TL;DR

/goal is an under-development OpenAI Codex CLI feature that gives a thread a durable objective. It is more than a reminder in the prompt: the CLI stores goal state, exposes goal tools to the model, tracks tokens and time, can continue work when the thread goes idle, and can stop at a budget. As of May 2, 2026, I would use it only in Codex CLI, with the goals experimental feature enabled. I do not see a supported /goal surface in Codex Desktop yet.

If you have searched for "Codex /goal", "OpenAI Codex goal command", or "Codex CLI goals", you are probably ahead of the docs.

The public Codex CLI slash-command docs currently list commands such as /model, /plan, /experimental, /status, /compact, and /diff, but not /goal. The Codex app commands docs list /feedback, /mcp, /plan-mode, /review, and /status, but not /goal.

So this post is intentionally date-stamped. It is based on official OpenAI docs checked on May 2, 2026 plus the public openai/codex source at tag rust-v0.128.0.

What /goal does

In Codex CLI, a thread is the durable conversation state behind your local session. It has an ID, a transcript, tool events, configuration context, and other state Codex can resume later.

/goal adds a structured objective to that thread.

At rust-v0.128.0, the CLI implementation supports these shapes:

```
/goal
/goal <objective>
/goal pause
/goal resume
/goal clear
```

The practical difference from typing "keep working on X" is that a goal is not just another chat message. It becomes thread state. Codex can ask for the current goal through a model tool, create one, and mark it complete when the work is genuinely done.

That matters because long-running coding work has several failure modes:

- The model finishes a turn but leaves the broader task unfinished.
- The session is resumed later and needs the real objective, not just the last visible message.

- The user wants progress bounded by a token budget.
- The model needs to know whether the current work is still in scope.

/goal is OpenAI's experimental answer to that class of problem.

How it works under the hood

The source is the most useful documentation right now.

In slash_dispatch.rs, /goal is feature-gated behind Feature::Goals. That means users should not assume it is available unless the goals feature is enabled in their CLI.

The persistence layer includes a thread_goals table in 0029_thread_goals.sql. The stored fields include:

- thread_id
- goal_id
- objective
- status
- optional token_budget
- tokens_used
- time_used_seconds
- created and updated timestamps

The model-facing tools live in goal_tool.rs. The tool surface is deliberately small:

- get_goal
- create_goal
- update_goal

The important constraint: the model can mark a goal complete, but pause, resume, and budget-limit transitions are controlled by user or runtime behavior. That keeps the model from silently changing the operating mode just because it wants to.

The runtime behavior lives in goals.rs. That is where Codex accounts for token usage, tracks elapsed time, reacts to interrupts, resumes paused goals, and can inject a continuation turn when the thread is idle and an active goal remains.

In plain English: /goal gives the LLM a real task state channel and gives the runtime a reason to keep going or stop cleanly.

How to enable /goal right now

As of May 2, 2026, treat goals as an experimental or under-development feature.

First, check whether your CLI sees it:

```
codex features list
```

OpenAI's Codex CLI features docs describe the feature-flag flow:

```
codex features list
codex features enable <feature>
codex features disable <feature>
```

For this specific feature, the local command is:

```
codex features enable goals
```

You can also enable a feature for a launch:

```
codex --enable goals
```

Once OpenAI officially releases `/goal` as a stable Codex feature, this experimental enablement step should no longer be needed. Until then, keep the article date in mind, check `codex features list`, and re-check the official CLI docs before telling a team to depend on it.

How to use it in a real session

Start Codex CLI from the repository you want it to work on:

```
codex --enable goals
```

Then set a concrete objective:

```
/goal fix the flaky OAuth callback test and push the smallest safe patch
```

Good goals are specific enough that Codex can tell when they are complete:

```
/goal implement Phase 0 of the SEO eval harness, run the documented verification, and leave a short handoff note
```

Weak goals are too broad:

```
/goal improve the repo
```

During a long run, use:

```
/goal pause
/goal resume
/goal clear
```

Bare `/goal` opens the goal UI in the CLI TUI when the feature is available.

CLI only, not Codex Desktop

Today, plan on using /goal in Codex CLI only.

The official Codex app commands page does not list /goal, and the Desktop build I inspected did not expose the experimental feature controls needed to turn on goals. It had signs of thread goal notifications at the app boundary, but not a supported composer command or user-facing experimental toggle.

That may change. Codex Desktop could add a goal UI later, and OpenAI may move /goal from under-development to stable. But if you are writing a workflow today, do not assume a Desktop user can type /goal and get the same behavior as the CLI.

What /goal is not

/goal is not a replacement for good handoffs, tests, or review. A durable objective helps Codex stay oriented, but it does not prove the implementation is correct.

It is also not the same thing as /compact. /compact shrinks old conversation history into a summary. /goal stores the active objective separately so the runtime and model can reason about progress.

It is not a general scheduler either. If you need a task to run tomorrow at 9 AM, use the automation mechanisms Codex supports for scheduled work. /goal is about maintaining progress inside a thread.

My current recommendation

Use /goal when the task is bigger than a single prompt and has a clear finish line.

I would use it for:

- fixing a test suite until the targeted tests pass
- implementing a named spec phase
- reviewing and addressing a bounded set of PR comments
- writing a small tool plus its verification steps

I would avoid it for:

- vague cleanup
- open-ended research with no stop condition
- anything where a Desktop-only teammate needs the same control surface
- production team docs that assume stable official behavior before OpenAI documents it

The safest phrasing for now is:

In Codex CLI 0.128.0, /goal exists behind the under-development goals feature. Enable it explicitly, use it for bounded long-running tasks, and re-check the official docs before depending on it as stable product behavior.

Sources checked

- OpenAI Codex CLI overview: developers.openai.com/codex/cli
- OpenAI Codex CLI slash commands: developers.openai.com/codex/cli/slash-commands
- OpenAI Codex CLI features: developers.openai.com/codex/cli/features
- OpenAI Codex app commands: developers.openai.com/codex/app/commands
- openai/codex source at rust-v0.128.0: github.com/openai/codex/tree/rust-v0.128.0