

GUÍA NODESTUDIO

G Stack

Ya lo instalaste. Ahora qué comandos usás, en qué orden y para qué.

⏱ Instalación ~5 min · Lectura 10–15 min

1 Qué es G Stack

G Stack es el setup de Claude Code que Garry Tan, CEO de Y Combinator, publicó con licencia MIT. Son archivos Markdown que se instalan como *slash commands* y le dan a Claude Code roles definidos: producto, arquitectura, diseño, revisión de código, QA, seguridad y release.

El problema que ataca: dejar de guiar a Claude paso por paso en cada feature. El proceso queda escrito y cada skill le pasa su resultado a la siguiente — `/office-hours` escribe un design doc que lee `/plan-ceo-review`, `/plan-eng-review` escribe un plan de tests que levanta `/qa`.

REPO https://github.com/garrytan/gstack

La terminología del repo (usala bien)

Término	Qué es
skills	Cómo el README llama a todo: archivos Markdown que se instalan como slash commands (`/review` , `/qa`).
specialists	Las skills del sprint, cada una con un rol: Office Hours, CEO, Eng Manager, Designer, Staff Engineer, QA Lead, CSO, Release Engineer.
power tools	Utilidades sueltas que no son etapa del sprint: `/careful` , `/freeze` , `/guard` , `/codex` , `/gstack-upgrade` .
standalone binaries	CLIs que corren fuera de la sesión: `gstack-analytics` , `gstack-egress` , `gstack-evidence` .

LOS NÚMEROS, VERIFICADOS HOY

El repo se presenta como **23 especialistas + 8 power tools**. Clonando la versión actual (`VERSION 1.87.4.0`) aparecen **54 carpetas con SKILL.md** instalables: se sumaron iOS, documentación, browser y setup. Si el número del reel no coincide, mandá el del repo.

QUÉ DICE GARRY QUE USA ÉL, Y QUÉ SOLO VIENE EN EL REPO

Él afirma que lo usa todos los días, que `/qa` fue el desbloqueo que lo llevó de 6 a 12 agentes en paralelo, y que corre 10–15 sprints simultáneos con Conductor. El resto (iOS QA, GBrain, `/pair-agent` , los bridges opcionales) está en el repo, pero no afirma que sea parte de su día a día.

Y no: no reemplaza a un CEO, un diseñador ni un ingeniero. Es proceso escrito en prompts. Las decisiones siguen siendo tuyas.

2 Quick Start

El propio README propone una prueba corta antes de casarte con la herramienta. Es buena: con esto sabés en media hora si te sirve.

```
instalar → /office-hours → /plan-ceo-review → /review → /qa → parar
```

#	Hacés	Qué estás probando
1	Pegás el prompt de la página siguiente	Que instale y registre las skills. Termina cuando Claude te lista los comandos disponibles.
2	`/office-hours` y describís qué estás construyendo	Si te sirve que te discutan el problema antes de escribir código. Deja un design doc escrito.
3	`/plan-ceo-review` sobre esa idea	Si el reencuadre de alcance te aporta o te molesta. Acá se decide si G Stack es para vos.
4	`/review` en una rama con cambios reales	La parte más rentable: qué encuentra en código que ya diste por bueno.
5	`/qa https://tu-staging`	Que abra un navegador de verdad, clickee los flujos y te traiga bugs con evidencia.

PROBALO SOBRE ALGO REAL

El paso 4 sobre un repo de juguete no te dice nada. Corré `/review` en una rama con cambios que pensabas mergear hoy; ahí se ve si el filtro sirve o hace ruido.

ANTES DE ARRANCAR

Necesitás **Claude Code**, **Git** y **Bun 1.0+**. En Windows, además **Node.js** (Bun tiene un bug con Playwright ahí). El instalador chequea Bun y corta con un error claro si no está.

