GUIDE PRATIQUE · INTELLIGENCE ARTIFICIELLE

Claude Code : 5 bonnes pratiques pour coder plus vite et en sécurité

Comment cadrer un agent de développement IA avec CLAUDE.md, la gestion du contexte, les sous-agents et les permissions.

As My Agency
asmyagency.com · Guide mis à jour le 18 août 2026

Blog & Formations IA

Au programme de ce guide

À retenir : trois priorités gouvernent un bon usage de Claude Code — des instructions de projet claires, une fenêtre de contexte préservée, et des permissions adaptées au risque de chaque action.

Claude Code ne se contente pas de compléter quelques lignes de code. Cet outil agentique d'Anthropic peut explorer un dépôt, modifier plusieurs fichiers, exécuter des commandes, lancer des tests et interagir avec Git ou des services externes.

Pour bien utiliser Claude Code, retenez trois priorités : lui fournir des instructions de projet claires, préserver sa fenêtre de contexte et encadrer ses actions avec des permissions adaptées. Les cinq bonnes pratiques ci-dessous permettent d'obtenir des réponses plus fiables, de limiter les allers-retours et d'éviter les modifications hasardeuses.

L'ESSENTIEL EN 30 SECONDES

Créez un fichier `CLAUDE.md` concis, utilisez `/context` puis `/compact` ou `/clear` au bon moment, déléguez les explorations volumineuses à des sous-agents, exigez un format de sortie précis et passez en mode Plan avant toute modification complexe.

Informations vérifiées le 18 août 2026 à partir de la documentation officielle d'Anthropic.

Qu'est-ce que Claude Code ?

Claude Code est un agent de développement assisté par intelligence artificielle. Contrairement à un simple chatbot, il peut lire une base de code, comprendre son organisation, proposer une stratégie, modifier des fichiers, exécuter des commandes et vérifier le résultat.

Selon la documentation officielle de Claude Code, l'outil est disponible dans le terminal, les environnements de développement comme VS Code et JetBrains, l'application Desktop et le navigateur. Il peut également se connecter à d'autres outils grâce au Model Context Protocol, ou MCP.

Son efficacité ne dépend donc pas seulement de la qualité de votre prompt. Elle repose surtout sur votre capacité à lui transmettre **le bon contexte, au bon moment, avec le bon niveau d'autonomie**.

1 Faites de CLAUDE.md le manuel opérationnel du projet

Le fichier `CLAUDE.md` donne à Claude Code les informations qu'il doit connaître pour travailler correctement sur un projet : commandes de build, conventions de code, architecture, bibliothèques à privilégier, procédures de test ou erreurs à éviter.

Anthropic conseille d'y placer les consignes que vous devriez autrement répéter à chaque nouvelle session. Voici un exemple simple :

```
# Projet

## Commandes
- Installer les dépendances : pnpm install
- Lancer les tests : pnpm test
- Vérifier le lint : pnpm lint

## Conventions
- Utiliser TypeScript strict.
- Préférer les fonctions pures dans le domaine métier.
- Ne jamais modifier une migration déjà appliquée.

## Avant de terminer
- Exécuter les tests concernés.
- Vérifier le formatage.
- Résumer les fichiers modifiés et les limites connues.
```

Où placer le fichier CLAUDE.md ?

Claude Code prend en charge plusieurs niveaux d'instructions :

- `~/.claude/CLAUDE.md` pour vos préférences personnelles, valables dans tous vos projets ;
- `./CLAUDE.md` ou `./.claude/CLAUDE.md` pour les règles partagées au niveau du dépôt ;
- `./CLAUDE.local.md` pour vos préférences personnelles liées à ce projet ;
- des fichiers `CLAUDE.md` dans les sous-répertoires pour des consignes propres à une partie du code.

Dans un dépôt complexe, le dossier `.claude/rules/` permet aussi de séparer les règles par thème et de les associer à certains chemins. Une règle dédiée à `src/api/**/*.ts`, par exemple, n'a pas besoin d'être chargée pour une tâche concernant uniquement le frontend.

Gardez des instructions courtes et vérifiables

Un bon fichier `CLAUDE.md` n'est pas une encyclopédie. Anthropic recommande de viser moins de 200 lignes par fichier : des instructions trop longues occupent inutilement la fenêtre de contexte et risquent d'être moins bien suivies.

Préférez donc :

- « Exécute `pnpm test` avant de terminer » à « Pense à tester le projet » ;
- « Les handlers API se trouvent dans `src/api/handlers/` » à « Respecte l'architecture » ;
- « Utilise une indentation de deux espaces » à « Formate correctement le code ».

Vous pouvez lancer `/init` pour générer une première version de `CLAUDE.md`, puis `/context` pour vérifier que le fichier a bien été chargé.

Attention : CLAUDE.md fournit du contexte, mais ce n'est pas une barrière de sécurité. Pour interdire techniquement une commande ou l'accès à un secret, utilisez les permissions ou un hook `PreToolUse`.

2 Gérez le contexte avant qu'il ne devienne du bruit

La fenêtre de contexte est la mémoire de travail de Claude Code. Elle contient la conversation, les fichiers lus, les sorties de commandes, les instructions et les résultats des outils utilisés pendant la session.

