EcomLabs

PLANTILLA Y MÉTODO

Claude Code vs Codex: el mismo bug, cuatro métricas

Cómo comparar dos agentes de código con tu propio repo: una base con el test que falla, dos ramas iguales, la misma instrucción y cuatro métricas (tiempo, diff, tests sin tocar y costo), con el script para medirlas y el marcador para llenar con tres bugs.

PARA	Quien duda entre Claude Code y Codex y quiere decidir con su propio código
PROBADO CON	Scripts en Node 22.22.3 · git 2.54.0 · Git Bash (Windows 11), con commits simulados; sin ejecutar ningún agente
ACTUALIZADA	11 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/headless · learn.chatgpt.com/docs/non-interactive-mode
ARTÍCULO	ecomlabs.dev/g/versus
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a medir
Paso 01	La base y dos ramas iguales
Paso 02	La instrucción
Paso 03	Las dos corridas
Paso 04	Mide las cuatro métricas
Paso 05	Bugs que ya arreglaste
Plantilla	El marcador

ANTES DE EMPEZAR

Qué vas a medir

El mismo bug, resuelto por Claude Code y por Codex desde el mismo commit y con la misma instrucción. De cada corrida mides **tiempo**, **diff**, **tests sin tocar** y **costo**; lo repites con tres bugs y decides con el marcador. Es un método, no un benchmark: no trae resultados. Los comandos se revisaron contra la documentación oficial de las dos CLI el 11 de noviembre de 2026 y el script se probó en un repo de práctica con commits hechos a mano, sin ejecutar ningún agente. Necesitas git, bash (en Windows, Git Bash) y las dos CLI con sus cuentas.

PASO 01

La base y dos ramas iguales

Confirma en una rama base el test que reproduce el bug y crea un worktree por agente desde ella:

```
git switch -c cmp/base-1
# agrega el test que reproduce el bug, y luego:
git add test/shipping.test.js
git commit -m "test: free shipping from exactly 999 (fails)"
npm test   # tiene que fallar, y por este motivo

mkdir -p ../cmp/runs
git worktree add ../cmp/claude -b cmp/claude-1 cmp/base-1
git worktree add ../cmp/codex -b cmp/codex-1 cmp/base-1
(cd ../cmp/claude && npm ci) && (cd ../cmp/codex && npm ci)
```

- Crea los worktrees tú: según su documentación, `claude --worktree` parte por defecto de la rama principal del remoto y la app de Codex, del `HEAD` de la rama elegida con tus cambios sin confirmar.
- Codex lee `AGENTS.md`; Claude Code lee `CLAUDE.md` y, desde la v2.1.277, `AGENTS.md` si no hay `CLAUDE.md`. Deja un `AGENTS.md` común o ninguno.

PASO 02

La instrucción

Guárdala como `../cmp/prompt.md`. En cada bug cambia solo la descripción:

```
Bug: con un subtotal de exactamente $999 MXN la tienda cobra envío.
El envío es gratis desde $999. Lo reproduce el test
"ships free from exactly $999" en test/shipping.test.js.

Arregla el código para que npm test pase.
No modifiques, borres ni saltes archivos de test/: son la especificación.
No hagas commit: deja los cambios en el árbol de trabajo.
Al terminar, resume en tres líneas qué cambiaste y por qué.
```

PASO 03

Las dos corridas

Una después de la otra, sin intervenir, alternando cuál va primero en cada bug:

```
cd ../cmp/claude
start=$(date +%s)
claude -p "$(cat ../prompt.md)" --permission-mode acceptEdits \
  --allowedTools "Bash(npm test *)" --output-format json > ../runs/claude-1.json
echo "claude-1: $(( $(date +%s) - start )) s" | tee -a ../runs/times.txt
git add -A && git commit -qm "claude: bug 1" --allow-empty

cd ../codex
start=$(date +%s)
codex exec --sandbox workspace-write --json "$(cat ../prompt.md)" > ../runs/codex-1.jsonl
echo "codex-1: $(( $(date +%s) - start )) s" | tee -a ../runs/times.txt
git add -A && git commit -qm "codex: bug 1" --allow-empty
```

- **Claude Code:** en `-p` se rechaza lo que necesita aprobación; `acceptEdits` le deja editar y `Bash(npm test *)` correr `npm test`.
- **Codex:** `codex exec` es de solo lectura por defecto; `--sandbox workspace-write` le deja editar (`--full-auto` ya no está soportado).
- Fija y anota el modelo (`--model` en Claude Code, `-m` en Codex) y la versión de cada CLI.