3 Instalación

No hace falta que toques la terminal. Abrí Claude Code en cualquier carpeta y pegale esto — el link del repo va adentro del prompt, así funciona aunque el prompt viaje solo:

REPO https://github.com/garrytan/gstack

PROMPT PARA PEGAR EN CLAUDE CODE

```
Instalá gstack en mi máquina: corré git clone --single-branch --depth 1
https://github.com/garrytan/gstack.git ~/.claude/skills/gstack && cd
~/.claude/skills/gstack && ./setup y esperá a que termine el build. Después agregá una
sección "gstack" a mi CLAUDE.md que diga que uses la skill /browse de gstack para todo
el browsing web y nunca las tools mcp__claude-in-chrome__*, y que liste las skills que
quedaron instaladas. Al final mostrame la lista de slash commands disponibles y
preguntame si quiero agregar gstack también al proyecto actual para que lo tenga el
equipo.
```

Claude Code corre los comandos por vos y te avisa cuando termina. El repo dice "30 segundos": es cierto para el clone, pero `./setup` además compila un Chromium propio con Playwright, así que contá unos minutos la primera vez.

Qué te conviene saber del instalador

- Instala en `~/.claude/skills/gstack` y registra las skills en Claude Code. Los comandos quedan cortos por defecto (`/review`); con `./setup --prefix` pasan a `/gstack-review` si ya usás otros packs de skills.
- Registra hooks en `~/.claude/settings.json` y un chequeo de auto-update al inicio de cada sesión (una vez por hora, silencioso).
- En equipo: `./setup --team` deja el repo configurado para que tus compañeros lo reciban solo. Pedíselo a Claude si te sirve; para vos solo, no hace falta.
- En macOS, si tenés el navegador Aside abierto, las skills de browser usan tus sesiones reales. Si no, usan el Chromium que compiló el setup. Las dos rutas funcionan.

DESINSTALAR

Hay script propio: pedile a Claude que corra `~/.claude/skills/gstack/bin/gstack-uninstall` . Borrar la carpeta a mano deja los hooks colgados en `settings.json` tirando error en cada sesión. Está documentado en el README, y es la clase de cosa que se descubre tarde.

4 Los comandos que valen la pena — planificar

De las 54 skills instaladas, estas nueve cubren casi todo. Las cuatro de esta página se usan antes de escribir código.

`/office-hours` YC OFFICE HOURS

Seis preguntas forzadas que reencuadran el producto antes de que exista. Te discute el planteo en vez de aceptarlo.

Cuándo: Tenés una idea o un problema difuso y todavía no hay nada escrito.

Le das: Una descripción del dolor real, con ejemplos concretos — no hipótesis.

Esperás: Un design doc guardado que las demás skills leen después.

Ejemplo: `/office-hours`

`/plan-ceo-review` CEO / FOUNDER

Repiensa el problema y busca el producto 10x escondido en el pedido. Cuatro modos: Expansion, Selective Expansion, Hold Scope, Reduction.

Cuándo: Ya tenés un plan y querés saber si estás resolviendo lo correcto.

Le das: El plan o el design doc de `/office-hours`.

Esperás: Una revisión de 10 secciones con el alcance desafiado y una recomendación explícita.

Ejemplo: `/plan-ceo-review`

`/plan-eng-review` ENG MANAGER

Cierra la arquitectura: flujo de datos, diagramas ASCII, máquinas de estado, casos borde, matriz de tests y modos de falla.

Cuándo: El qué está definido y falta el cómo. Es la última puerta antes de codear.

Le das: El plan ya revisado. Trabaja en plan mode.

Esperás: Plan cerrado, supuestos ocultos explicitados y un plan de tests que después levanta `/qa`.

Ejemplo: `/plan-eng-review`

`/autoplan` PIPELINE DE REVISIÓN

Corre CEO → diseño → DX → ingeniería de corrido con decisiones automáticas, y te sube solo las decisiones de criterio.

Cuándo: Querés el plan revisado entero sin contestar 20 preguntas intermedias.

Le das: Un archivo de plan.

Esperás: Plan revisado por las cuatro miradas, con un único punto de aprobación al final.

Ejemplo: `/autoplan`

4 Los comandos que valen la pena — construir y enviar

Estas cinco se usan con código ya escrito. Son las que más rinden el primer día.

`/review` STAFF ENGINEER

Analiza el diff contra la rama base buscando bugs que pasan CI y explotan en producción: seguridad SQL, side effects condicionales, problemas estructurales.

Cuándo: Antes de mergear cualquier rama.

Le das: Una rama con cambios. Nada más.

