Chapitre E — Debug, Investigation & Code Review

6 skills regroupés sous ce thème.

- review
- investigate
- codex
- health
- careful
- devex-review

review

review

> *Catalogue généré le 2026-05-11*

En une phrase

Une relecture de code automatique avant de fusionner ta branche : Claude lit ta « diff » (les changements) et te signale tout ce qui pourrait poser problème.

Quand l'utiliser

- Quand tu viens de finir une fonctionnalité et que tu veux un avis avant de la pousser en production
- Quand tu veux vérifier qu'aucun changement parasite ne s'est glissé dans ta branche (« scope drift »)
- Quand tu veux savoir si ta branche couvre bien ce qui était prévu dans ton plan ou ton ticket
- Avant de demander à Claude de lancer `/ship`, pour repérer les soucis tôt
- Quand un PR est ouvert et que tu veux une seconde lecture critique

Comment l'invoquer

- **Slash command** : `/review` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "review this pr" / "code review" / "check my diff" / "pre-landing review"
- **Auto-invocation** : Oui — Claude propose ce skill proactivement quand tu t'apprêtes à fusionner ou livrer du code.

Description détaillée

`/review` est une relecture sérieuse de tout ce que ta branche a changé par rapport à la branche principale (`main` ou `master`). Claude détecte d'abord la branche cible (GitHub, GitLab ou en git pur), puis lance plusieurs vérifications. Il compare le contenu de ta « diff » à ce qui était demandé dans `TODOS.md`, dans la description du PR ou dans tes messages de commit — pour voir si tu as fait trop, trop peu, ou exactement ce qu'il fallait.

Le skill cherche les problèmes structurels que les tests automatiques ne voient pas : risques de sécurité SQL, frontières de confiance avec les LLM, effets de bord conditionnels, code dupliqué, manques de tests, etc. Quand un fichier de plan existe (par exemple un design issu de `/office-hours`), Claude vérifie aussi si chaque élément du plan a bien été livré, ou s'il manque quelque chose.

À la fin tu reçois un rapport clair listant les points qui doivent être corrigés avant la fusion, ceux qui peuvent attendre, et ceux qui sont déjà bons. C'est l'étape « passage à l'inspection » avant de livrer.

Source

- **Plugin** : `gstack`
- **Nom interne** : `review`
- **Fichier** : `/home/thymon/.claude/skills/gstack/review/SKILL.md`

investigate

investigate

> *Catalogue généré le 2026-05-11*

En une phrase

Une méthode rigoureuse pour débugger en quatre phases : Claude refuse de réparer un bug sans avoir d'abord trouvé sa cause profonde.

Quand l'utiliser

- Quand un truc « marchait hier et ne marche plus » et que tu ne comprends pas pourquoi
- Quand tu vois une erreur 500, un message d'erreur ou un comportement inattendu
- Quand tu as déjà essayé deux ou trois réparations rapides et que ça ne tient pas
- Quand le bug semble revenir à chaque fois, comme un jeu de taupes
- Avant de fermer un ticket de bug, pour t'assurer que la vraie cause a été traitée

Comment l'invoquer

- **Slash command** : `/investigate` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "debug this" / "fix this bug" / "why is this broken" / "investigate this error" / "root cause analysis"
- **Auto-invocation** : □ Oui — Claude invoque ce skill automatiquement quand tu signales une erreur, une stack trace, un comportement bizarre, ou un dysfonctionnement.

Description détaillée

`/investigate` applique une règle stricte appelée « Iron Law » : pas de correctif sans cause profonde identifiée. C'est la différence entre éteindre la flamme et trouver la fuite de gaz. Claude suit quatre phases : enquête (lire les symptômes, les logs, les changements récents), analyse (matcher avec des patterns connus comme race condition, cache obsolète, null propagé), hypothèse (vérifier l'idée avant d'écrire du code), implémentation (le correctif minimal + un test de régression).

Le skill verrouille aussi automatiquement le périmètre de modification au dossier concerné par le bug, pour t'éviter de changer du code non lié. Il vérifie les enseignements (« learnings ») accumulés par Claude sur tes projets précédents — si tu as déjà rencontré un bug similaire, il s'en sert.

Règle des trois échecs : si trois hypothèses successives sont fausses, Claude s'arrête et te demande s'il faut continuer, escalader, ou ajouter du logging et observer. À la fin tu obtiens un rapport structuré (symptôme, cause, correctif, preuve que ça marche, test de non-régression). C'est plus lent que « répare-moi ça vite » mais ça évite les bugs qui reviennent.

Source

- **Plugin** : `gstack`
- **Nom interne** : `investigate`
- **Fichier** : `/home/thymon/.claude/skills/gstack/investigate/SKILL.md`

codex

codex

> *Catalogue généré le 2026-05-11*

En une phrase

Demande un second avis à une IA concurrente (OpenAI Codex) — version « développeur 200 QI direct et sans filtre » qui challenge ton code ou tes idées.

