supporte pas le sandboxing. Si vous voyez « Sandboxing requires WSL2 », vous devez mettre à niveau vers WSL2 ou exécuter Claude Code sans sandboxing.

Erreurs de permission pendant l'installation

Si l'installateur natif échoue avec des erreurs de permission, le répertoire cible peut ne pas être accessible en écriture. Voir [Vérifier les permissions des répertoires](#).

Si vous avez précédemment installé avec npm et rencontrez des erreurs de permission spécifiques à npm, passez à l'installateur natif :

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

Permissions et authentification

Ces sections traitent des échecs de connexion, des problèmes de jetons et du comportement des invites de permission.

Invites de permission répétées

Si vous vous trouvez à approuver à plusieurs reprises les mêmes commandes, vous pouvez autoriser des outils spécifiques à s'exécuter sans approbation en utilisant la commande `/permissions`. Voir [Documentation des permissions](#).

Problèmes d'authentification

Si vous rencontrez des problèmes d'authentification :

1. Exécutez `/logout` pour vous déconnecter complètement
2. Fermez Claude Code
3. Redémarrez avec `claude` et complétez le processus d'authentification à nouveau

Si le navigateur ne s'ouvre pas automatiquement pendant la connexion, appuyez sur `c` pour copier l'URL OAuth dans votre presse-papiers, puis collez-la dans votre navigateur manuellement.

Erreur OAuth : Code invalide

Si vous voyez

`OAuth error: Invalid code. Please make sure the full code was copied`, le code de connexion a expiré ou a été tronqué lors de la copie-collage.

Solutions :

- Appuyez sur Entrée pour réessayer et complétez la connexion rapidement après l'ouverture du navigateur
- Tapez `c` pour copier l'URL complète si le navigateur ne s'ouvre pas automatiquement
- Si vous utilisez une session distante/SSH, le navigateur peut s'ouvrir sur la mauvaise machine. Copiez l'URL affichée dans le terminal et ouvrez-la dans votre navigateur local à la place.

403 Forbidden après la connexion

Si vous voyez `API Error: 403 {"error":{"type":"forbidden","message":"Request not allowed"}}` après la connexion :

- **Utilisateurs Claude Pro/Max** : vérifiez que votre abonnement est actif sur claude.ai/settings
- **Utilisateurs Console** : confirmez que votre compte a le rôle « Claude Code » ou « Developer » assigné par votre administrateur
- **Derrière un proxy** : les proxies d'entreprise peuvent interférer avec les requêtes API. Voir [configuration réseau](#) pour la configuration du proxy.

La connexion OAuth échoue dans WSL2

La connexion basée sur navigateur dans WSL2 peut échouer si WSL ne peut pas ouvrir votre navigateur Windows. Définissez la variable d'environnement `BROWSER` :

```
export BROWSER="/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"
claude
```

Ou copiez l'URL manuellement : lorsque l'invite de connexion apparaît, appuyez sur `c` pour copier l'URL OAuth, puis collez-la dans votre navigateur Windows.

« Not logged in » ou jeton expiré

Si Claude Code vous demande de vous connecter à nouveau après une session, votre jeton OAuth a peut-être expiré.

Exécutez `/login` pour vous réauthentifier. Si cela se produit fréquemment, vérifiez que votre horloge système est exacte, car la validation des jetons dépend des horodatages corrects.

Emplacements des fichiers de configuration

Claude Code stocke la configuration dans plusieurs emplacements :

Fichier	Objectif
<code>~/ .claude/ settings.json</code>	Paramètres utilisateur (permissions, hooks, remplacements de modèle)
<code>.claude/ settings.json</code>	Paramètres de projet (vérifiés dans le contrôle de source)
<code>.claude/ settings.local.json</code>	Paramètres de projet locaux (non validés)
<code>~/ .claude.json</code>	État global (thème, OAuth, serveurs MCP)
<code>.mcp.json</code>	Serveurs MCP de projet (vérifiés dans le contrôle de source)
<code>managed-mcp.json</code>	Serveurs MCP gérés
Paramètres gérés	Paramètres gérés (gérés par serveur, politiques MDM/niveau OS, ou basés sur fichier)

Sur Windows, `~` fait référence à votre répertoire personnel utilisateur, tel que `C:\Users\YourName`.