Esperás: Arregla solo lo obvio, marca el resto para que decidas, y avisa dónde quedó código incompleto.

Ejemplo: `/review`

`/qa` QA LEAD

Abre un navegador real, recorre los flujos, encuentra bugs, los arregla con commits atómicos y vuelve a verificar. Genera un test de regresión por cada fix.

Cuándo: La feature ya está levantada en algún lado y querés saber si funciona de verdad.

Le das: Una URL que funcione (staging o local). Tres niveles: Quick, Standard, Exhaustive.

Esperás: Health score antes/después, capturas y un resumen de si está para enviar.

Ejemplo: `/qa https://staging.miapp.com`

`/qa-only` QA REPORTER

La misma metodología que `/qa` pero sin tocar una línea: reporte puro con pasos de reproducción.

Cuándo: Estás auditando código ajeno o no querés commits automáticos.

Le das: Una URL.

Esperás: Un reporte de bugs con evidencia y severidad.

Ejemplo: `/qa-only https://staging.miapp.com`

`/investigate` DEBUGGER

Debugging de causa raíz con una regla dura: no se arregla nada sin investigar antes. Traza el flujo de datos, prueba hipótesis y frena a los 3 intentos fallidos.

Cuándo: Hay un bug que no entendés, o Claude ya intentó dos parches que no funcionaron.

Le das: El síntoma y cómo reproducirlo.

Esperás: Causa raíz identificada antes del fix. Congela las ediciones al módulo investigado.

Ejemplo: `/investigate`

`/ship` RELEASE ENGINEER

Sincroniza main, corre los tests, audita cobertura, bumpea VERSION, actualiza el CHANGELOG, commitea, pushea y abre el PR.

Cuándo: El código pasó `/review` y `/qa`.

Le das: La rama lista. Si no hay framework de tests, lo arma.

Esperás: PR abierto con auditoría de cobertura. Si no querés que toque VERSION o CHANGELOG, decíselo antes.

Ejemplo: `/ship`

5 Flujo para un proyecto desde cero

El README lo llama "el sprint": pensar → planificar → construir → revisar → testear → enviar → reflexionar. Estos son los comandos reales de cada etapa.

```
/office-hours → /autoplan → aprobar → construir → /review
      → /qa → /ship → /land-and-deploy → /retro
```

Etapa	Comando	Qué pasa
Idea	`/office-hours`	Te discuten el problema y sale un design doc. Si la idea ya está clara, saltealo.
Producto y plan	`/autoplan`	Corre CEO, diseño, DX e ingeniería sobre el plan. Ingeniería va última, para revisar el plan ya corregido.
Decisión	vos	Aprobás el plan y salís de plan mode. Esta etapa no es delegable.
Build	Claude Code normal	No hay comando de G Stack acá: implementa contra el plan aprobado.
Review	`/review`	Diff contra la base. Autofix de lo obvio, el resto marcado.
QA	`/qa <url>`	Navegador real sobre la app levantada, con tests de regresión por cada fix.
Seguridad	`/cso`	Opcional pero barato: OWASP + STRIDE con escenario de explotación por hallazgo.
Ship	`/ship`	Tests, cobertura, CHANGELOG y PR. Dispara `/document-release` solo.
Deploy	`/land-and-deploy`	Mergea, espera CI y deploy, verifica producción. Requiere `/setup-deploy` una vez.
Reflexión	`/retro`	Retro semanal con salud de tests y rachas. Útil recién después de varias semanas.

SI EL PROYECTO TIENE INTERFAZ

Entre plan y build entran `/design-consultation` (arma el sistema de diseño y escribe `DESIGN.md`), `/design-shotgun` (4–6 variantes de mockup para elegir) y `/design-html` (convierte el mockup elegido en HTML de producción). Es la cadena que más rinde si no tenés diseñador.

6 Flujo sobre un proyecto que ya anda

Ya tenés la app funcionando y querés agregar una feature. La secuencia cambia: no arrancás desde la idea, arrancás desde código que no querés romper.

```
describir la feature → /autoplan → construir → /review → /qa → /ship → /land-and-deploy
```

La primera vez en ese repo (una sola vez)

- `/cso` — línea de base de seguridad sobre lo que ya existe. Te dice con qué estás conviviendo.
- `/setup-deploy` — detecta plataforma, URL de producción y comandos de deploy. Sin esto, `/land-and-deploy` no corre.
- `/learn` — gestiona lo que G Stack va aprendiendo del repo (patrones, trampas, preferencias) para que las sesiones siguientes arranquen con contexto.

