T3 Code Handbook

Flavio Copes

Updated November 24, 2026

Contents

What T3 Code is (and isn't) . . . 3
The mental model . . . 4
Install T3 Code . . . 6
Connect providers, ACP, secrets, and text generators . . . 8
Start your first thread and master the composer . . . 12
Workspaces, worktrees, and multi-repo projects . . . 19
Permission modes, approvals, and checkpoints . . . 21
Multi-provider orchestration and portable handoffs . . . 23
Review changes: files, diffs, visual replies, browser, and devices . . . 26
Issues, pull requests, review guides, and PR watch . . . 31
The thread sidebar and Usage dashboard . . . 34
Projects, `t3.json` cookbook, and automation . . . 37
Remote access, linked environments, and mobile . . . 43
Inside the architecture . . . 47
My real setup and five playbooks . . . 50
T3 Code compared with the alternatives . . . 51
Quick-reference cheat sheets . . . 53
Release history . . . 56
Go deeper on the foundations . . . 58

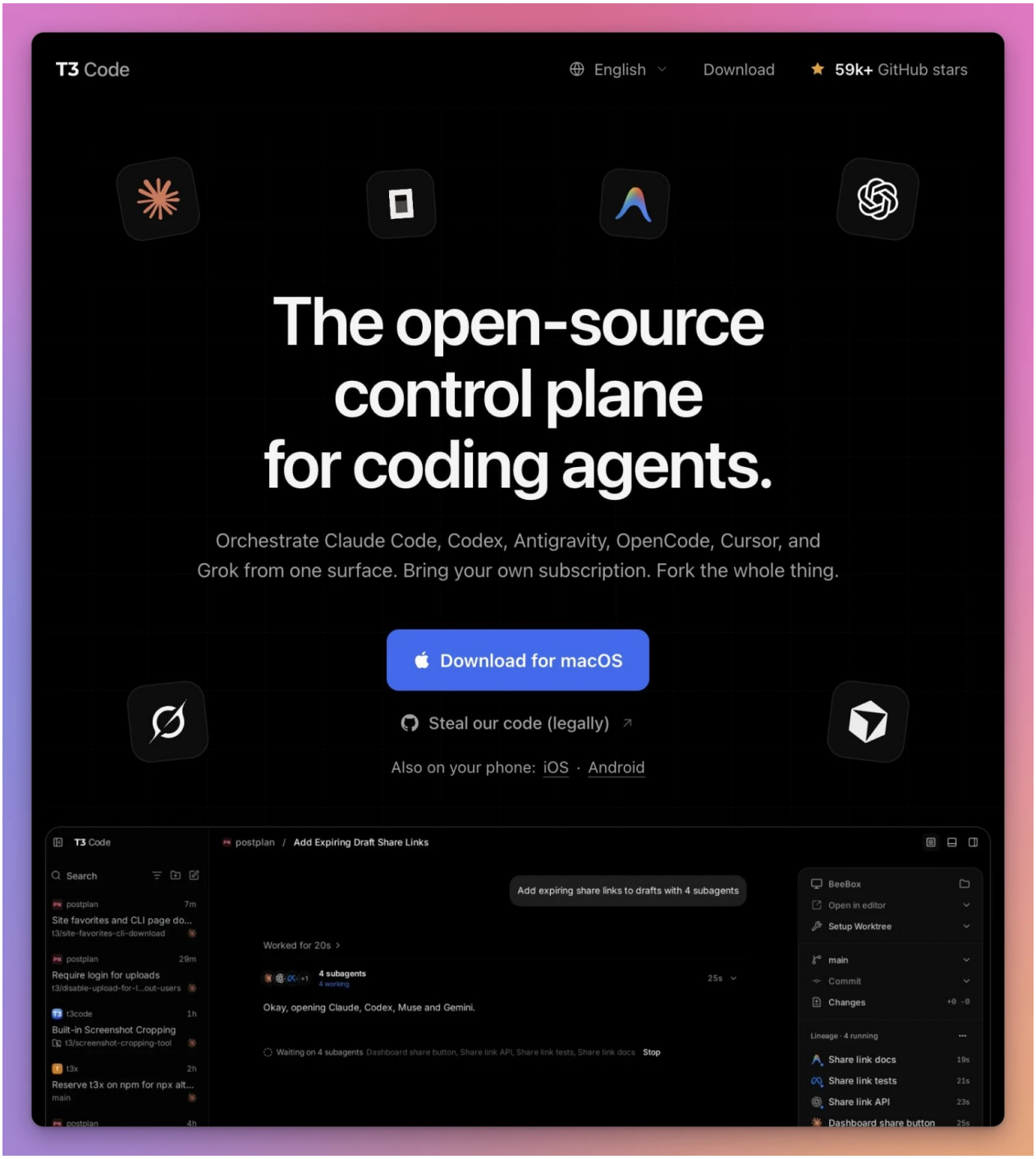
Preface

This is the offline handbook edition of T3 Code Handbook, written by Flavio Copes.

You can read the online version with interactive diagrams and videos at https://flaviocopes.com/t3-code/, and find more programming tutorials, courses, and handbooks at https://flaviocopes.com.

> **Video walkthrough:** Watch the embedded video in the online edition at https://flaviocopes.com/t3-code/.

T3 Code is a free, open source control plane that puts the coding agents you already use behind one interface. A server on your machine starts Codex, Claude Code, Cursor, Antigravity, and the rest, while you supervise them from a desktop app, a browser, or your phone.



It was created by @theo and @jullerino and lives in pingdotgg/t3code under the MIT license. The

desktop app is called **T3 Code (Beta)**, and development moves fast: on August 27, 2026 it passed 20,000 GitHub stars, and recent releases regularly pull in hundreds of contributors.

I use several coding agents. Codex is strong on some tasks, Claude Code on others, Cursor and Antigravity have their own strengths, and OpenCode, Pi, Grok Build, and fx add even more options. I wrote down how I pick among them.

Starting one agent in a terminal is easy. Keeping track of five agents across three repositories, checking which one is waiting for approval, reviewing their diffs, and verifying their changes in a browser or simulator is the real bottleneck.

T3 Code gives all of them one control surface. The work still happens on your machine, using your existing provider logins and subscriptions.

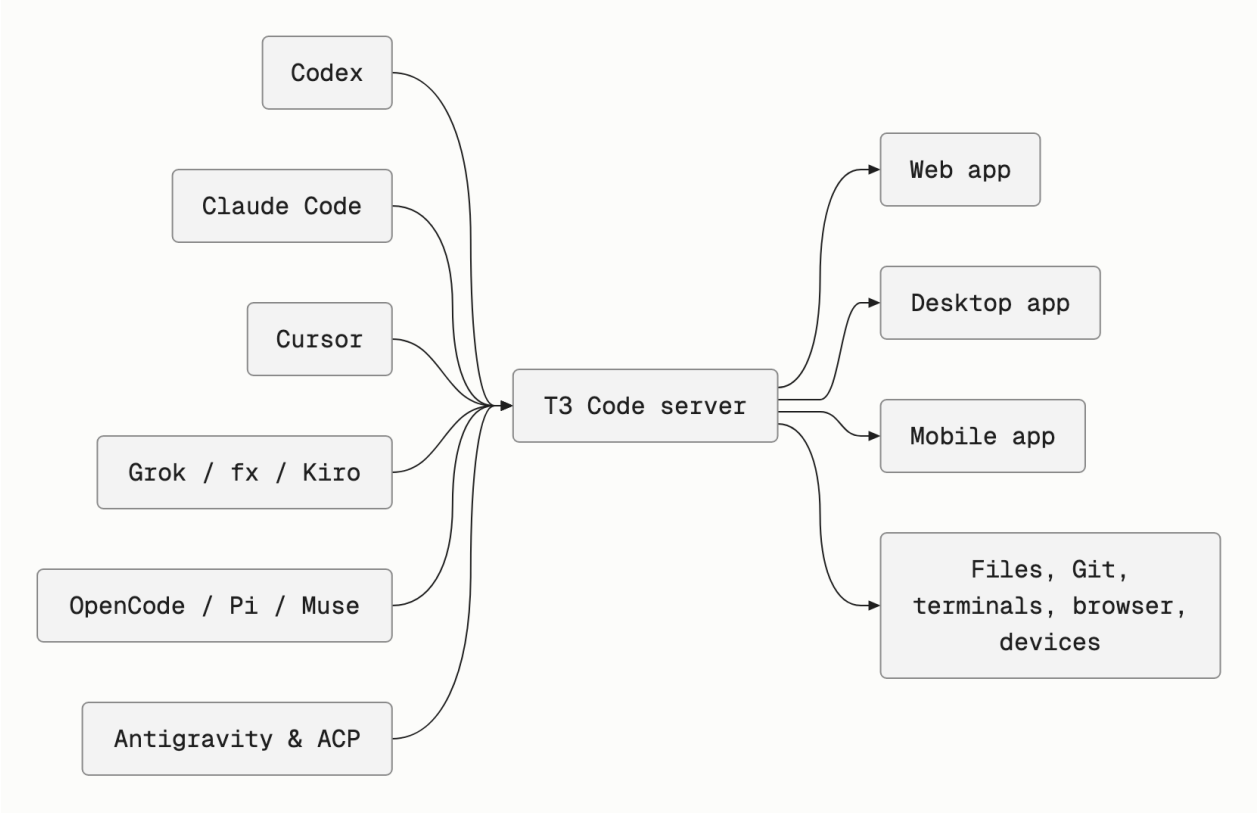
This guide covers T3 Code through v0.1.11 (November 20, 2026). You can also watch several of these workflows live in Theo's T3 Code walkthrough.

