C02 Prompting Basics pour Claude Code

Conception Beginner

Les principes qui font la différence entre un bon et un mauvais prompt

Contexte > longueur

Un prompt de 3 lignes avec le bon contexte produit de meilleurs résultats qu'un prompt de 30 lignes vague. Claude a besoin de savoir où il se trouve dans le projet, quel est l'objectif précis, et ce qu'il ne doit pas toucher.

La question à se poser avant d'écrire un prompt : « Est-ce que quelqu'un qui découvre cette codebase aujourd'hui comprendrait exactement ce que je veux ? » Si la réponse est non, le prompt manque de contexte.

Spécificité : le mot qui change tout

« Refactoring » ne dit rien. « Extraire la fonction `validateUser` vers `src/auth/validation.ts` en gardant tous les tests verts et sans modifier les types publics » dit tout ce qu'il faut.

Formulation vague	Formulation précise
Améliore ce code	Réduire la complexité cyclomatique de parseConfig() sans changer l'API publique
Ajoute des tests	Écrire des tests unitaires pour validateUser, couvrir les cas null et les emails malformés
Corrige le bug	La fonction sortItems trie par ordre inverse, corriger en gardant le même type de retour

Itération courte : valider par incréments

Demander tout d'un coup est tentant mais risqué. Claude peut produire une solution cohérente en interne mais qui ne correspond pas aux attentes implicites. La pratique qui fonctionne : décomposer en étapes de 20-40 lignes maximum, valider chacune avant de continuer.

```
# Après chaque changement, vérifier avant de continuer
git diff
pnpm test
```

Montrer plutôt qu'expliquer

Donner un fichier de référence vaut plus qu'expliquer le style en prose. Si vous voulez que Claude suive les conventions de `UserService`, indiquez-lui simplement : « Suis le même pattern que `src/services/UserService.ts` . »

Le principe s'applique aussi aux types : si vous avez un type `ApiResponse<T>` existant, montrez-le au lieu de le décrire.

Ce qu'il ne faut PAS faire

Questions rhétoriques : « Peux-tu refactoriser cette fonction ? » invite Claude à potentiellement répondre « oui » puis attendre. Préférez l'impératif direct : « Refactorise cette fonction. »

Contraintes implicites : Si les tests doivent rester verts, si les types publics ne doivent pas changer, si une librairie spécifique est interdite, dites-le explicitement. Claude n'a pas accès à vos conventions non documentées.

Ambiguïté sur le scope : « Ce fichier » sans référence, « l'authentification » sans préciser quel module. Utilisez des chemins absolus ou des noms de fonctions.

Le prompt minimaliste efficace

```
Contexte : [fichier ou module concerné]
Objectif : [action précise + critère de succès]
Contraintes : [ce que tu ne dois PAS modifier]
```

Trois lignes suffisent pour la majorité des tâches courantes.