Quand cette fenêtre se remplit de données devenues inutiles, Claude peut perdre de vue les décisions importantes ou consommer davantage de ressources à chaque nouvelle requête. Il dispose d'une auto-compaction, mais vous pouvez garder la main avec trois commandes :

Commande	Quand l'utiliser	Effet
`/context`	Pour comprendre ce qui occupe la fenêtre de contexte	Affiche la répartition de l'espace utilisé
`/compact`	Pour poursuivre la même tâche avec un historique résumé	Compresse la conversation en conservant l'essentiel
`/clear`	Pour passer à une tâche sans rapport avec la précédente	Démarre une nouvelle session avec un contexte vide

Vous pouvez guider la compaction en précisant les informations à conserver :

```
/compact Conserve les fichiers modifiés, les décisions d'architecture,
les erreurs de test et les prochaines étapes.
```

Il n'existe pas de seuil universel du type « compactez exactement à 60 % ». Le bon moment dépend du modèle, de la taille de sa fenêtre et de la complexité de la tâche. La commande `/autocompact` permet d'ailleurs de régler la limite de déclenchement automatique sur les versions compatibles.

La règle la plus simple reste celle-ci :

- utilisez `/compact` lorsque vous voulez poursuivre le même objectif ;
- utilisez `/clear` lorsque vous changez complètement de sujet ;
- utilisez un sous-agent lorsqu'une recherche risque de produire beaucoup de lectures ou de logs.

À noter : le `CLAUDE.md` placé à la racine du projet est relu après une compaction. Les instructions propres à un sous-dossier sont, elles, rechargées lorsque Claude lit de nouveau un fichier correspondant.

3 Déléguez les explorations volumineuses à des sous-agents

Un sous-agent travaille dans sa propre fenêtre de contexte. Il peut donc explorer de nombreux fichiers, analyser des logs ou réaliser une revue ciblée sans remplir la conversation principale de détails intermédiaires.

Cette délégation est particulièrement utile pour :

- rechercher l'origine d'un bug dans plusieurs modules ;
- analyser des logs volumineux ;
- réaliser une revue de code en lecture seule ;
- comparer plusieurs approches techniques ;
- examiner les résultats détaillés d'une suite de tests ;
- consulter une documentation longue et restituer uniquement les conclusions.

Exemple de demande :

```
Utilise un sous-agent en lecture seule pour analyser les risques de sécurité
dans les fichiers modifiés. Retourne uniquement les alertes confirmées,
leur gravité, la preuve et le fichier concerné.
```

Donnez au sous-agent une mission autonome

Les sous-agents ne reçoivent pas automatiquement tout l'historique de la conversation principale. Une mission efficace doit donc préciser :

1. l'objectif à atteindre ;
2. le périmètre à analyser ;
3. les outils autorisés ou interdits ;
4. les critères de réussite ;
5. le format du résultat attendu.

Les agents personnalisés peuvent être placés dans `.claude/agents/` pour un projet ou dans `~/.claude/agents/` pour tous les projets. Ils peuvent recevoir un modèle, des outils, des permissions, une mémoire persistante ou un environnement Git isolé avec `isolation: worktree`.

> Chaque agent consomme ses propres tokens. Multiplier les agents sans raison n'accélère pas automatiquement le travail : déléguez uniquement les tâches indépendantes ou suffisamment volumineuses pour justifier un contexte séparé.

4 Demandez un résultat précis, pas seulement « une réponse concise »

Une consigne de sortie vérifiable améliore la qualité du résultat. « Sois concis » reste subjectif ; une structure, une limite et des champs obligatoires sont beaucoup plus faciles à respecter.

Par exemple :

```
Réponds en trois sections maximum.
Pour chaque problème, indique :
1. le fichier ;
2. la gravité ;
3. la preuve ;
4. la correction proposée.
Ne mentionne pas les fichiers inchangés.
```

Pour réduire le bruit sans perdre les informations utiles, vous pouvez aussi demander à Claude Code de :

- retourner uniquement les erreurs confirmées ;
- filtrer les logs et exclure les lignes sans impact ;
- présenter un diff ciblé plutôt que le contenu complet des fichiers ;
- classer les problèmes par priorité ;
- séparer les faits, les hypothèses et les recommandations ;
- terminer par les tests exécutés et les risques encore ouverts.

Le niveau de détail doit rester proportionné au risque. Une correction de libellé n'exige pas le même rapport qu'une migration de base de données ou qu'une revue de sécurité. Le but n'est pas d'obtenir la réponse la plus courte, mais **la réponse la plus utile pour décider**.

Si vous répétez souvent le même format, ajoutez-le à `CLAUDE.md` ou créez une skill Claude Code dédiée. Vous éviterez ainsi de reformuler la même consigne à chaque session.

5 Passez en mode Plan avant une modification complexe

Le mode Plan permet à Claude Code d'explorer le dépôt et de proposer une stratégie avant de modifier les fichiers. Il est particulièrement utile lorsqu'une tâche touche plusieurs composants, comporte des risques ou nécessite une validation humaine.

