cxcodex Manual (Master)

Last updated: 2026-02-26

Intent

cx is built for one outcome: **predictable LLM-assisted work under load**. That means bounded context, schema-validated structured outputs, strict telemetry, and safe automation boundaries.

This master manual synthesizes three viewpoints:

- a story-first walkthrough (to build the mental model),
- a field guide (to debug and tune quickly),
- an operator playbook (to validate and execute safely).

Architecture Overview

- Canonical engine: `cxrs` (Rust). Bash is fallback only.
- Authoritative entrypoint: `bin/cx` routes Rust-first with explicit fallback.
- Capture pipeline applies to *system output* only (optional RTK, then budgets).
- Structured commands are schema-enforced (JSON-only, validated, replayable).
- Schema failures are quarantined (`.codex/quarantine/`) and logged with `quarantine_id`.
- Run telemetry is append-only JSONL (`.codex/cxlogs/runs.jsonl`).
- Policy engine blocks destructive/out-of-repo writes by default.

Technical Expose (Rust Refactor Branch)

- Active branch focus: split monolithic command logic into cohesive Rust modules.
- Orchestrator: `src/app.rs` (routing + composition).
- Core extracted command modules: `introspect`, `runtime_controls`, `agentcmds`, `logview`, `analytics`, `diagnostics`, `routing`, `prompting`, `optimize`, `doctor`, `schema_ops`, `settings_cmds`.
- Behavioral contract is unchanged: deterministic schemas, quarantine/replay, policy gating, budgeted capture, append-only logs.
- Quality gate per extraction slice: `cargo fmt` + `cargo test` must pass.

Quickstart (3 Commands)

```
cd ~/cxcodex
./bin/cx version
./bin/cx budget
./bin/cx diffsum-staged | jq .
```

Common Workflows (Copy/Paste)

```
# Structured outputs:
./bin/cx diffsum-staged | jq .
./bin/cx commitjson | jq .

# Task graph:
./bin/cx task fanout "Objective..." --from staged-diff
./bin/cx task run-all --status pending

# Optimization:
./bin/cx optimize 200
```

Installation / Prereqs

- Required: `bash`
- Required: `git`
- Required: `jq`
- Required: `codex` CLI (default backend)
- Development: Rust toolchain (`cargo`, `rustc`)

```
./bin/cx doctor
./bin/cx rtk-status
```

Optional: `rtk` (system output only), `ollama` (local backend).

Troubleshooting (Fast)

- Unexpected fallback: `./bin/cx where <cmd>` and confirm `execution_path` via `./bin/cx version`.
- Schema command fails: inspect `./bin/cx quarantine list`, then `./bin/cx replay <id>`.
- Budget clipping surprises: check `./bin/cx budget` and `./bin/cx trace`; reduce capture scope or adjust budgets intentionally.
- Strict log validation fails: legacy rows may predate the current telemetry contract; new runs should conform.
- Backend confusion: run `./bin/cx llm show` and verify the selected model when using ollama.

Contents

1 Start Here: Verify What You're Running

Fast path (30 seconds)

```
cd ~/cxcodex
./bin/cx version
./bin/cx diag
./bin/cx where version diffsum-staged task optimize
```

If `execution_path` is not Rust, you are on fallback. Expect reduced guarantees.

2 The Pipeline (What cx Does With Your Reality)

2.1 Capture and Budgeting

The core reliability move is this: **system output is constrained before it becomes prompt text**.

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

Budgets (defaults):

- `CX_CONTEXT_BUDGET_CHARS=12000`
- `CX_CONTEXT_BUDGET_LINES=300`
- `CX_CONTEXT_CLIP_MODE=smart|head|tail` (default `smart`)
- `CX_CONTEXT_CLIP_FOOTER=1`

Inspect current budgets + last-run clip stats:

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

2.2 Modes

- `lean`: minimal prompts/output
- `deterministic`: strict schema-only behavior where applicable
- `verbose`: additional human detail (where supported)

Schema commands always force deterministic mode by default. Override with `CX_SCHEMA_RELAXED=1`.

3 Structured Outputs (Schemas, Quarantine, Replay)

3.1 Schema Registry

Schemas live under `.codex/schemas/`:

- `commitjson.schema.json`
- `diffsum.schema.json`
- `next.schema.json`
- `fixrun.schema.json`

```
./bin/cx schema list
./bin/cx schema list --json | jq .
```

3.2 Schema Commands

```
./bin/cx commitjson | jq .
./bin/cx diffsum-staged | jq .
./bin/cx next git status | jq .
```

3.3 Failure Procedure

If validation fails, cx writes the raw response + metadata to quarantine and returns non-zero.

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

4 Safety Sandbox (Policy Engine)

The policy engine is the boundary between “suggest” and “execute”.

Blocked by default (minimum):

- `sudo`, `rm -rf`, `curl | bash/sh/zsh`
- `chmod/chown` on system paths
- writes outside repo root

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

5 Task Graph (Fan-out, Run, Run-all)

5.1 Why tasks

Tasks are the unit of migration to parallel work: small, role-tagged, log-linked, and replayable.

```
./bin/cx task fanout "Ship release notes improvements" --from staged-diff
./bin/cx task list --status pending
```

5.2 Run tasks

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

6 Telemetry (Contract, Validation, Interpretation)

6.1 Log locations

- runs: `.codex/cxlogs/runs.jsonl`
- schema failures: `.codex/cxlogs/schema_failures.jsonl`

6.2 Validate

```
./bin/cx logs validate --fix=false
```

Note: strict validation will flag older historical rows that predate the current contract.

6.3 Interpretation (fast heuristics)

- `effective_input_tokens` high: prompts too large or too variable.
- cache trend down: stabilize prompt templates and capture scope.
- frequent clipping: reduce capture scope or raise budgets intentionally.
- schema failures spike: enforce deterministic mode and reduce context ambiguity.

7 Self-Optimization (Pure Analysis)

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

8 Operator Procedures (Copy/Paste)

Budget stress test

```
CX_CONTEXT_BUDGET_CHARS=2000 CX_CONTEXT_BUDGET_LINES=40 ./bin/cx cxo git
   status
./bin/cx budget
./bin/cx trace
```

Schema integrity check

```
./bin/cx commitjson | jq .
./bin/cx diffsum-staged | jq .
```

Task loop

```
./bin/cx task fanout "Objective: tighten schema errors" --from log
./bin/cx task run-all --status pending
./bin/cx optimize 200
```