What T3 Code is (and isn't)

Running one agent in a terminal is simple: open a repo, run `codex` or `claude`, describe the task, and wait.

The trouble starts when several jobs run at once. Suddenly you have to track which agent owns which branch, who is waiting on a permission prompt, which edits came from which turn, where the dev server is listening, which thread has an open pull request, and how much weekly quota is left.

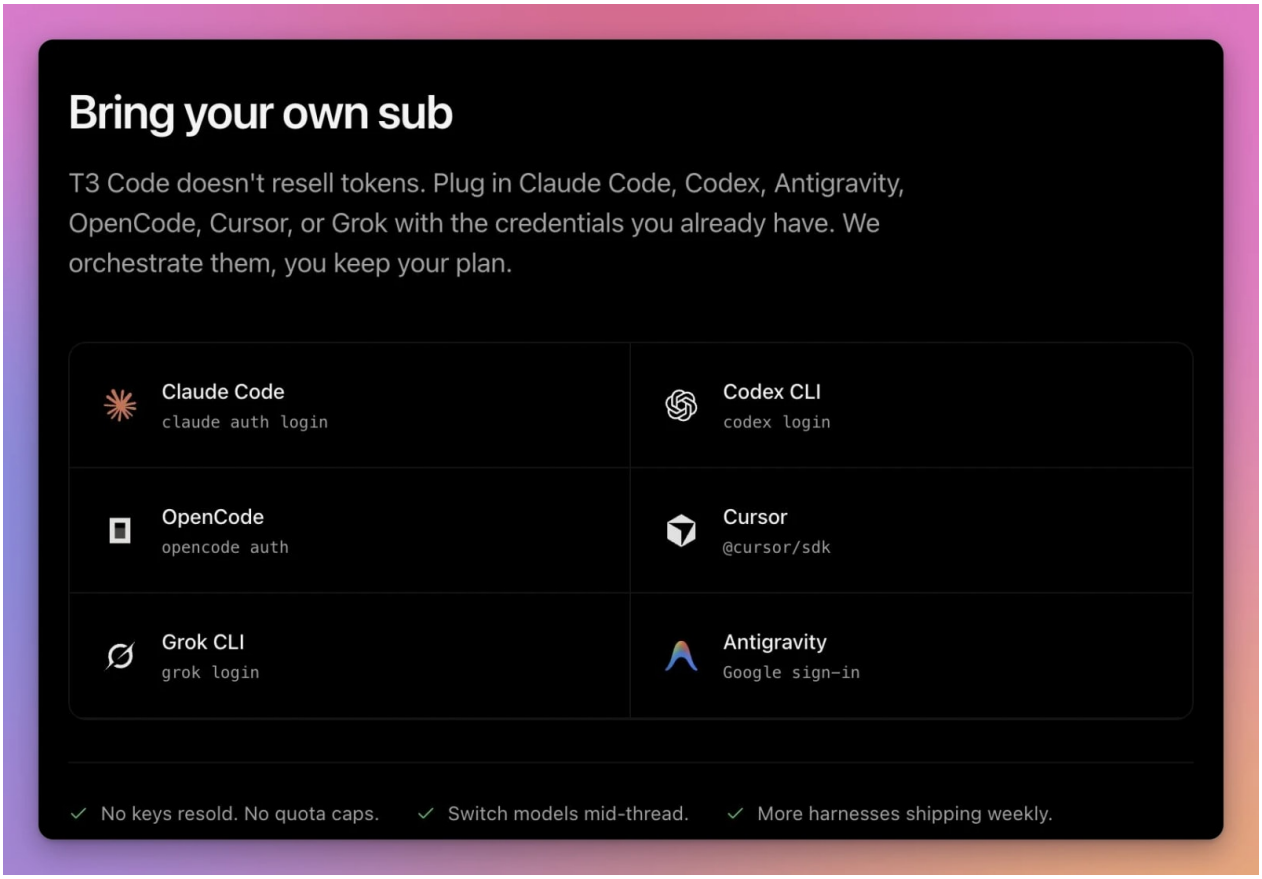
T3 Code solves that by running a local server beside your repositories. That server starts each provider's own tool, runs Git commands, hosts terminals and browser tabs, reads files, and streams normalized events to every connected client.



The project calls itself an **agent harness control surface**. A model generates text; a harness like Codex CLI or Claude Code wraps it with file tools, shell commands, approvals, and session state (OpenAI also offers the Codex harness directly, covered in my guide to the OpenAI Agents API). T3 Code sits one layer above those harnesses, translating each provider's structured protocol into one shared interface.

Drawing a clear line around what T3 Code is *not* prevents most early surprises:

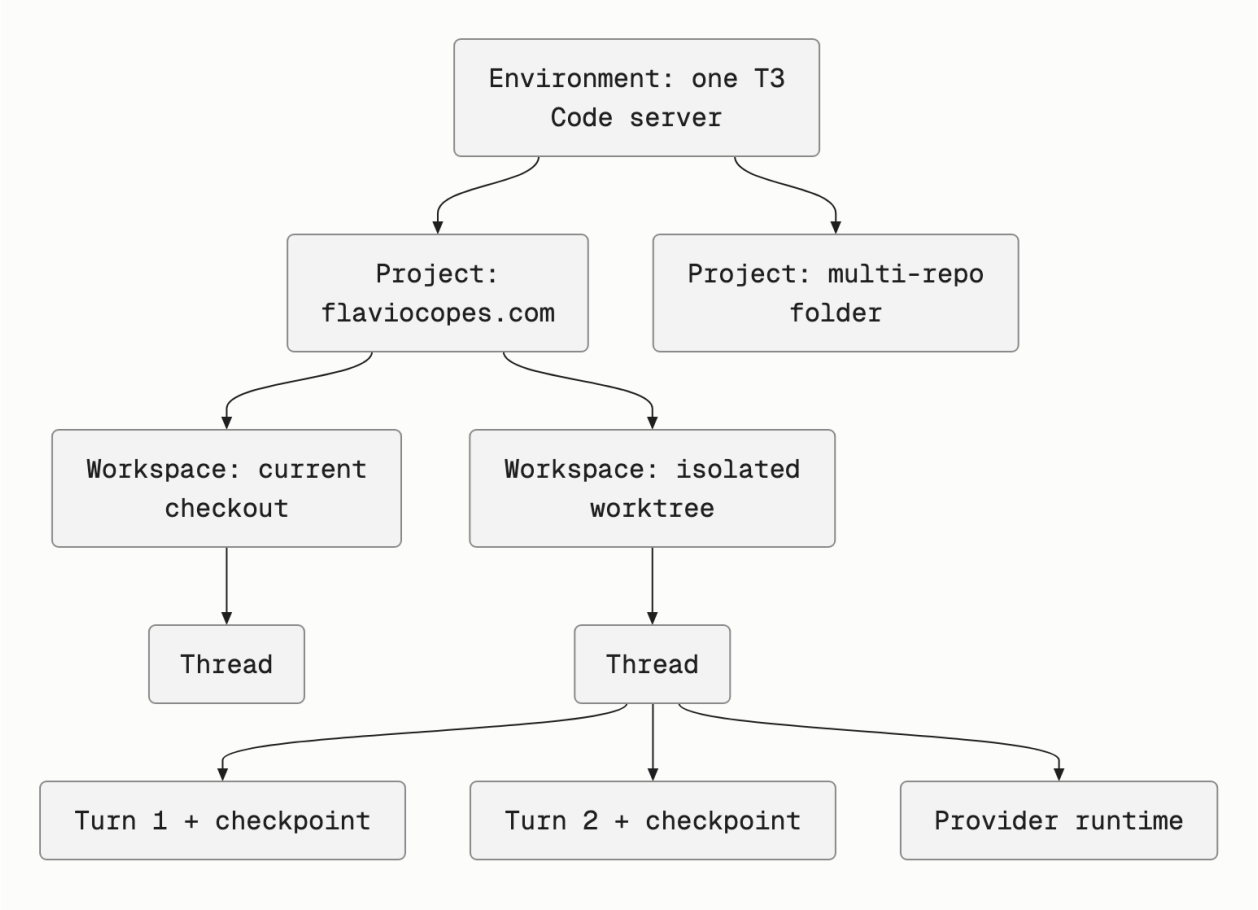
- **Not a model provider.** T3 Code sells no tokens or credits. You bring your own subscriptions or API keys and authenticate each provider on the server machine.