Pour plus de détails sur la configuration de ces fichiers, voir [Paramètres](#) et [MCP](#).

Réinitialiser la configuration

Pour réinitialiser Claude Code aux paramètres par défaut, vous pouvez supprimer les fichiers de configuration :

```
## Réinitialiser tous les paramètres utilisateur et l'état
rm ~/ .claude.json
rm -rf ~/ .claude/

## Réinitialiser les paramètres spécifiques au projet
rm -rf .claude/
rm .mcp.json
```

Warning:

Cela supprimera tous vos paramètres, configurations de serveur MCP et historique de session.

Performance et stabilité

Ces sections couvrent les problèmes liés à l'utilisation des ressources, la réactivité et le comportement de recherche.

Utilisation élevée du CPU ou de la mémoire

Claude Code est conçu pour fonctionner avec la plupart des environnements de développement, mais peut consommer des ressources importantes lors du traitement de grandes bases de code. Si vous rencontrez des problèmes de performance :

1. Utilisez `/compact` régulièrement pour réduire la taille du contexte
2. Fermez et redémarrez Claude Code entre les tâches majeures
3. Envisagez d'ajouter les grands répertoires de construction à votre fichier `.gitignore`

Les commandes se figent ou se gèlent

Si Claude Code semble ne pas répondre :

1. Appuyez sur Ctrl+C pour tenter d'annuler l'opération actuelle
2. Si ne répond pas, vous devrez peut-être fermer le terminal et redémarrer

Problèmes de recherche et de découverte

Si l'outil Search, les mentions `@file`, les agents personnalisés et les skills personnalisés ne fonctionnent pas, installez le système `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
```

Puis définissez `USE_BUILTIN_RIPGREP=0` dans votre [environnement](#).

Résultats de recherche lents ou incomplets sur WSL

Les pénalités de performance de lecture de disque lors du [travail sur les systèmes de fichiers sur WSL](#) peuvent entraîner moins de correspondances que prévu lors de l'utilisation de Claude Code sur WSL. La recherche fonctionne toujours, mais retourne moins de résultats que sur un système de fichiers natif.

Note:

`/doctor` affichera Search comme OK dans ce cas.

Solutions :

1. **Soumettre des recherches plus spécifiques** : réduisez le nombre de fichiers recherchés en spécifiant des répertoires ou des types de fichiers : « Search for JWT validation logic in the auth-service package » ou « Find use of md5 hash in JS files ».
2. **Déplacer le projet vers le système de fichiers Linux** : si possible, assurez-vous que votre projet est situé sur le système de fichiers Linux (`/home/`) plutôt que sur le système de fichiers Windows (`/mnt/c/`).
3. **Utiliser Windows natif à la place** : envisagez d'exécuter Claude Code nativement sur Windows au lieu de via WSL, pour une meilleure performance du système de fichiers.

Problèmes d'intégration IDE

Si Claude Code ne se connecte pas à votre IDE ou se comporte de manière inattendue dans un terminal IDE, essayez les solutions ci-dessous.

IDE JetBrains non détecté sur WSL2

Si vous utilisez Claude Code sur WSL2 avec des IDE JetBrains et recevez des erreurs « No available IDEs detected », cela est probablement dû à la configuration réseau de WSL2 ou au pare-feu Windows bloquant la connexion.

Modes de mise en réseau WSL2

WSL2 utilise la mise en réseau NAT par défaut, ce qui peut empêcher la détection d'IDE. Vous avez deux options :

Option 1 : Configurer le pare-feu Windows (recommandé)

1. Trouvez votre adresse IP WSL2 :

```
wsl hostname -I
## Exemple de sortie : 172.21.123.45
```

1. Ouvrez PowerShell en tant qu'administrateur et créez une règle de pare-feu :

```
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
```

Ajustez la plage IP en fonction de votre sous-réseau WSL2 à partir de l'étape 1.

1. Redémarrez à la fois votre IDE et Claude Code

Option 2 : Passer à la mise en réseau en miroir

Ajoutez à `.wslconfig` dans votre répertoire utilisateur Windows :

