Configuration d'assistant de code, partageable en équipe

Configuration Claude Code packagée pour être réutilisée par une équipe : crochets automatiques, procédures méthodologiques, et un gabarit d'instructions à personnaliser.

Fiche de méthode associée : `../../docs/15-claude-code-configuration.md`

Le principe qui structure tout

Les instructions orientent, les crochets garantissent.

Toute règle dont la violation coûte cher doit être un crochet, pas une phrase dans un fichier d'instructions. Une règle écrite trois fois et toujours violée est une règle qui doit devenir un crochet.

Contenu

Crochets (`hooks/`)

Crochet	Moment	Rôle
`block-sensitive.sh` + `block_sensitive.py`	avant lecture, écriture, commande	bloque l'accès aux fichiers et variables sensibles, et les publications non conformes
`block-coauthor.sh`	avant commande	refuse une validation contenant une mention de co-auteur automatique
`ruff-on-save.sh`	après écriture	formate et analyse les fichiers Python, remonte les erreurs non corrigeables
`sqlfluff-on-save.sh`	après écriture	formate le SQL si le projet a une configuration de dialecte
`session-start.sh`	démarrage	audite le projet, signale ce qui manque, injecte du contexte
`precompact-context.sh`	avant compaction	dicte ce qui doit survivre au résumé
`sync-config-repos.sh`	fin de session	synchronise et sauvegarde la configuration
`toast-notify.sh`	arrêt, notification	alerte visuelle quand une action est attendue

Procédures (`skills/`)

Neuf procédures encapsulant des méthodes réutilisables : chercher avant d'écrire, conventions Python, patterns de test, patterns de conteneurisation, revue de sécurité, boucle de vérification, conversion de diagrammes, traitement d'une friction, graphe de connaissance.

Ce qui distingue une bonne procédure d'une documentation : elle décrit une méthode (dans quel ordre faire les choses, où s'arrêter), pas un catalogue de connaissances.

Gabarit d'instructions (`CLAUDE.md.template`)

Base à personnaliser, dont la partie communication est explicitement à adapter. Le reste (garde-fous, échelle de décision avant d'écrire du code, stack, séparation configuration et secrets, posture d'arrivée sur un projet) est directement réutilisable.

Ce qui vaut d'être regardé

1. Le crochet de protection des secrets (`hooks/block_sensitive.py`)

Le fichier le plus intéressant du projet, parce qu'il a été construit itérativement en corrigeant ses propres faux positifs.

Deux familles de contrôles : sur les chemins de fichiers (motifs sensibles, avec une liste blanche pour les fichiers d'exemple, qui sont une bonne pratique et doivent rester lisibles), et sur les commandes (lecture de fichier sensible, affichage de variable sensible, publication non conforme).

Trois leçons de sécurité inscrites dans le code, chacune issue d'un incident :

Leçon	Pourquoi
Normaliser les jetons avant de les tester	un nom de fichier suivi d'un point-virgule, ce qui arrive dans toute boucle shell, échappe à une liste blanche ancrée sur la fin de chaîne
Tester le programme invoqué, pas le texte de la commande	un motif appliqué à la chaîne entière déclenche sur toute commande qui *mentionne* l'outil, y compris dans un message de validation. Cela rend impossible d'écrire de la documentation sur ses propres outils
Un contrôle trop large finit contourné	s'il bloque des usages légitimes plusieurs fois par session, l'utilisateur trouve un chemin de traverse et le garde-fou ne protège plus rien

La précision de ciblage est une propriété de sécurité, pas un détail de confort. C'est la leçon principale.

2. Les tests du crochet (`hooks/test_block_sensitive.py`)

26 tests. Un crochet est du code qui décide d'autoriser ou refuser : il mérite le même traitement que du code de production.

Couvert en priorité : les blocages attendus par catégorie, les exceptions légitimes (fichiers d'exemple), et les cas de fermeture par défaut.

3. Le découpage partageable (`README.md`)

Séparation stricte entre ce qui va dans un dépôt d'équipe et ce qui reste personnel :

Partagé	Personnel
crochets génériques	chemins de poste, synchronisations locales
procédures réutilisables	conventions internes à l'organisation
gabarit d'instructions	identifiants et jetons

C'est ce qui rend la configuration réellement adoptable : chaque nouvel arrivant gagne les mois de tâtonnement qui l'ont produite.

4. La synchronisation à sens unique (`hooks/sync-config-repos.sh`)

Une source de vérité, une synchronisation automatique dans un seul sens, à un moment déterministe du cycle de vie.

Le commentaire du script documente le piège corrigé : une copie dans le mauvais sens écrasait les modifications faites dans le dépôt. Depuis, les entrées du répertoire d'exécution sont des liens symboliques vers le dépôt, et seule la couche Windows exige une copie réelle, parce que l'interpréteur de commandes Windows ne suit pas les liens symboliques.

Vérification après anonymisation

Contrôle	Résultat
Syntaxe des 8 crochets shell	ok
Compilation des fichiers Python	ok
Tests du crochet de sécurité	26 passent
Motifs sensibles résiduels	aucun

Ce qui a été retiré

- l'historique de contrôle de version ;
- les caches d'outillage ;

- les références de demandes internes dans les commentaires, remplacées par une mention générique quand l'énumération devenait illisible.

Un point conservé volontairement : les commentaires expliquant pourquoi un contrôle existe et quel incident l'a motivé. C'est ce qui rend le crochet maintenable par quelqu'un d'autre.