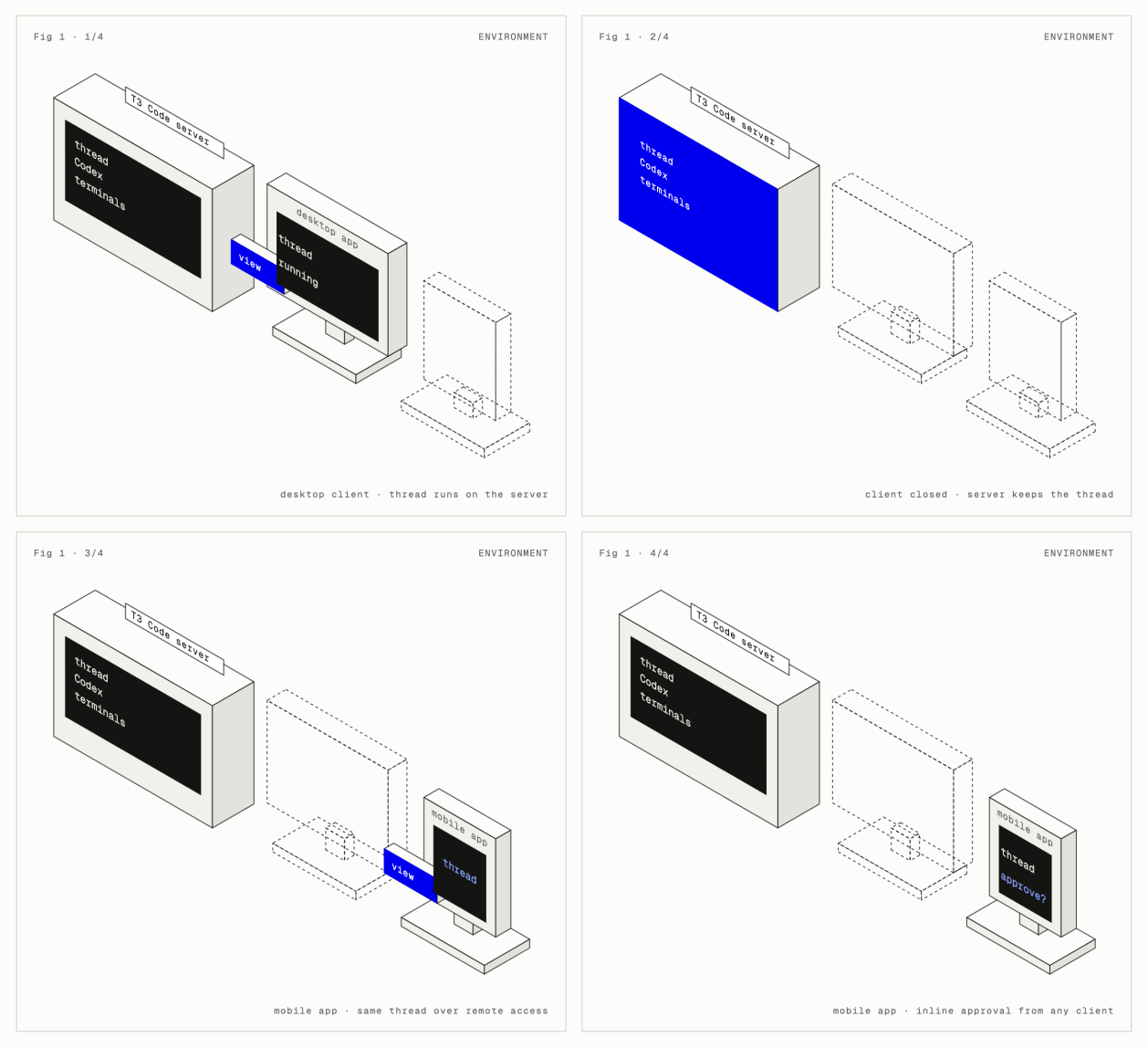
- **Not one homogenized agent.** Provider-specific models, thinking levels, permission classifiers, and tools stay visible. Switching providers changes the actual agent runtime underneath.
- **Not a hosted cloud sandbox.** Everything runs on a machine you control. Close your laptop and its local server stops. Use an always-on Linux box or Mac if you want background jobs running while your laptop sleeps.
- **Not a local inference engine.** Keeping T3 Code state on your machine still sends prompts, code context, and tool results to the cloud provider you selected (unless you point OpenCode or a text generator at a local model).
- **Not a security boundary on its own.** Permission modes reduce mistakes, but **Full access** still gives commands full access on the server machine. Choose workspaces and permissions with the same care you use in the provider CLI.

The mental model

T3 Code is built around six concepts. Their boundaries explain almost every button in the app.



- **Environment:** one running T3 Code server and the machine behind it. It owns the filesystem, projects, threads, credentials, terminals, Git state, and local SQLite database. Your MacBook Pro can be one environment and a Linux VPS another. *The client shows the work; the environment performs it.*



- **Project:** an environment-local record rooted in a directory (such as `/Users/flaviocopes/www/flaviocopes.com`). A project can cover a single Git repository, a plain folder containing several repositories, or external folders tracked alongside it, plus checked-in `t3.json` configuration.
- **Workspace:** the directory a thread edits, either the **current checkout** or an isolated **new Git worktree**.
- **Thread:** the durable conversation for one task, holding provider/model state, messages, approvals, artifacts, linked issues, and linked pull requests. Closing the client leaves the thread intact on the server.
- **Turn:** one user-agent cycle, wrapped in hidden Git checkpoints so you can inspect per-turn diffs or rewind to before a message.
- **Provider:** the underlying coding agent runtime (Codex, Claude Code, Cursor, Antigravity, or an ACP agent).

Install T3 Code

You can run T3 Code as the `t3` command-line server or as the desktop app. The desktop app bundles the `t3` command and sets it up on first launch (or anytime from **Settings -> About -> t3 command**).

Command line

On macOS or Linux:

```
curl -fsSL https://t3.codes/install.sh | sh
```

On Windows PowerShell:

```
irm https://t3.codes/install.ps1 | iex
```

This installs `t3` into `~/.local/bin`. Start the server and open the web interface with:

```
t3
```

To try T3 Code once without installing it, run `npx t3@latest` (requires Node.js 22.16+, 23.11+, or 24.10+). Set `T3CODE_CHANNEL=nightly` before running the installer to follow the nightly train, or `T3CODE_VERSION` to pin an exact build. Intel Macs have no standalone `t3` executable, so use the desktop app or build from source.

