gstack *mémo*

Les commandes à connaître pour utiliser **GStack**, la suite de skills de Garry Tan qui fait jouer à Claude Code (ou Codex) les rôles d'une équipe produit : CEO, designer, eng manager, QA, release manager.

Famille Méthodes de développement · **Éditeur** Garry Tan · **Licence** MIT
Note du wiki 10/15 · **Commandes vérifiées le** 25 septembre 2026
Fiche, notes et avis aitoolswiki.viellard.fr/outils/methodes-de-developpement/gstack

Installation

`git clone --single-branch --depth 1 https://github.com/garrytan/gstack.git ~/.claude/skills/gstack && cd ~/.claude/skills/gstack && ./setup`	Claude Code (installation globale)
`./setup --host codex`	Codex, Cursor, OpenCode… : cibler l'agent
`(cd ~/.claude/skills/gstack && ./setup --team) && ~/.claude/skills/gstack/bin/gstack-team-init required`	Mode équipe : les coéquipiers récupèrent gstack automatiquement dans ce repo

Prérequis : Git, Bun ≥ 1.0. Sur macOS, le navigateur Aside est recommandé pour les skills de navigation.

Démarrage rapide

Le README conseille d'essayer ces quatre commandes, puis de s'arrêter pour juger :

1. `/office-hours` – décrire ce qu'on construit
2. `/plan-ceo-review` – sur une idée de fonctionnalité
3. `/review` – sur une branche avec des changements
4. `/qa <url>` – sur une URL de staging

Le cycle complet

Réfléchir → Planifier → Construire → Relire → Tester → Livrer → Rétrospective. Chaque commande produit un document que la suivante lit (design doc, plan de test…).

Réfléchir et planifier

`/office-hours`	Point de départ. Six questions qui recadrent l'idée avant d'écrire du code, remet en cause les hypothèses, propose des alternatives et écrit un design doc.
`/plan-ceo-review`	Revue produit : cherche la meilleure version du besoin. Quatre modes : élargir, élargir de façon ciblée, garder le périmètre, réduire.
`/plan-eng-review`	Revue d'architecture : flux de données, diagrammes, cas limites, plan de tests.
`/plan-design-review`	Note chaque dimension du design de 0 à 10 et modifie le plan pour atteindre 10. Détecte le « AI slop ».
`/plan-devex-review`	Revue de l'expérience développeur (API, CLI, SDK, docs).
`/autoplan`	Enchaîne automatiquement les revues CEO → design → DX → eng et ne remonte que les décisions de goût.
`/spec`	Transforme une intention floue en spécification exécutable, en cinq phases.

Le cycle complet (suite)

Relire et déboguer

`/review`	Revue de code « staff engineer » : trouve les bugs qui passent la CI, corrige les évidents, signale le code trop complexe.
`/investigate`	Débogage par cause racine : pas de correctif sans enquête, arrêt après 3 échecs. Verrouille les modifications au module étudié.
`/cso`	Audit de sécurité (OWASP, STRIDE) avec modèle de l'application et findings argumentés.
`/codex`	Deuxième avis : revue indépendante par OpenAI Codex (depuis Claude Code).
`/claude-code`	Deuxième avis par Claude Code (depuis Codex ou un autre agent).
`/deslop-shared-libs`	Repère le code dupliqué qui mériterait une bibliothèque partagée. Recommandations uniquement.

Tester dans un vrai navigateur

`/qa <url>`	Teste l'application dans un navigateur, corrige les bugs par commits atomiques et ajoute un test de non-régression pour chacun.
`/qa-only <url>`	Même méthode, mais rapport de bugs seul, sans modifier le code.

Le cycle complet (suite)

`/browse`	Donne des « yeux » à l'agent : navigation, clics, captures d'écran.
`/design-review`	Audit visuel de l'application en ligne, puis correctifs avec captures avant/après.
`/benchmark`	Mesure temps de chargement et Core Web Vitals, compare avant/après.
`/setup-browser-cookies`	Importe les cookies de ton navigateur pour tester des pages authentifiées.

Livrer et suivre

`/ship`	Synchronise main, lance les tests, audite la couverture, pousse et ouvre la PR. Met aussi la doc à jour.
`/land-and-deploy`	Merge la PR, attend la CI et le déploiement, vérifie la production. Configurer d'abord avec `/setup-deploy`.
`/canary`	Surveillance après déploiement : erreurs console, régressions de performance.
`/document-release`	Met à jour README, ARCHITECTURE, CLAUDE.md… selon ce qui vient d'être livré.
`/retro`	Rétrospective hebdomadaire ; `/retro global` couvre tous les projets et agents.

Le cycle complet (suite)

Design

`/design-consultation`	Construit un design system complet et écrit `DESIGN.md`.
`/design-shotgun`	Génère 4 à 6 variantes de maquettes, les compare dans le navigateur et itère selon tes retours.
`/design-html`	Transforme une maquette validée en HTML de production (détecte React, Svelte, Vue).

Garde-fous et outils

`/careful`	Avertit avant toute commande destructrice (`rm -rf`, `DROP TABLE`, force-push). S'active aussi en disant « be careful ».
`/freeze` / `/unfreeze`	Restreint (ou libère) les modifications à un seul dossier.
`/guard`	`/careful` + `/freeze` en une commande, pour le travail en production.
`/learn`	Consulte et nettoie ce que gstack a retenu du projet entre les sessions.
`/make-pdf`	Markdown vers PDF (ou HTML, DOCX) de qualité publication, diagrammes inclus.
`/diagram`	Description en anglais vers diagramme éditable (Mermaid, Excalidraw, SVG).
`/gstack-upgrade`	Met gstack à jour.

Quelle revue choisir ?

Tu construis pour…	Avant le code	Après livraison
Des utilisateurs finaux (UI, web, mobile)	`/plan-design-review`	`/design-review`
Des développeurs (API, CLI, SDK)	`/plan-devex-review`	`/devex-review`
L'architecture (données, perf, tests)	`/plan-eng-review`	`/review`
Tout à la fois	`/autoplan`	–

Conseils d'utilisation

- **Commencer petit.** Un utilisateur rapporte qu'après un mois il n'utilisait plus que 6 commandes sur 35. Les plus citées : `/qa`, `/review`, `/ship`, `/learn`, `/plan-eng-review`, `/investigate`.
- **Pour une fonctionnalité de bout en bout** : `/office-hours` → `/autoplan` → implémentation → `/review` → `/qa` → `/ship`.
- **Planifier sans coder** : demander `/office-hours` puis `/autoplan`, en précisant de sauvegarder le plan sans l'implémenter.
- **Adapter plutôt que tout adopter** : `/plan-ceo-review` et les commandes de design apportent peu sur un outil interne ou un projet B2B contraint.
- **Vérifier `/ship` avant de l'adopter en équipe** : il suit un workflow git précis (sync main, push, PR) qui peut ne pas correspondre au vôtre.
- **Sécurité** : lancer `/cso` sur tout changement qui touche aux identifiants, clés API ou validations d'entrée.
- **Langage naturel** : les skills se déclenchent aussi avec des phrases comme « run a security check » ou « test the website ».
- **Travail parallèle** : avec plusieurs sessions (via Conductor ou des worktrees), le cycle structuré évite que chaque agent parte dans une direction différente.

Sources (documentation officielle)

- README gstack
- Détail de chaque skill (docs/skills.md)
- Critique après un mois d'usage – The Claude Codex
- gstack in Practice – Shane Hartman