Quand l'utiliser

- Quand Claude a fini un truc important et tu veux qu'une autre IA le critique
- Quand tu veux qu'une seconde IA essaie volontairement de casser ton code (mode adversarial)
- Quand tu hésites sur une décision technique et tu veux entendre un autre point de vue
- Quand tu suspectes que Claude est trop d'accord avec toi et tu veux quelqu'un de plus tranchant
- Pour les décisions importantes où l'accord entre deux IA renforce ta confiance

Comment l'invoquer

- **Slash command** : `/codex` (à taper dans Claude Code)
- **Voice triggers** : « code x » · « code ex » · « get another opinion »
- **Phrases déclencheurs (texte)** : "codex review" / "codex challenge" / "ask codex" / "second opinion" / "consult codex"
- **Auto-invocation** : Sur demande explicite (tu dois invoquer toi-même)

Description détaillée

`/codex` est un pont vers la CLI d'OpenAI Codex, présenté comme « le développeur 200 QI un peu autiste » : direct, terse, techniquement précis, qui challenge les hypothèses. Le skill fonctionne en trois modes. **Review** lance une relecture de code indépendante sur ta « diff » actuelle, avec un verdict pass/fail. **Challenge** met Codex en mode adversarial : il essaie activement de casser ton code, de trouver les cas limites, les bugs subtils. **Consult** te laisse poser n'importe quelle question avec continuité de session pour les suivis.

Si tu tapes juste `/codex` sans argument, Claude détecte ce qui est pertinent : s'il y a une diff il propose review ou challenge, s'il y a un fichier de plan il propose de le relire, sinon il te demande ce que tu veux explorer. Tu peux aussi forcer un niveau de raisonnement plus profond avec `--xhigh`.

La sortie est présentée fidèlement, pas résumée — c'est l'avis brut de Codex que tu vois. C'est utile quand l'accord entre Claude et Codex donne plus de confiance, et quand leur désaccord signale une décision où ton jugement humain est important. Tu dois avoir installé `codex` (`npm install -g @openai/codex`) et être authentifié.

Source

- **Plugin** : `gstack`
- **Nom interne** : `codex`
- **Fichier** : `/home/thymon/.claude/skills/gstack/codex/SKILL.md`

health

health

> *Catalogue généré le 2026-05-11*

En une phrase

Un tableau de bord de la santé de ton code : Claude lance tous les outils de qualité du projet et te donne une note globale sur 10.

Quand l'utiliser

- Quand tu veux savoir en deux minutes si ton projet est en bon état
- Avant une grosse session de refacto, pour avoir un point de référence
- Après une grosse session de refacto, pour vérifier que rien ne s'est dégradé
- Quand tu reprends un projet après plusieurs semaines et tu veux faire le point
- Pour suivre dans le temps l'évolution de la qualité (l'historique est gardé)

Comment l'invoquer

- **Slash command** : `/health` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "health check" / "code quality" / "how healthy is the codebase" / "run all checks" / "quality score"
- **Auto-invocation** : Sur demande explicite

Description détaillée

`/health` détecte automatiquement les outils de qualité installés dans ton projet : vérificateur de types (TypeScript, ruff, pylint), linter (Biome, ESLint, RuboCop), runner de tests (vitest, jest, pytest, cargo test, go test), détecteur de code mort (knip), linter shell (shellcheck), et état de gbrain si tu l'utilises. La première fois, il te propose de sauvegarder cette configuration dans `CLAUDE.md` pour ne pas avoir à redétecter ensuite.

Le skill lance chaque outil indépendamment, capture la sortie, le code de retour et la durée. Chaque catégorie reçoit une note de 0 à 10 selon un barème clair (typecheck propre = 10, plus de 50 erreurs = 0, etc.). Une formule pondérée calcule ensuite une note composite : les tests pèsent 28%, le typecheck 22%, le lint 18%, le code mort 13%, le lint shell 9%, gbrain 10%.

Tu obtiens un tableau de bord lisible avec le score, le statut (CLEAN / WARNING / NEEDS WORK / CRITICAL), la durée et les détails des problèmes principaux. Chaque exécution est archivée dans `~/.gstack/projects/<projet>/health-history.jsonl` pour que Claude puisse te montrer les tendances et recommander où concentrer tes efforts.

Source

- **Plugin** : `gstack`
- **Nom interne** : `health`
- **Fichier** : `/home/thymon/.claude/skills/gstack/health/SKILL.md`

careful

careful

> *Catalogue généré le 2026-05-11*

En une phrase

Mode « prudence » : Claude t'avertit avant chaque commande dangereuse (`rm -rf`, `DROP TABLE`, `git push --force`, etc.) pour que tu puisses confirmer ou annuler.