Desktop and mobile apps

Download installers from GitHub Releases or install through a package manager:

Platform	Command or source
macOS	`brew install --cask t3-code`
Windows (Winget)	`winget install T3Tools.T3Code`
Windows (Scoop)	`scoop install extras/t3code`
Debian / Ubuntu	`sudo apt install ./T3-Code-*.deb` (or AppImage for x86_64 / ARM64)
Arch Linux	`yay -S t3code-bin` (nightly: `yay -S t3code-nightly-bin`)
Nix (unstable)	`nix profile add github:NixOS/nixpkgs/nixos-unstable#t3code`
iOS & Android	App Store and Google Play

The direct macOS build uses a standard drag-to-Applications installer:

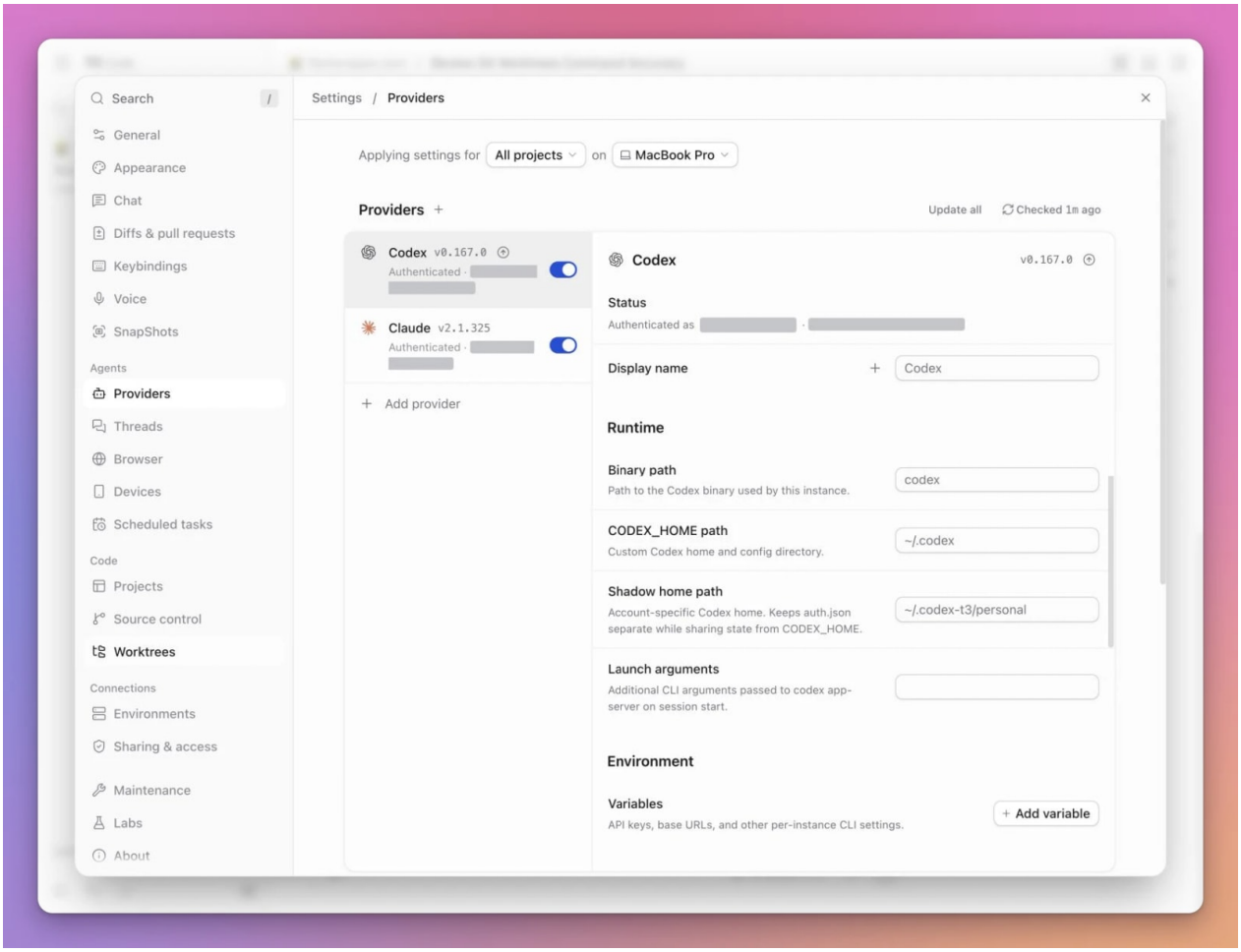


Three quick desktop tricks worth knowing on day one:

- **Open any folder from your terminal:** run `t3 app` (or `t3 app ../my-project`) while the desktop app is open to jump straight into a new thread for that folder.
- **Drag folders onto the sidebar:** drop one or more folders from Finder or Explorer directly onto the sidebar to add them as projects.
- **Windows Subsystem for Linux (WSL):** pick a WSL distro in **Settings -> Sharing & access** to run agents and repositories inside WSL; T3 Code installs its own server runtime there automatically.

Connect providers, ACP, secrets, and text generators

Provider installation and authentication happen on the **server machine**, not on your phone or browser. Open **Settings -> Providers**, pick the environment, and enable what you use.



Built-in providers and capability matrix

T3 Code ships with ten built-in providers plus generic ACP Registry support. Codex and Claude are on by default; enable the rest in **Settings -> Providers**.

Provider	Setup & sign-in	Transport	Permission modes	Edit from here	Extra notes
Codex	Connect with ChatGPT or `codex login` (0.159+)	App-server JSON-RPC	All 4 (AI Auto)	Native rewind	MCP Apps, `/goal`, native resume
Claude Code	`claude auth login` (2.1.280+)	Claude Agent SDK	All 4 (Auto)	Native rewind	MCP Apps (2.1.301+), `/goal`, native resume
Cursor	`agent login` (`cursor-agent`) or browser sign-in	Official Cursor SDK	All 4 (Auto)	Native rewind (`v0.1.11`)	Native resume, 40+ models

Provider	Setup & sign-in	Transport	Permission modes	Edit from here	Extra notes
Antigravity	Install runtime & Google sign-in in Settings	Antigravity ACP (1.5+)	All 4 (Auto on eligible accounts)	Fresh-session replay (`v0.1.11`)	Gemini, Claude & GPT models, `/goal`, video inputs
Grok Build	`grok login` (1.0.13+)	ACP	Supervised, Auto, Full	Provider-dependent	Blocked Auto commands escalate to you
OpenCode	`opencode auth login` (2.0.18+)	OpenCode SDK	All 4 (Auto asks)	Native rewind	Native resume, `/goal` (2.2.5+); don't mix 1.x & 2.x
Pi	`pi`, then `/login` (0.80.5+, Node 22.19+)	Pi CLI	Supervised, Auto-accept, Full	Provider-dependent	Session cost tracking
fx	`curl -fsSL https://fx.sh/setup.sh | bash`, `fx login`	fx CLI	Supervised, Auto, Full (0.0.14+)	Provider-dependent	Vercel's agent, uses your existing skills
Muse Code	`muse login`	Muse CLI	Supervised, Full	No	Session/weekly limits appear after first turn
Kiro	`kiro-cli login` (beta)	Kiro CLI	Supervised, Full	Provider-dependent	Reviews turn file changes in Supervised
ACP Registry	Settings -> Providers -> Add provider -> Show more	ACP v2 / v1	T3 answers by mode	Fresh-session replay	GitHub Copilot, GLM Agent, custom ACP CLIs

Settings -> Providers flags outdated CLIs and lets you click **Update now** for installs owned by Homebrew, mise, npm, pnpm, Yarn, Bun, Volta, or Vite+.

Antigravity and Gemini subscriptions T3 Code installs and drives Google's real Antigravity binary rather than mimicking its API. Because of that, a Google Gemini subscription works inside T3 Code without tripping account bans aimed at third-party shims. On November 23, 2026, Theo summed it up cleanly:

```
T3 Code -> Antigravity binary -> Google API (allowed)
Pi -> fake Antigravity shim -> Google API (not allowed)
```

Antigravity also supports four sign-in methods in Settings: personal Google account, Gemini Enterprise (GCP project/location), Gemini API key, and Vertex AI / Agent Platform. You can check remaining Gemini and shared Claude/GPT quotas under **Usage -> Limits**. Note that Antigravity

reads global skills from `~/.gemini/config/skills` rather than `~/.agents/skills`, unless you list shared skill folders in `~/.gemini/config/skills.json`.

Adding agents from the ACP Registry Beyond the ten built-in providers, open **Settings -> Providers -> Add provider -> Show more** to install agents from the ACP Registry (such as GitHub Copilot or GLM Agent) or wire up any local ACP command (like `dsh --profile acp` for DeepSeek Harness).

T3 Code prefers **ACP v2 preview** for richer plans, usage, compaction, and terminal updates, falling back to ACP v1 automatically. Registry agents install under `~/.t3/tools/<agent-id>/<version>/` with SHA-256 verification when provided. If an ACP agent supports session listing, expand its provider card and choose **List sessions** to import existing native sessions directly into T3 threads.