```
[wsl2]
networkingMode=mirrored
```

Puis redémarrez WSL avec `wsl --shutdown` depuis PowerShell.

Note:

Ces problèmes de mise en réseau n'affectent que WSL2. WSL1 utilise directement le réseau de l'hôte et ne nécessite pas ces configurations.

Pour des conseils de configuration JetBrains supplémentaires, voir le [guide IDE JetBrains](#).

Signaler les problèmes d'intégration IDE Windows

Si vous rencontrez des problèmes d'intégration IDE sur Windows, [créez un problème](#) avec les informations suivantes :

- Type d'environnement : Windows natif (Git Bash) ou WSL1/WSL2
- Mode de mise en réseau WSL, le cas échéant : NAT ou miroir
- Nom et version de l'IDE
- Version de l'extension/plugin Claude Code
- Type de shell : Bash, Zsh, PowerShell, etc.

La touche Échap ne fonctionne pas dans les terminaux IDE JetBrains

Si vous utilisez Claude Code dans les terminaux JetBrains et que la touche **Esc** n'interrompt pas l'agent comme prévu, cela est probablement dû à un conflit de liaison de touches avec les raccourcis par défaut de JetBrains.

Pour corriger ce problème :

1. Allez à Paramètres → Outils → Terminal
2. Soit :
 - Décochez « Move focus to the editor with Escape », soit
 - Cliquez sur « Configure terminal keybindings » et supprimez le raccourci « Switch focus to Editor »
3. Appliquez les modifications

Cela permet à la touche **Esc** d'interrompre correctement les opérations Claude Code.

Problèmes de formatage Markdown

Claude Code génère parfois des fichiers markdown avec des balises de langage manquantes sur les clôtures de code, ce qui peut affecter la coloration syntaxique et la lisibilité dans GitHub, les éditeurs et les outils de documentation.

Balises de langage manquantes dans les blocs de code

Si vous remarquez des blocs de code comme celui-ci dans le markdown généré :

```
```
function example() {
 return "hello";
}
```text
```

Au lieu de blocs correctement balisés comme :

```
```javascript
function example() {
 return "hello";
}
```text
```

Solutions :

1. **Demander à Claude d'ajouter des balises de langage** : demandez « Add appropriate language tags to all code blocks in this markdown file. »
2. **Utiliser des hooks de post-traitement** : configurez des hooks de formatage automatique pour détecter et ajouter les balises de langage manquantes. Voir [Auto-format code after edits](#) pour un exemple de hook PostToolUse de formatage.
3. **Vérification manuelle** : après la génération de fichiers markdown, examinez-les pour un formatage correct des blocs de code et demandez des corrections si nécessaire.

Espacement et formatage incohérents

Si le markdown généré a des lignes vides excessives ou un espacement incohérent :

Solutions :

1. **Demander des corrections de formatage** : demandez « Fix spacing and formatting issues in this markdown file. »
2. **Utiliser des outils de formatage** : configurez des hooks pour exécuter des formateurs markdown comme `prettier` ou des scripts de formatage personnalisés sur les fichiers markdown générés.

3. **Spécifier les préférences de formatage** : incluez les exigences de formatage dans vos invites ou fichiers [memory](#) de projet.

Réduire les problèmes de formatage markdown

Pour minimiser les problèmes de formatage :

- **Être explicite dans les demandes** : demandez « properly formatted markdown with language-tagged code blocks »
- **Utiliser les conventions de projet** : documentez votre style markdown préféré dans `CLAUDE.md`
- **Configurer des hooks de validation** : utilisez des hooks de post-traitement pour vérifier et corriger automatiquement les problèmes de formatage courants

Obtenir plus d'aide

Si vous rencontrez des problèmes non couverts ici :

1. Utilisez la commande `/bug` dans Claude Code pour signaler les problèmes directement à Anthropic
2. Vérifiez le [référentiel GitHub](#) pour les problèmes connus
3. Exécutez `/doctor` pour diagnostiquer les problèmes. Il vérifie :
 - Type d'installation, version et fonctionnalité de recherche
 - Statut de mise à jour automatique et versions disponibles
 - Fichiers de paramètres invalides (JSON malformé, types incorrects)
 - Erreurs de configuration du serveur MCP
 - Problèmes de configuration des liaisons de touches
 - Avertissements d'utilisation du contexte (fichiers CLAUDE.md volumineux, utilisation élevée de jetons MCP, règles de permission inaccessibles)
 - Erreurs de chargement des plugins et agents
4. Demandez directement à Claude ses capacités et fonctionnalités - Claude a un accès intégré à sa documentation