



KIT DE ARRANQUE

Tu proyecto listo para trabajar con Claude Code: un comando, cuatro reglas y el prompt con que arranco cada cambio.

1. CONTROLES EN CADA COMMIT

Claude escribe rápido, pero no recuerda lo que le dijiste ayer y no revisa su propio trabajo si nadie se lo pide. El kit se encarga de las dos cosas: deja las reglas escritas dentro del proyecto y pone controles que revisan cada cambio por ti.

Desde la carpeta raíz de tu proyecto (JavaScript o TypeScript, con Node 20 o más):

```
~/tu-app
```

```
$ npx devground-init --yes
```

Eso instala devground, un proyecto mío, gratis y de código abierto. Te deja:

- ✓ **Tests listos** con Vitest, y un mínimo de cobertura que sube cada vez que corres `test:coverage`. Si la IA borra tests sin avisar, el chequeo falla.
- ✓ **Linters y formato** con ESLint y Prettier, solo sobre los archivos que cambiaste.
- ✓ **Hooks de git**: antes de cada commit se revisa el código, y el mensaje tiene que decir qué tipo de cambio es (`feat:` , `fix:` , `docs:`).
- ✓ **Reglas para la IA** en `AGENTS.md` , enlazado como `CLAUDE.md` para Claude Code y con los nombres que buscan Cursor, Copilot, Gemini y Codex.

Compruébalo: haz un cambio e intenta `git commit -am "arreglos"` . Lo tiene que rechazar. Con `git commit -am "fix: corrige el botón"` tiene que pasar.

2. REGLAS DE TRABAJO PARA TU AGENTS.MD

devground te deja un `AGENTS.md` con buenas prácticas generales. Yo le agrego siempre estas cuatro reglas, las mismas que tiene el proyecto de elkis.dev. Pégalas al final del archivo:

```
AGENTS.md

## Cómo trabajamos

1. Piensa antes de programar. Si el pedido se puede entender de dos formas,
muéstrame las dos y pregúntame. Si hay una forma más simple de hacerlo,
dímelo.
2. Lo mínimo que resuelve el problema. Nada de funciones que no pedí, ni
abstracciones para algo que se usa una vez.
3. Cambios quirúrgicos. Toca solo lo que el pedido necesita. Si ves código
muerto o algo raro en otro archivo, avísame, pero no lo cambies.
4. Primero el test. Antes de escribir el código, escribe el test que define
qué tiene que pasar, corrélo y muéstrame que falla. Después implementa hasta
que pase.

No des una tarea por terminada sin correr los tests.
```

La cuarta es la que más cambia el resultado. Cuando Claude escribe primero el test, tiene que decidir qué significa "funciona" antes de programar, y tú tienes una forma de comprobarlo que no depende de crearle.

3. EL PROMPT DE ARRANQUE

Cada vez que vas a pedir algo nuevo, en vez de describir el código, describe el resultado y deja que Claude arme el plan contigo:

```
■ ■ ■ claude

Quiero [describe lo que quieres que pase, en palabras de usuario].

Antes de escribir código:

1. Lee el AGENTS.md y los archivos que vas a tocar.
2. Dime en pocas líneas qué entendiste, qué archivos cambiarías y qué podría romperse.
3. Escribe los tests que demuestran que funciona y muéstrame que fallan.

Espera mi visto bueno antes de implementar.
```

Ese "espera mi visto bueno" es tu momento de dirigir. Lees el plan en un minuto, corriges lo que no calza, y recién ahí Claude programa. Es mucho más barato corregir un plan de diez líneas que deshacer cuarenta archivos.

ANTES DE SEGUIR PROGRAMANDO

- ✓ Corriste `npx devground-init --yes` en la raíz.
- ✓ Un commit con el mensaje "arreglos" se rechaza.
- ✓ Tu `AGENTS.md` tiene las cuatro reglas de trabajo.
- ✓ Guardaste el prompt de arranque donde lo tengas a mano.