Multiple accounts for one provider

T3 Code supports multiple configured instances of the same provider, useful for separating personal and work subscriptions:

- **Codex:** keep the same `CODEX_HOME` for shared session history while using a shadow home for separate login credentials, which keeps existing threads reachable across account switches.
- **Claude Code:** Claude isolates sessions by `CLAUDE_CONFIG_DIR`, so switching accounts starts separate session state rather than silently blending them.
- **Antigravity:** add one instance per Google account; all instances share the downloaded runtime.

Secrets, 1Password, and `request_secret`

Sensitive environment variables and tokens are stored on the server and never sent back to clients after saving.

If you use 1Password, connect the 1Password CLI (`op`) on the server machine under **Settings -> Secrets**. Any credential field (provider keys, source control tokens, usage keys, device host tokens) then lets you switch from **Value** to a secret reference like `op://vault/item/field`. On a headless Linux host, sign `op` in with a 1Password service account or Connect server instead of desktop biometric approval.

When a running agent needs a password, API key, pairing code, or `sudo` password, it triggers `request_secret`. Instead of asking you to paste a secret into chat history, T3 Code shows a private card (on web, desktop, or mobile) where you type the value or pick a 1Password reference. The value never enters the transcript; if a shell command needs it, T3 Code writes it to a temporary private file readable by that command for about 24 hours.

Offload titles and commits to a text generator

By default, T3 Code spins up a coding agent (like Codex or Claude) to write thread titles, branch names, commit messages, PR descriptions, and voice transcript cleanup. That works, but it is slower and burns coding-agent turns.

Open **Settings -> Providers -> Text generators -> Add generator** to plug in a fast, cheap text-only API:

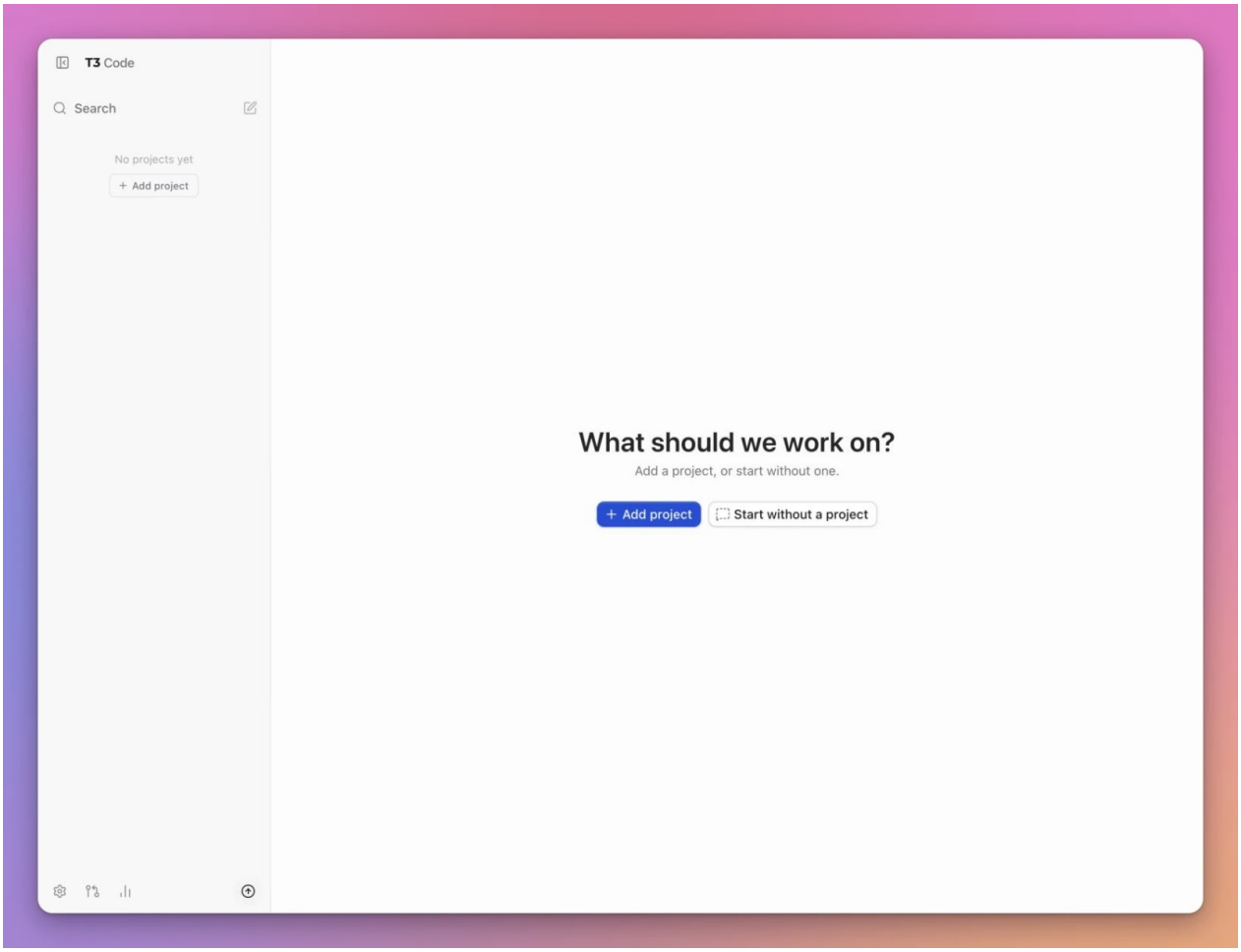
- **OpenAI**, **Anthropic**, or **OpenRouter** with an API key (or 1Password reference)

- **Custom (OpenAI-compatible)** for local models such as Ollama (`http://localhost:11434/v1`) or LM Studio (note that `localhost` resolves on the environment server)

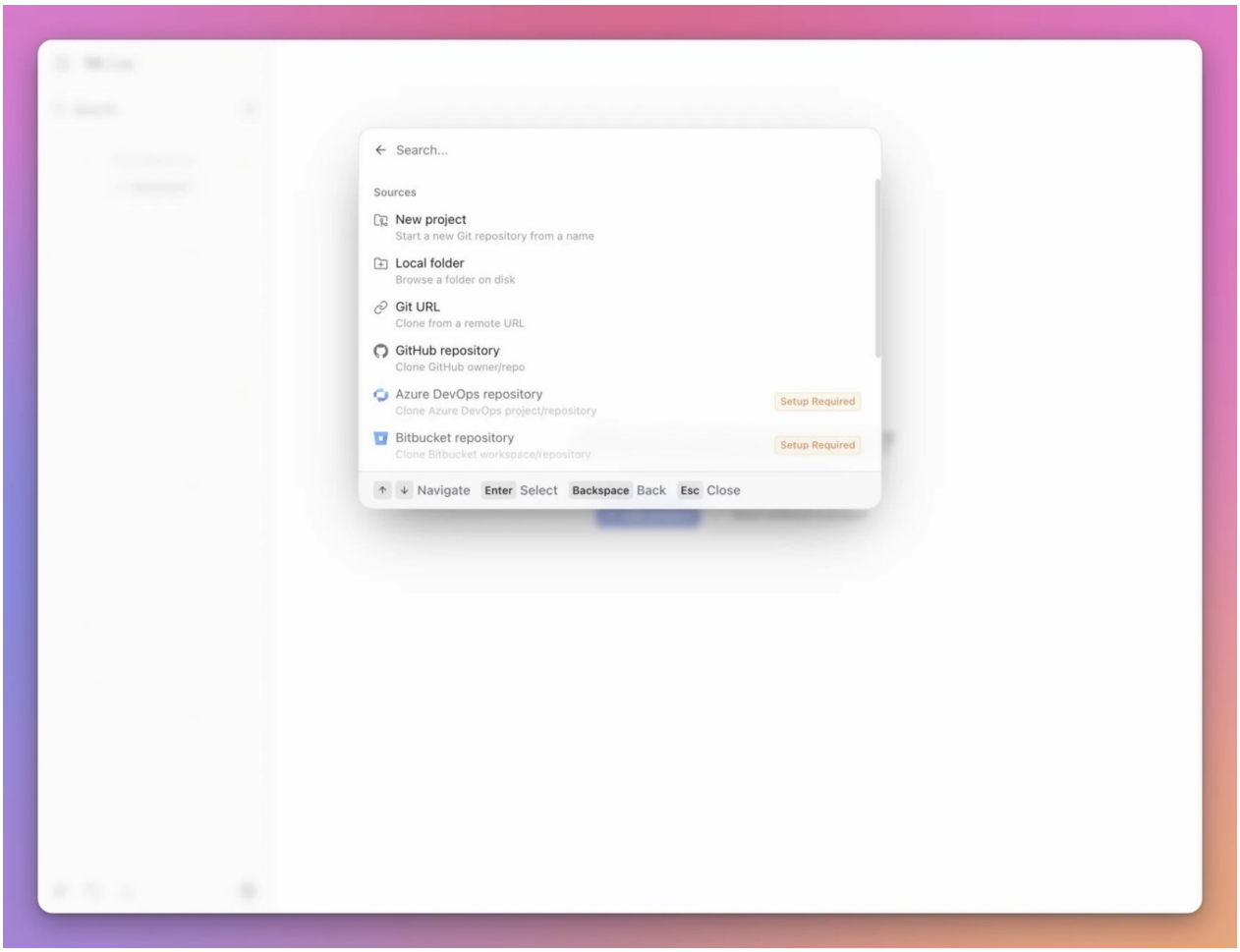
Then assign it under **Settings -> Threads -> Text generation model**, **Settings -> Source control & issues**, or **Settings -> Voice**. One exception: **PR Review Guides** walk through code across multiple steps, so they always require Codex or Claude rather than a text generator.