Según qué estés tocando

Estás construyendo para...	Antes de codear	Auditoría en vivo
Usuarios finales (UI, web, mobile)	`/plan-design-review`	`/design-review`
Desarrolladores (API, CLI, SDK, docs)	`/plan-devex-review`	`/devex-review`
Arquitectura (datos, performance, tests)	`/plan-eng-review`	`/review`
Todo junto	`/autoplan`	—

Si lo que tenés es un bug, no una feature

El flujo es otro y es más corto: `/investigate` para encontrar la causa raíz → el fix → `/review` → `/qa` sobre el flujo afectado → `/ship`. Saltearse `/investigate` y pedir el fix directo es cómo se acumulan tres parches que no arreglan nada.

> **FEATURE CHICA**
> Para un cambio de copy, un campo nuevo o un fix de una línea, `/autoplan` es pérdida de tiempo y de tokens. Pedí el cambio normal y corré `/review` antes de mergear. Nada más.

7 Mi stack mínimo

Si tuviera que quedarme con cinco comandos y ignorar los otros 49, serían estos. Dan el 80% del valor sin cambiarte la forma de trabajar.

`/review` EL DE MEJOR RELACIÓN ESFUERZO / RESULTADO

Cero configuración, cero contexto que darle, y encuentra cosas en código que ya diste por bueno. Si vas a usar un solo comando de G Stack, es este.

`/qa` LO QUE NINGÚN LINTER TE DA

Un navegador real recorriendo tu app. Es la diferencia entre "compila" y "funciona". Necesita una URL levantada, y eso es todo lo que pide.

`/investigate` PARA LOS BUGS QUE SE RESISTEN

La regla de no arreglar sin investigar es la que te ahorra la tarde entera de parches a ciegas. Frena a los 3 intentos fallidos en vez de seguir probando.

`/ship` CIERRA EL CICLO

Tests, cobertura, CHANGELOG y PR en un comando. Si tu proyecto no tiene framework de tests, lo arma. Acá es donde el proceso se vuelve hábito.

`/office-hours` SOLO CUANDO LA IDEA NO ESTÁ CLARA

No lo corras en cada feature. Corralo cuando estés por construir algo que todavía no sabés si vale la pena. Ahí te ahorra semanas, no minutos.

EL SEXTO, SI TRABAJÁS SOBRE PRODUCCIÓN

`/guard` — activa `/careful` (avisa antes de `rm -rf`, `DROP TABLE` o force-push) y `/freeze` (limita las ediciones a un solo directorio) juntos. Es la red de seguridad para cuando delegás con la base de datos real a mano.

8 Ejemplo completo

Feature: **agregar login con Google a un SaaS que ya está en producción.** Así la recorrería de punta a punta. Comando → para qué → qué pasa después.

Comando	Para qué	Qué pasa después
`/plan-ceo-review`	Chequear el alcance: ¿solo Google, o el paso real es un sistema de identidad con varios proveedores?	Sale una recomendación explícita de alcance. Vos elegís si expandís o sostenés.
`/plan-eng-review`	Cerrar el cómo: OAuth, sesiones, qué pasa con las cuentas que ya existen con el mismo mail.	Plan con diagramas de flujo, casos borde y matriz de tests. Acá aparece el caso de la cuenta duplicada, que es el que rompe.
aprobar el plan	Decidir el merge de cuentas: ¿vinculás automático o pedís confirmación?	Decisión tuya. Es producto, no implementación.
Claude Code normal	Implementar contra el plan aprobado.	Código escrito, sin PR todavía.
`/review`	Buscar lo que pasa CI y explota después: tokens en logs, redirects sin validar, sesión mal invalidada.	Arregla lo obvio solo, te marca el resto para decidir.
`/cso`	Auth es la superficie de ataque por excelencia. OWASP + STRIDE sobre el flujo nuevo.	Hallazgos con escenario de explotación concreto, no advertencias genéricas.
`/qa <staging>`	Recorrer el flujo real: login, cancelar a mitad, usuario ya existente, logout, volver a entrar.	Bugs arreglados con commit atómico y un test de regresión por cada uno.
`/ship`	Tests, cobertura, CHANGELOG y PR.	PR abierto, docs actualizadas solas.
`/land-and-deploy`	Mergear, esperar CI y deploy, verificar que producción responde.	Feature en producción, verificada. Si algo se degrada, `/canary` lo mira un rato.

