cxcodex Manual (Story-First Walkthrough)

Last updated: 2026-02-22

Premise

You are in a repo, you want answers you can trust, and you want the runtime to stay predictable. This walkthrough follows one realistic loop: *capture context → summarize → fan out → run tasks → optimize*.

1 The Cast (What cx Actually Is)

- **bin/cx**: the one entrypoint you call. It routes Rust-first and falls back explicitly.
- **cxrs**: the canonical Rust engine. It owns capture, budgeting, schemas, quarantine, logging.
- **Bash**: compatibility/bootstrap only (`lib/cx/*.sh`).
- **.codex/**: the repo-local memory: schemas, logs, quarantine, tasks, state.

2 Scene 1: You Enter the Repo

You start by asking cx who it is *today*: which backend is active, what the budgets are, and where logs are going.

```
cd ~/cxcodex
./bin/cx version
./bin/cx diag
```

What you should notice:

- `execution_path` is `rust:bin/cx` when cxrs is available.
- `log_file` points at `.codex/cxlogs/runs.jsonl`.
- Budgets are explicit (chars/lines/clip_mode).

3 Scene 2: Capture Reality (Without Drowning the Model)

The most common failure mode is context bloat: huge diffs, noisy logs, unbounded output. cx treats system output as a first-class input with a strict pipeline:

```
raw system output
  -> optional RTK routing (system output only)
  -> optional native reduction
  -> mandatory budgeting (clip/chunk)
  -> embed into prompt
```

Try a benign capture-driven call:

```
./bin/cx cxo git status
```

If the repo is large and output is clipped, it should be *visible* in telemetry:

```
./bin/cx budget
./bin/cx trace
```

4 Scene 3: Ask for a Deterministic Summary (Schema-Enforced)

When you need a structured artifact (commit JSON, diff summary, next commands), you do *not* want prose. You want JSON-only, validated, and replayable.

```
./bin/cx schema list
./bin/cx diffsum-staged | jq .
```

If the model returns invalid JSON, cx quarantines it and gives you a handle:

```
./bin/cx quarantine list
./bin/cx quarantine show <id>
./bin/cx replay <id>
```

5 Scene 4: Turn Work Into Tasks (Fan-out)

When the objective is bigger than a single prompt, you fan out into small, role-tagged tasks.

```
./bin/cx task fanout "Harden schema validation errors" --from staged-diff
./bin/cx task list --status pending
```

6 Scene 5: Execute Tasks (Sequential for Now)

You run a task (or run all pending tasks) through the same engine, and logs link the run to the task.

```
./bin/cx task run task_001 --mode deterministic --backend codex
./bin/cx task run-all --status pending
```

7 Scene 6: Stay Safe When Commands Are Suggested

For semi-autonomy, suggestions are cheap. Execution is dangerous. The policy engine is the gate.

```
./bin/cx policy show
./bin/cx policy check "rm -rf ."
```

8 Scene 7: Learn From the Past (Optimize)

The loop ends by asking: what is slow, what is token-heavy, and what is drifting?

```
./bin/cx optimize 200
./bin/cx optimize 200 --json | jq .
```

9 Appendix: Rules You Can Quote

- Budget system output. Always.
- Enforce schema for structured artifacts. Always.
- Quarantine failures; replay deterministically.
- Make telemetry a contract; validate it.
- Prefer small tasks over giant prompts.