Start your first thread and master the composer

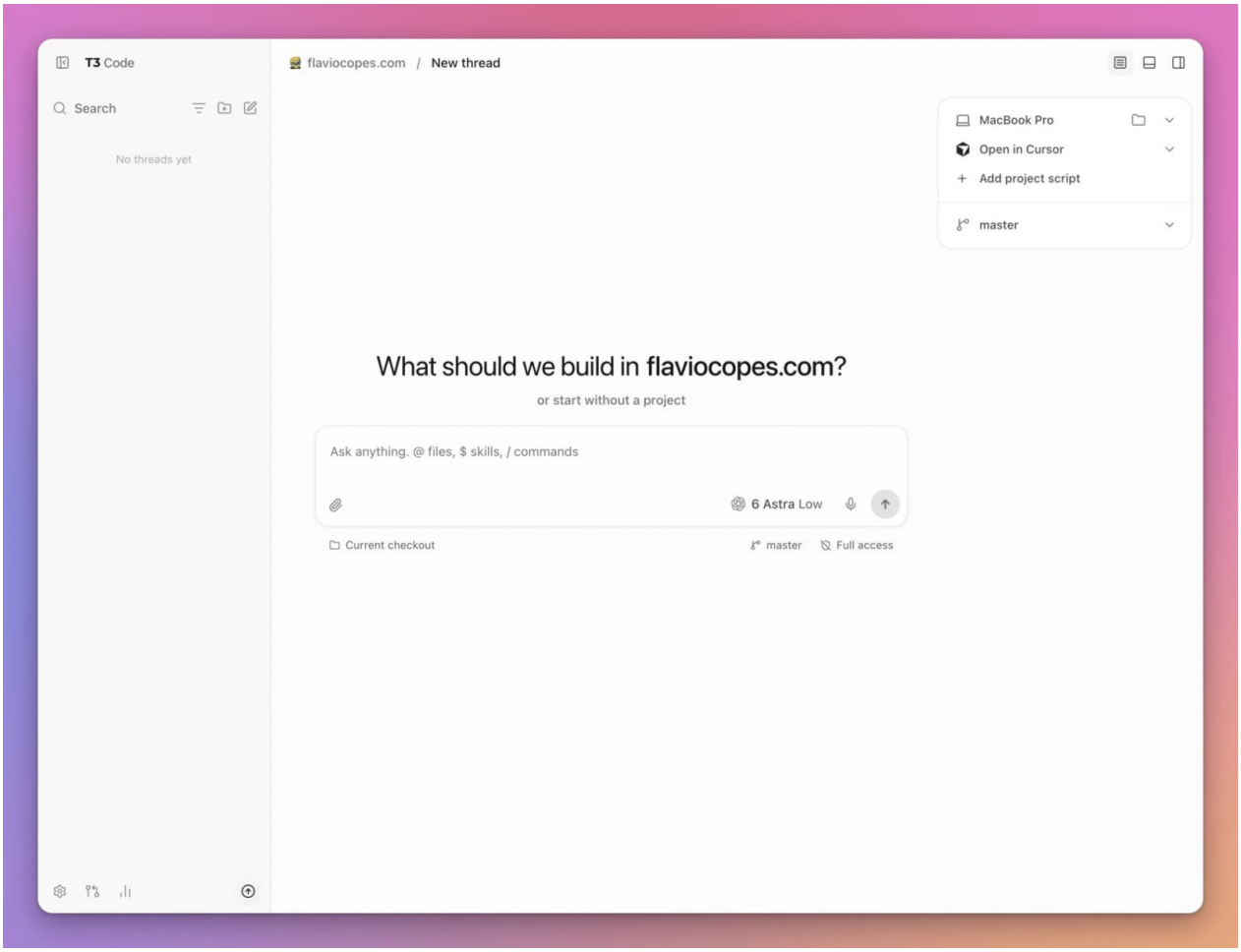
When you first open T3 Code, you can add a project or jump straight into a scratch thread with **Start without a project** (`Cmd+Alt+N` on macOS, `Ctrl+Alt+N` elsewhere):



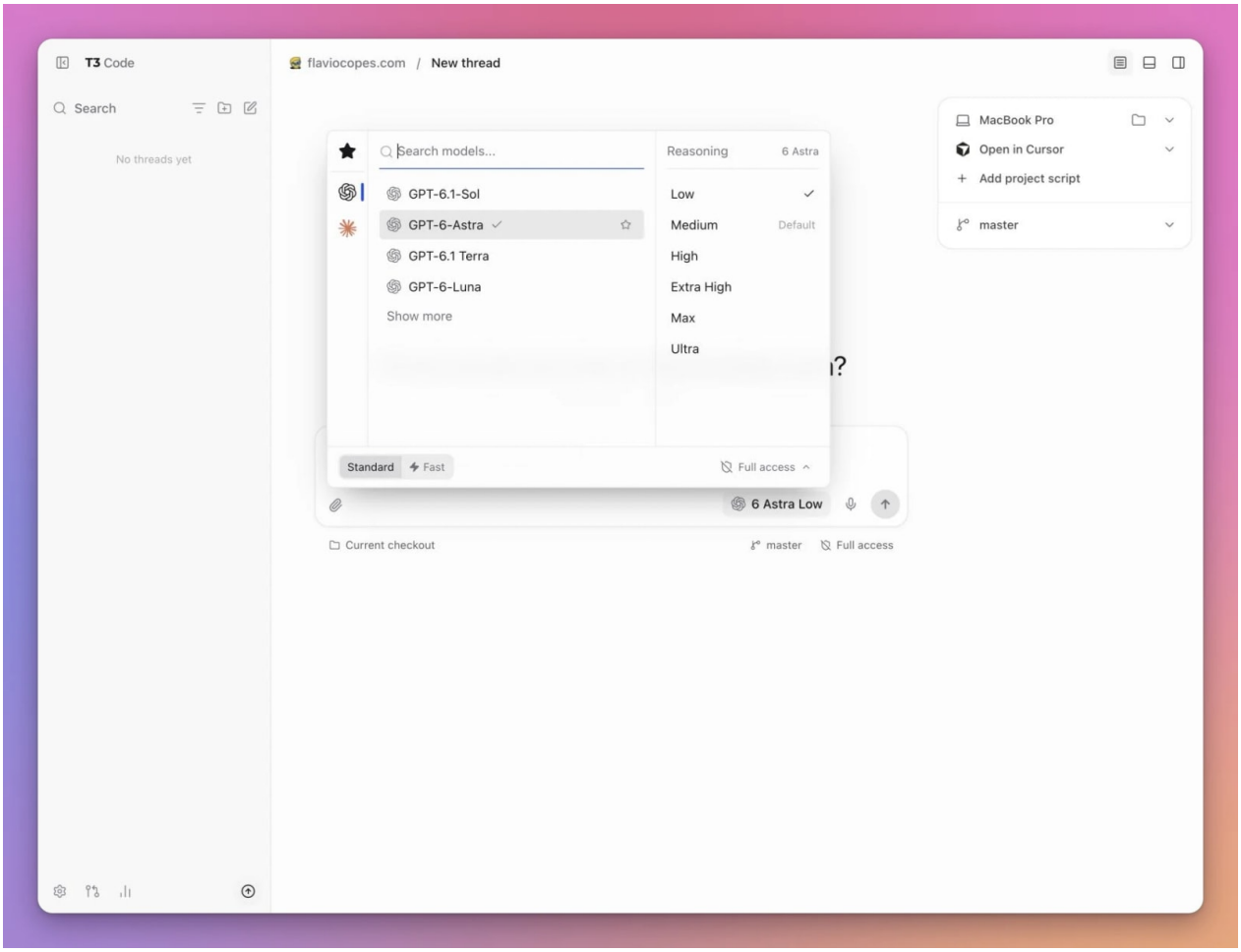
Add project lets you create a brand-new Git repository under `~/.t3/projects`, open a local folder, paste any Git URL (even a GitHub/GitLab branch or file URL), or browse connected hosts. Clones run in the background while you start typing your first prompt.