Vous pouvez formuler votre demande ainsi :

```
Passe d'abord en mode Plan. Analyse les fichiers concernés, propose les étapes,
identifie les risques et attends ma validation avant toute modification.
```

Une bonne demande de plan précise :

- les fichiers ou répertoires concernés ;
- le comportement attendu ;
- les tests à créer ou à modifier ;
- les contraintes techniques et métier ;
- les effets de bord à surveiller ;
- le critère qui permettra de considérer la tâche comme terminée.

Une fois le plan validé, avancez par petits incréments : une modification ciblée, un test, puis l'étape suivante. Anthropic recommande cette progression, car elle permet de détecter rapidement une mauvaise hypothèse avant qu'elle ne contamine tout le projet.

Pour les équipes qui pilotent des projets digitaux, cette logique rejoint les fondamentaux de cadrage, de gestion des risques et de recette abordés dans la formation **Gestion de Projet Web** d'As My Agency.

! Le réflexe indispensable : sécurisez les actions de Claude Code

La productivité ne doit jamais effacer la maîtrise du risque. Claude Code permet de définir des règles pour autoriser, demander une confirmation ou interdire certaines actions, notamment la lecture de fichiers, les commandes shell, les modifications, les requêtes web et les outils MCP.

La commande `/permissions` permet de consulter ou d'ajuster ces règles. Vous pouvez par exemple :

- autoriser une commande de test connue comme `Bash(npm run test *)` ;
- interdire la lecture des fichiers `.env` ;
- bloquer un `git push` direct ;
- exiger une validation avant une migration destructive ;
- déclencher un contrôle automatique avec un hook `PreToolUse`.

Une instruction écrite dans `CLAUDE.md` rappelle une règle ; une permission ou un hook peut réellement la faire appliquer.

> N'utilisez le mode `bypassPermissions` que dans un environnement isolé, comme un conteneur ou une machine virtuelle. Ce mode supprime la plupart des demandes de confirmation : rapide, certes, mais pas magique — un agent très motivé reste capable de faire une grosse bêtise très vite.

✓ Workflow Claude Code : la checklist avant de commencer

Voici un déroulé simple pour garder une session claire et contrôlée :

1. Lancez Claude Code à la racine du projet.
2. Utilisez `/init` si le dépôt ne possède pas encore de CLAUDE.md.
3. Vérifiez les instructions et l'occupation du contexte avec `/context`.
4. Pour une tâche complexe, activez le mode Plan et définissez les critères de réussite.
5. Déléguez les recherches volumineuses à un sous-agent avec une mission précise.
6. Faites modifier et tester le projet par petites étapes.
7. Utilisez `/compact` pour poursuivre le même objectif ou `/clear` pour changer de sujet.
8. Demandez un bilan final : fichiers modifiés, tests exécutés, limites et risques restants.

Claude Code devient réellement efficace lorsqu'on le traite comme un environnement de développement configurable, et non comme une simple boîte de dialogue. Le gain ne vient pas d'un prompt « parfait », mais d'un cadre de travail cohérent : contexte propre, instructions concrètes, délégation raisonnée, vérifications explicites et permissions proportionnées au risque.

Questions fréquentes sur Claude Code

À quoi sert Claude Code ?

Claude Code sert à analyser une base de code, modifier des fichiers, exécuter des commandes, lancer des tests et automatiser des tâches de développement. Il fonctionne comme un agent capable d'enchaîner plusieurs actions, sous le contrôle des permissions définies par l'utilisateur ou l'organisation.

Quelle est la différence entre CLAUDE.md et un prompt ?

Un prompt décrit généralement la tâche du moment. Le fichier CLAUDE.md contient les règles et informations réutilisables du projet : commandes, conventions, architecture ou critères de validation. Claude Code le charge comme contexte, mais il ne remplace pas une permission de sécurité.

Quand utiliser /compact ou /clear dans Claude Code ?

Utilisez /compact pour résumer l'historique tout en poursuivant la même tâche. Utilisez /clear lorsque vous commencez une mission sans rapport avec la précédente et que vous n'avez plus besoin de son contexte.

Pourquoi utiliser un sous-agent Claude Code ?

Un sous-agent permet d'isoler une exploration, une revue ou une analyse volumineuse dans une autre fenêtre de contexte. La conversation principale ne reçoit que la synthèse, ce qui préserve sa lisibilité. En contrepartie, chaque sous-agent consomme ses propres ressources.

Le mode bypassPermissions est-il sécurisé ?

Le mode bypassPermissions réduit fortement les demandes de confirmation et augmente donc le risque d'action non souhaitée. Anthropic recommande de l'utiliser uniquement dans un environnement isolé, par exemple un conteneur ou une machine virtuelle, et jamais comme réglage par défaut sur un poste contenant des données sensibles.

Pour aller plus loin avec As My Agency

Vous souhaitez intégrer l'IA générative dans vos méthodes de travail sans sacrifier la sécurité ni le jugement humain ? Découvrez nos formations ou parlons de votre projet.

Formation Les bases de l'IA générative

Formation Gestion de Projet Web

Contacter As My Agency

As My Agency — asmyagency.com