Nueve pasos, dos decisiones tuyas: el alcance y el merge de cuentas. El resto es proceso.

9 Errores que evitaría

CORRER TODOS LOS AGENTES PORQUE SÍ

`/autoplan` son cuatro revisiones encadenadas. Sobre un cambio de copy es tiempo, tokens y una discusión de alcance que no pediste. El propio repo enruta revisiones según el cambio: copiale el criterio.

CONFUNDIR REVISIÓN AUTOMÁTICA CON GARANTÍA

`/review` arregla lo obvio y marca el resto — eso es un revisor, no una prueba de que el código está bien. Y `/qa` encuentra bugs en los flujos que recorre; los que no recorrió siguen ahí. Ausencia de hallazgos no es lo mismo que ausencia de bugs.

ACEPTAR TODO LO QUE PROPONE `/PLAN-CEO-REVIEW`

Uno de sus cuatro modos es Expansion: está diseñado para encontrar el producto más grande escondido en tu pedido. Sirve para pensar. Si decís que sí a todo, terminás construyendo el triple de lo que ibas a construir. Los modos Hold Scope y Reduction existen por algo.

PEDIR `/QA` SIN APP LEVANTADA

Necesita una URL que responda. Sin staging ni servidor local no hay QA — y es el comando que más se pide de más, porque suena a que testea el código.

CORRER `/SHIP` EN UN REPO AJENO SIN MIRAR

Bumpea VERSION, edita el CHANGELOG, commitea y abre PR. En tu proyecto es lo que querés; en el repo de un cliente con convenciones propias, avisale antes qué puede tocar.

INSTALAR SIN SABER QUÉ TOCA

Agrega hooks en `~/.claude/settings.json`, un auto-update por sesión y secciones en tu `CLAUDE.md`. Todo documentado y reversible — pero si después borrás la carpeta a mano en vez de usar `gstack-uninstall`, te quedan hooks tirando error en cada sesión.

10 Cheat Sheet

Para dejar abierta al lado de Claude Code.

Si querés...	Usá
Saber si vale la pena construir algo	`/office-hours`
Definir bien una feature	`/plan-ceo-review`
Revisar arquitectura antes de codear	`/plan-eng-review`
Revisar el plan de diseño / UX	`/plan-design-review`
Todas las revisiones de plan de una	`/autoplan`
Revisar el código antes de mergear	`/review`
Una segunda opinión de otro modelo	`/codex`
Probar la interfaz en un navegador real	`/qa <url>`
Un reporte de bugs sin que toque el código	`/qa-only <url>`
Auditar el diseño de una página ya publicada	`/design-review`
Entender por qué algo está roto	`/investigate`
Una auditoría de seguridad (OWASP + STRIDE)	`/cso`
Ver opciones de diseño antes de codear	`/design-shotgun`
Convertir un mockup en HTML de producción	`/design-html`
Preparar release: tests, cobertura y PR	`/ship`
Mergear, deployar y verificar producción	`/land-and-deploy`
Vigilar producción después del deploy	`/canary`
Actualizar la documentación de lo que enviaste	`/document-release`
Que avise antes de un comando destructivo	`/careful` (o `/guard`)
Que no edite fuera de un directorio	`/freeze` / `/unfreeze`
Ver o limpiar lo que aprendió del repo	`/learn`
Actualizar G Stack	`/gstack-upgrade`

Verificado sobre `garrytan/gstack` VERSION 1.87.4.0, septiembre 2026. El repo se mueve rápido: si un comando no aparece, corré `/gstack-upgrade` y mirá el README.



NodeStudio

Automatizamos procesos de negocio con IA y sistemas a medida.

nodestudioai.com