Inside a project, the composer brings the model, reasoning effort, permission mode, workspace, and branch together on one bar:



The model picker (`Cmd+Shift+M`) groups models by provider, supports search and favorites, and lets you move straight to reasoning effort and speed with the right arrow key:



For your very first run on a real repo, pick **Supervised** mode and start read-only:

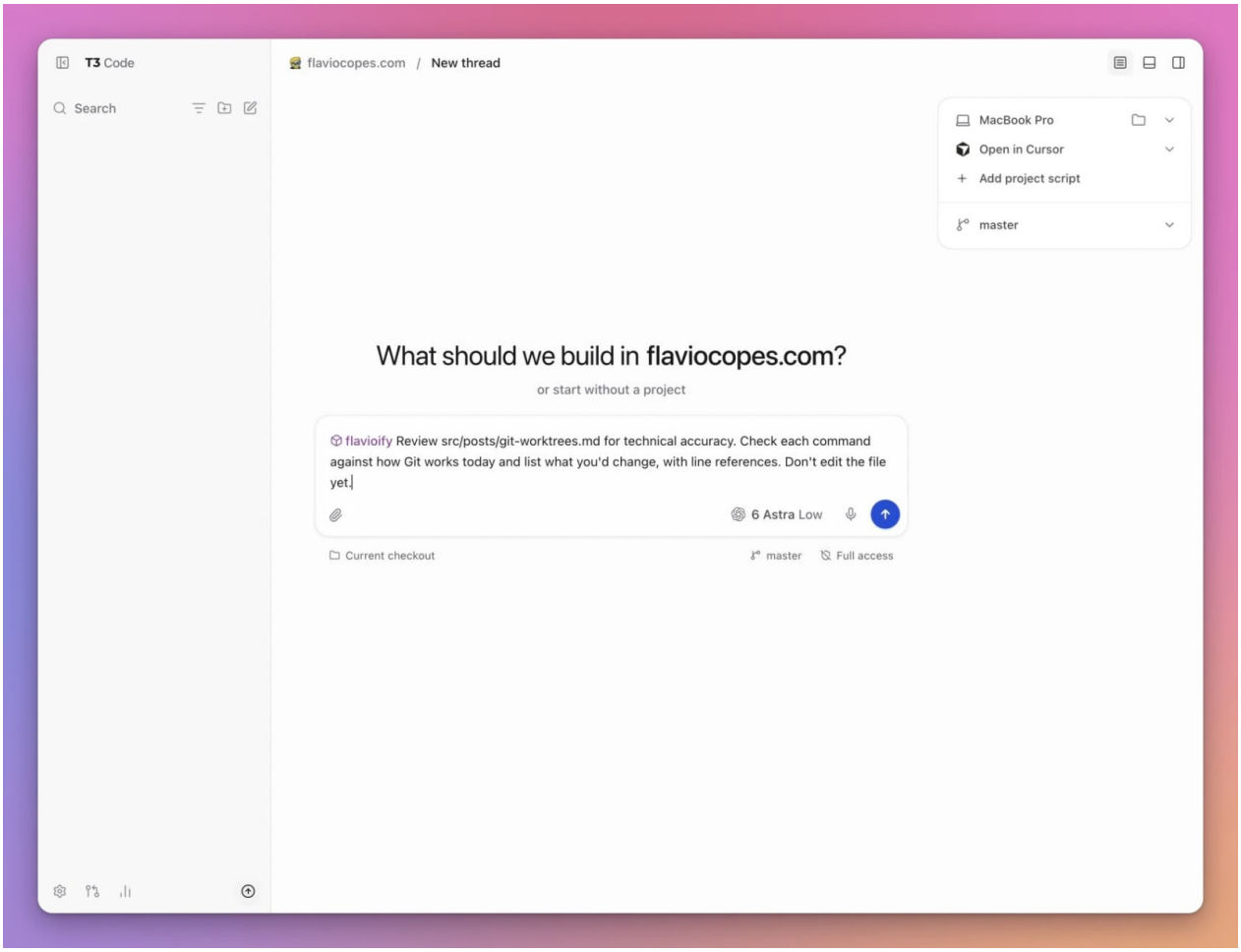
```
Read this repository and explain how a request moves from the HTTP route to the database. Do
```

That tests provider startup, file reads, streaming, and thread persistence safely. Follow it with one tiny edit and test run so you can inspect the per-turn diff and try **Edit from here**.

Composer power features

The composer handles far more than plain prompts:

- **Large prompts and attachments:** up to 120,000 characters and 100 files per message (10 MiB per image, 80 MiB total images; 50 MiB per other file). Pasting 32 KiB or more automatically turns the paste into a text-file attachment so it doesn't blow up context (`Cmd+Shift+V` keeps it inline). HEIC/HEIF photos from iPhone convert to JPEG automatically, and you can send a message while uploads finish.
- **Rich Markdown:** headings, lists, task checklists, and fenced code blocks style live as you type (turn off **Settings -> Chat -> Rich text composer** if you prefer raw Markdown).
- **Commands and skills:** type `/` for commands and `$` to search System, Personal, Project, and App skills. After installing new skills, plugins, or MCP servers, run **Restart agent session** from the command palette to pick them up without losing the thread.