Quand l'utiliser

- Quand tu travailles sur de la production ou un serveur partagé
- Quand tu débugges un système en direct et tu ne veux pas effacer un fichier important par mégarde
- Quand tu vas exécuter des migrations de base de données et tu veux un filet de sécurité
- Quand tu n'es pas sûr de toi et tu préfères que Claude te demande deux fois plutôt qu'une
- Pour activer juste les avertissements (sans le verrouillage de dossier de `/guard`)

Comment l'invoquer

- **Slash command** : `/careful` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "be careful" / "safety mode" / "prod mode" / "careful mode" / "warn before destructive"
- **Auto-invocation** : Sur demande explicite

Description détaillée

`/careful` installe un garde-fou qui intercepte chaque commande Bash que Claude veut exécuter, avant qu'elle ne tourne. Si la commande correspond à un motif dangereux, Claude s'arrête et te demande confirmation avec un message explicite. Tu peux toujours autoriser, mais tu ne peux pas le faire à ton insu.

Voici les motifs qui déclenchent un avertissement : suppression récursive (`rm -rf`, `rm -r`, `rm --recursive`), suppression de table ou de base SQL (`DROP TABLE`, `DROP DATABASE`, `TRUNCATE`), réécriture d'historique git (`git push --force`, `git push -f`, `git reset --hard`), perte de modifications non commitées (`git checkout .`, `git restore .`), suppressions Kubernetes (`kubectl delete`), nettoyage Docker (`docker rm -f`, `docker system prune`).

Quelques cas évidents et fréquents sont autorisés sans alerte parce qu'ils sont sûrs : `rm -rf node_modules`, `.next`, `dist`, `__pycache__`, `.cache`, `build`, `.turbo`, `coverage`. Pas la peine de t'embêter pour vider un dossier de build.

Techniquement, le skill fonctionne via un « hook » PreToolUse de Claude Code qui lit la commande, la compare aux motifs ci-dessus, et renvoie une demande de permission si nécessaire. Le hook est actif tant que la session dure — pour le désactiver, tu termines la conversation ou tu en commences une nouvelle.

Source

- **Plugin** : `gstack`
- **Nom interne** : `careful`
- **Fichier** : `/home/thymon/.claude/skills/gstack/careful/SKILL.md`

devex-review

devex-review

> *Catalogue généré le 2026-05-11*

En une phrase

Audit en direct de l'expérience développeur : Claude teste vraiment ton produit (CLI, API, docs) au lieu de juste lire le plan — il chronomètre, clique, et fait des captures.

Quand l'utiliser

- Après avoir livré une feature destinée aux développeurs, pour mesurer la réalité plutôt que la théorie
- Quand tu veux savoir combien de temps prend vraiment ton « hello world »
- Quand tu veux vérifier si les messages d'erreur sont compréhensibles, avec captures à l'appui
- Pour comparer ce que `/plan-devex-review` avait prédit (« 3 minutes pour démarrer ») avec la réalité (« 8 minutes en vrai »)
- Avant de publier ou promouvoir un outil pour développeurs

Comment l'invoquer

- **Slash command** : `/devex-review` (à taper dans Claude Code)
- **Voice triggers** : « dx audit » · « test the developer experience » · « try the onboarding » · « developer experience test »
- **Phrases déclencheurs (texte)** : "test the DX" / "DX audit" / "developer experience test" / "try the onboarding" / "live dx audit"

- **Auto-invocation** : □ Oui — Claude propose ce skill après avoir livré une feature destinée aux développeurs.

Description détaillée

`/devex-review` est la version « test sur le terrain » de `/plan-devex-review`. Au lieu de relire un plan, Claude utilise l'outil `browse` (un navigateur sans interface) pour ouvrir vraiment ta documentation, suivre le tutoriel d'installation, essayer le premier exemple, déclencher volontairement des erreurs pour voir ce qu'il se passe. Il prend des captures d'écran et chronomètre chaque étape.

Le skill applique les mêmes huit principes DX que la version planning (zéro friction au démarrage, étapes incrémentales, défauts opinionnés avec échappatoires, etc.) et les mêmes sept dimensions de scoring (Utilisable, Crédible, Trouvable, Utile, Précieux, Accessible, Désirable). Mais cette fois les notes sont basées sur l'observation directe, pas sur l'analyse du plan.

`browse` peut tester les surfaces accessibles via le web : pages de docs, playgrounds d'API, dashboards web, flux d'inscription, tutoriels interactifs, pages d'erreur. Il ne peut pas tester ce qui demande un terminal local, une vraie adresse mail, ou des credentials réels — pour ces parties, Claude te dit clairement ce qui reste à vérifier à la main.

Le rapport final compare aussi tes scores réels avec ceux prédits par `/plan-devex-review` si tu l'avais lancé avant. C'est le « boomerang » qui ramène les promesses du plan face à la réalité du produit livré.

Source

- **Plugin** : `gstack`
- **Nom interne** : `devex-review`
- **Fichier** : `/home/thymon/.claude/skills/gstack/devex-review/SKILL.md`