EcomLabs

TEMPLATE AND METHOD

Claude Code vs Codex: same bug, four metrics

How to compare two coding agents on your own repo: a base with the failing test, two identical branches, one prompt and four metrics (time, diff, untouched tests and cost), with the script to measure them and a scoreboard to fill in with three bugs.

FOR	Anyone choosing between Claude Code and Codex who wants to decide on their own code
TESTED WITH	Scripts on Node 22.22.3 · git 2.54.0 · Git Bash (Windows 11), with simulated commits; no agent was run
UPDATED	November 11, 2026
DOCS	code.claude.com/docs/en/headless · learn.chatgpt.com/docs/non-interactive-mode
ARTICLE	ecomlabs.dev/g/versus-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will measure
Step 01	The base and two identical branches
Step 02	The prompt
Step 03	The two runs
Step 04	Measure the four metrics
Step 05	Bugs you already fixed
Template	The scoreboard

BEFORE YOU START

What you will measure

The same bug, solved by Claude Code and by Codex from the same commit with the same prompt. For each run you measure **time**, **diff**, **untouched tests** and **cost**; you repeat it with three bugs and decide with the scoreboard. It is a method, not a benchmark: there are no results in it. The commands were checked against both CLIs' official documentation on November 11, 2026, and the script was tested in a practice repo with hand-made commits, without running either agent. You need git, bash (Git Bash on Windows) and both CLIs with their accounts.

STEP 01

The base and two identical branches

Commit the test that reproduces the bug on a base branch and create one worktree per agent from it:

```
git switch -c cmp/base-1
# add the test that reproduces the bug, then:
git add test/shipping.test.js
git commit -m "test: free shipping from exactly 999 (fails)"
npm test   # must fail, and for this reason

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)
```

- Create the worktrees yourself: per their docs, `claude --worktree` starts from the remote's default branch by default, and the Codex app from the `HEAD` of the selected branch plus your uncommitted changes.
- Codex reads `AGENTS.md`; Claude Code reads `CLAUDE.md` and, since v2.1.277, `AGENTS.md` when there is no `CLAUDE.md`. Keep one shared `AGENTS.md` or none.

STEP 02

The prompt

Save it as `../cmp/prompt.md`. For each bug, change only the description:

```
Bug: with a subtotal of exactly $999 MXN the store charges shipping.
Shipping is free from $999. The test "ships free from exactly $999"
in test/shipping.test.js reproduces it.

Fix the code so that npm test passes.
Do not modify, delete or skip files in test/: they are the spec.
Do not commit: leave your changes in the working tree.
When you finish, summarize in three lines what you changed and why.
```

STEP 03

The two runs

One after the other, without stepping in, alternating which goes first on each 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:** in `-p`, anything that needs approval is denied; `acceptEdits` lets it edit and `Bash(npm test *)` lets it run `npm test`.
- **Codex:** `codex exec` is read-only by default; `--sandbox workspace-write` lets it edit (`--full-auto` is no longer supported).
- Pin and record the model (`--model` in Claude Code, `-m` in Codex) and each CLI's version.

STEP 04

Measure the four metrics

Diff and tests: save `../cmp/agent-metrics.sh` and run it from `../cmp` with `bash agent-metrics.sh cmp/base-1 claude "npm test" test` (then the same with `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)"
```

> **CAUTION**
>
> The script ends with `git reset --hard` and `git clean` in that folder, which is why it refuses to run with uncommitted changes. Use it only on the test folders.

- **Untouched tests:** `tests: untouched` and `suite: pass`. A green `npm test` is not enough: in our test, deleting the test file left it green, because `node --test` with no test files does not fail.
- **Time:** from `runs/times.txt`, the same clock for both. Treat small differences as a tie.
- **Cost in Claude Code:** `total_cost_usd` and `modelUsage` from the JSON. It is a local estimate at list price; on a Pro, Max or Ultra subscription it is not what you pay.
- **Cost in Codex:** add up the `usage` on every `turn.completed` line of the JSONL. On a ChatGPT plan it draws on the plan's limits (`/status`); with an API key it is billed per token.
- Use a common unit: dollars if you pay by API for both; on subscriptions, compare each tool only with itself. The article's `usage.mjs` script reads both outputs.

STEP 05

Bugs you already fixed

They are the best candidates, but in a worktree the fix shows up in `git log --all`. Use clones that only carry history up to the base, and measure against `origin/cmp/base-2`:

```
fix=a1b2c3d   # the commit where you fixed 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
```

TEMPLATE

The scoreboard

One row per run:

BUG	TOOL	TIME (S)	DIFF (LINES)	UNTOUCHED TESTS	ORIGINAL SUITE	COST
1	Claude Code:	____	+___ −___	yes / no	pass / FAIL	______
1	Codex:	____	+___ −___	yes / no	pass / FAIL	______
2	Claude Code:	____	+___ −___	yes / no	pass / FAIL	______
2	Codex:	____	+___ −___	yes / no	pass / FAIL	______
3	Claude Code:	____	+___ −___	yes / no	pass / FAIL	______
3	Codex:	____	+___ −___	yes / no	pass / FAIL	______

Gates (untouched tests and suite at `pass`): Claude Code ___ / 3 · Codex ___ / 3. **Models and versions:** _____. **Cost unit:** _____.

How to decide

1. The gate first: a run that touched tests or fails the original suite is not a fix.
2. Count each tool's gates across the three bugs.
3. If the gates are tied, look at the diff: changes outside the bug's area, refactors nobody asked for, new dependencies.
4. Then cost in your unit, and time last, since it is the noisiest metric.
5. If it is still close, call it a tie: decide by what you already pay for and how each one fits your workflow.

When you are done, remove the test folders (`git worktree remove` or the clones) and the `cmp/` branches. Keep the scoreboard and `runs/`: repeat the test when a version or a model changes.