PASO 04

Mide las cuatro métricas

Diff y tests: guarda `../cmp/agent-metrics.sh` y córrelo desde `../cmp` con `bash agent-metrics.sh cmp/base-1 claude "npm test" test` (y lo mismo con `codex`):

```
#!/usr/bin/env bash
# Diff size, test files touched and the original suite for one agent run.
# usage: bash agent-metrics.sh <base> <dir> "<test command>" <test path>...
set -uo pipefail
[ $# -ge 4 ] || { echo 'usage: agent-metrics.sh <base> <dir> "<test command>" <test path>...' >&2; exit 2; }
base=$1 dir=$2 cmd=$3
shift 3

git -C "$dir" rev-parse -q --verify "$base^{commit}" > /dev/null || { echo "unknown base: $base" >&2; exit 2;
}
if [ -n "$(git -C "$dir" status --porcelain)" ]; then
  echo "$dir has uncommitted changes: commit the agent's result first" >&2
  exit 2
fi

echo "run:    $(git -C "$dir" branch --show-current)"
stat=$(git -C "$dir" diff --shortstat "$base" HEAD)
```

```
echo "diff:  ${stat:- no changes}"

touched=$(git -C "$dir" diff --name-only "$base" HEAD -- "$@")
if [ -n "$touched" ]; then
  echo "tests:  TOUCHED"
  echo "$touched" | sed 's/^/        /'
else
  echo "tests:  untouched"
fi

# Put the original tests back, run the suite, then return to the agent's commit.
git -C "$dir" restore --source="$base" -- "$@" || exit 2
if (cd "$dir" && bash -c "$cmd") > /dev/null 2>&1; then suite=pass; else suite=FAIL; fi
git -C "$dir" reset -q --hard
git -C "$dir" clean -qfd -- "$@"
echo "suite:  $suite (with the original tests)"
```

> **CUIDADO**
>
> El script termina con `git reset --hard` y `git clean` en esa carpeta, por eso se niega a correr con cambios sin commit. Úsalo solo en las carpetas de prueba.

- **Tests sin tocar:** `tests: untouched` y `suite: pass`. Un `npm test` en verde no basta: en nuestra prueba, borrar el archivo de test lo dejó en verde porque `node --test` sin archivos no falla.
- **Tiempo:** el de `runs/times.txt`, el mismo reloj para las dos. Trata las diferencias pequeñas como empate.
- **Costo en Claude Code:** `total_cost_usd` y `modelUsage` del JSON. Es una estimación local a precio de lista; con suscripción Pro, Max o Ultra no es lo que pagas.
- **Costo en Codex:** suma el `usage` de cada línea `turn.completed` del JSONL. Con un plan de ChatGPT consume los límites del plan (`/status`); con clave de API se cobra por token.
- Usa una unidad común: dólares si pagas por API en las dos; con suscripciones, compara cada herramienta solo consigo misma. El script `usage.mjs` del artículo lee las dos salidas.

PASO 05

Bugs que ya arreglaste

Son los mejores candidatos, pero en un worktree el arreglo se ve con `git log --all`. Usa clones que solo traen el historial hasta la base, y mide contra `origin/cmp/base-2`:

```
fix=a1b2c3d   # el commit donde arreglaste el bug 2
git switch -c cmp/base-2 "$fix^"
git checkout "$fix" -- test/shipping.test.js
git commit -m "test: bug 2 (fails)"
git switch main

git clone --no-local --single-branch --no-tags --branch cmp/base-2 . ../cmp/claude-2
git clone --no-local --single-branch --no-tags --branch cmp/base-2 . ../cmp/codex-2
git -C ../cmp/claude-2 switch -c cmp/claude-2
git -C ../cmp/codex-2 switch -c cmp/codex-2
```

PLANTILLA

El marcador

Una fila por corrida:

BUG	HERRAMIENTA	TIEMPO (S)	DIFF (LÍNEAS)	TESTS SIN TOCAR	SUITE ORIGINAL	COSTO
1	Claude Code:	____	+___ −___	sí / no	pass / FAIL	______
1	Codex:	____	+___ −___	sí / no	pass / FAIL	______
2	Claude Code:	____	+___ −___	sí / no	pass / FAIL	______
2	Codex:	____	+___ −___	sí / no	pass / FAIL	______
3	Claude Code:	____	+___ −___	sí / no	pass / FAIL	______
3	Codex:	____	+___ −___	sí / no	pass / FAIL	______

Puertas (tests sin tocar y suite en `pass`): Claude Code ___ / 3 · Codex ___ / 3. **Modelos y versiones:** _____. **Unidad de costo:** _____.

Cómo decidir

1. Primero la puerta: una corrida que tocó tests o falla la suite original no es un arreglo.
2. Cuenta las puertas de cada herramienta en los tres bugs.
3. Con las puertas empatadas, mira el diff: cambios fuera del área del bug, refactors no pedidos, dependencias nuevas.
4. Después el costo en tu unidad y al final el tiempo, la métrica más ruidosa.
5. Si sigue parejo, es empate: decide por lo que ya pagas y por cómo encaja cada una en tu flujo.

Al terminar, borra las carpetas de prueba (`git worktree remove` o los clones) y las ramas `cmp/`. Guarda el marcador y `runs/`: repite la prueba cuando cambie una versión o un modelo.