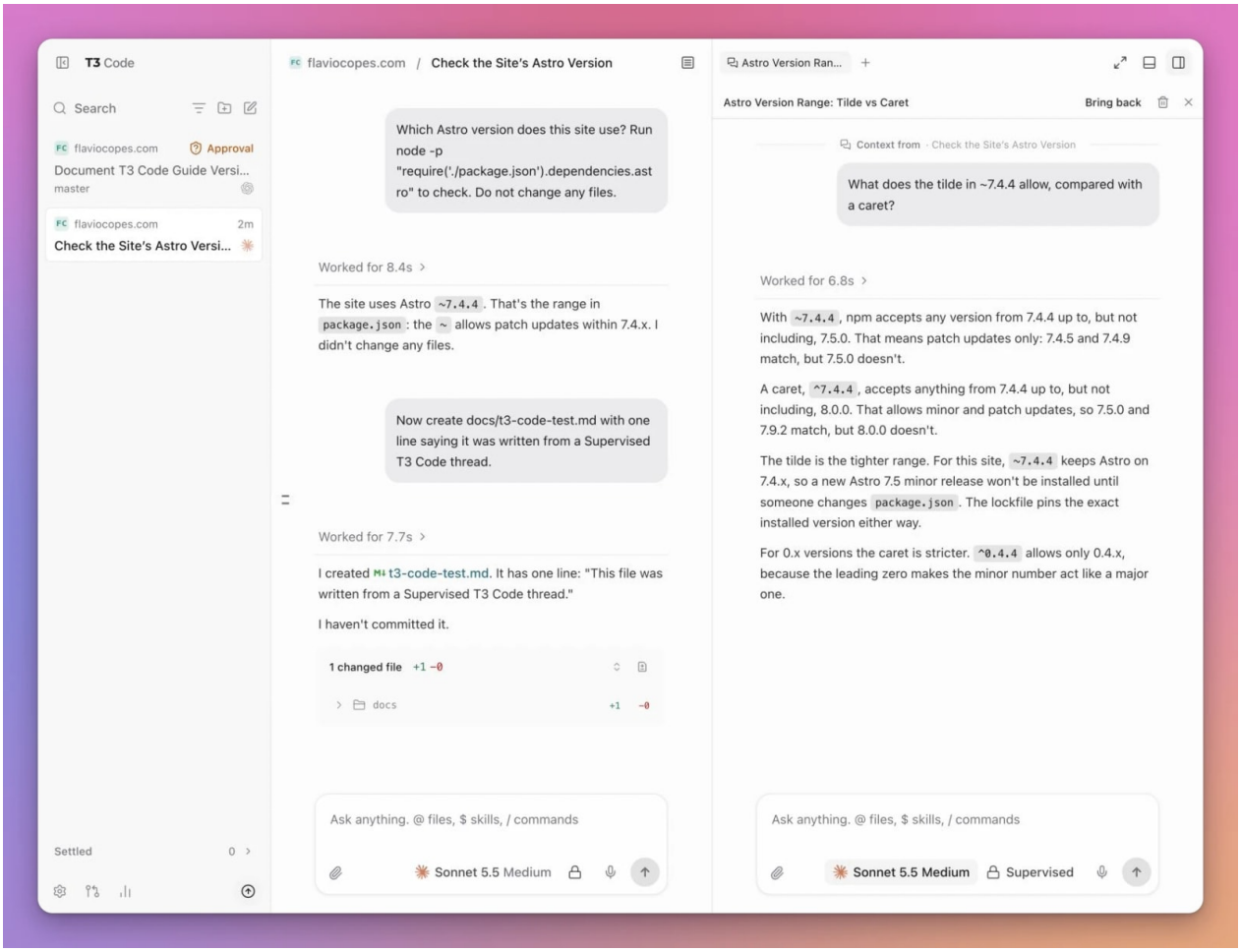
- **Context chips (`@`, `#`, citations, and files):**
  - Type `@` plus part of a thread title (or drag a thread from the sidebar) to reference another thread from the same server or a linked machine. The agent reads that thread's history only if needed, so attaching a long thread costs zero tokens up front.
  - Type `@` plus a model or provider name (like `@opus` or `@opus:opencode`) to tell an agent which model to delegate to without changing the current thread's model.
  - Type `#` to search recent or exact pull request numbers and insert them as live status chips.
  - Highlight any text in an assistant reply and choose **Cite in composer** (or long-press **Cite** on mobile) to quote that passage with an inline comment.
- **Session controls right in chat:**
  - `/clear` starts a fresh provider session inside the same thread while keeping earlier messages and diffs visible.
  - `/compact` shrinks context when the provider supports it.
  - `/usage-limits` checks remaining subscription limits for the active model without burning a turn.
  - `/goal <definition of done>` (Codex, Claude, OpenCode 2.2.5+, and Antigravity) keeps the agent iterating across turns until the goal is satisfied. Use `/goal clear`, `/goal pause`, and `/goal resume` on Codex and OpenCode.
- **Prompt stash & recall:** press `Cmd+S` (`Ctrl+S`) to stash a half-written draft and its attachments for later (24-hour upload retention); press `Cmd+S` in an empty composer to pop it back. Press `ArrowUp` in an empty composer to cycle previous prompts, or turn on **Settings -> Labs -> Suggest next prompt** for `Tab`-to-accept predictions on Claude and Codex.

Side chats (`/side` and `/btw`)

Start a prompt with `/side` or `/btw` to ask a tangent question without polluting or pausing the main thread:

```
/side Which file holds the retry logic?
```

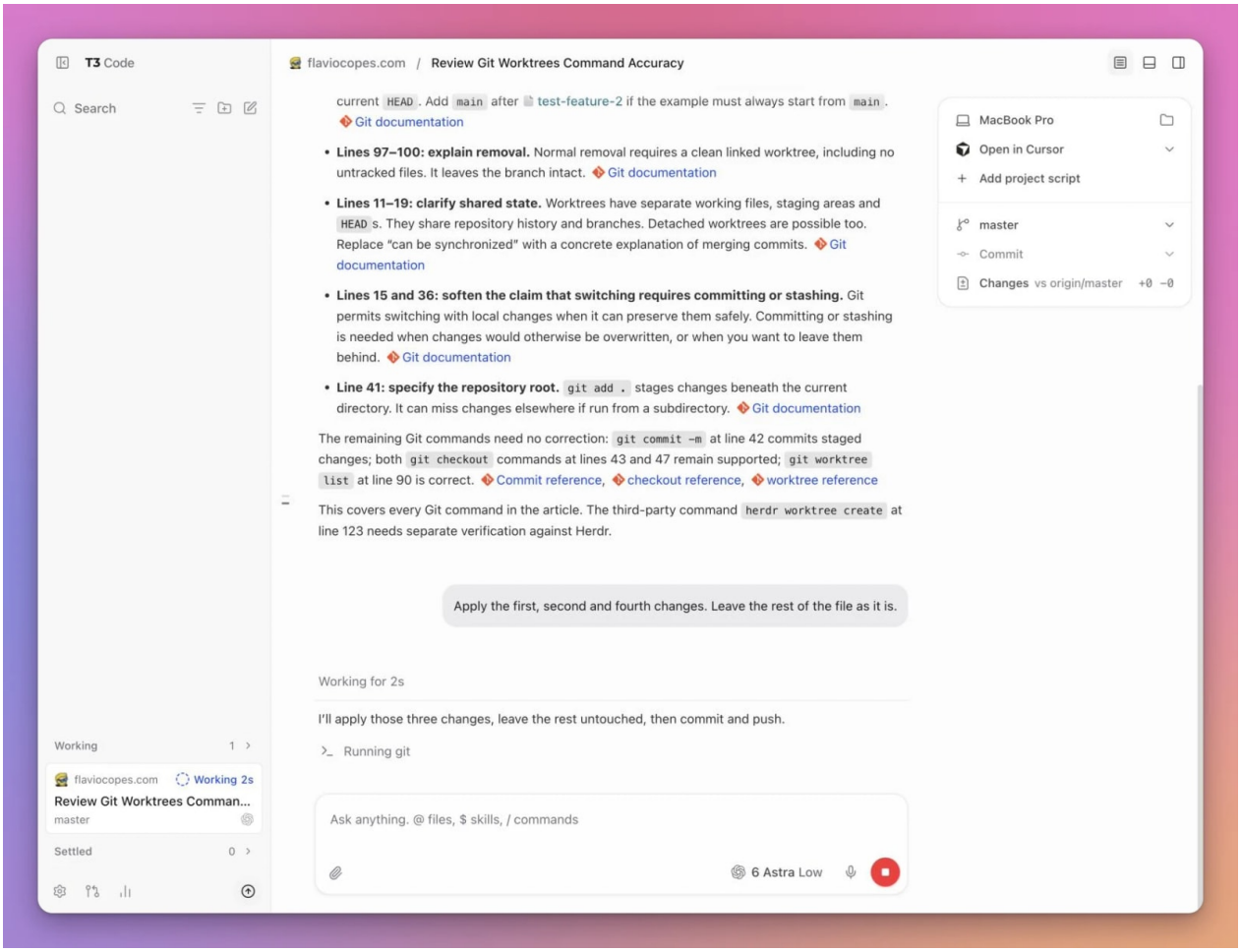
The side chat opens in the right panel with its own composer and never clutters the sidebar:



When you learn something useful, click **Bring back** to attach the side chat to your next message in the main thread (you can also reference side chats directly from the main thread since `v0.1.11`).

Background dispatch, server-side queue, and steering

Pressing `Cmd+Enter` (`Ctrl+Enter`) on a new thread, or `Cmd+Alt+Enter` on an existing one, fires the prompt in the background and immediately drops you into a fresh composer.



While a turn is running, **Settings -> Chat -> Follow-up behavior** decides whether sending another message **Queues** it for the next turn or **Steers** the active turn right away. Use `Cmd+Enter` to invert your default for one message, `Option+Up` to edit the latest queued message, and `Cmd+Shift+Enter` to promote the oldest queued message into an immediate steer.

Because the queue lives on the server, queued messages survive restarts. If you stop a turn or hit a usage limit, pressing **Resume** (or letting auto-resume kick in when the quota resets) uses native continuation on Claude, Codex, OpenCode, and Cursor without injecting a fake user message.

Voice input and dictation

Press `Cmd+Shift+D` (`Ctrl+Shift+D`) or click the microphone to dictate into the composer, diff annotations, PR comments, or browser annotations.

You choose explicitly where audio goes in **Settings -> Voice**:

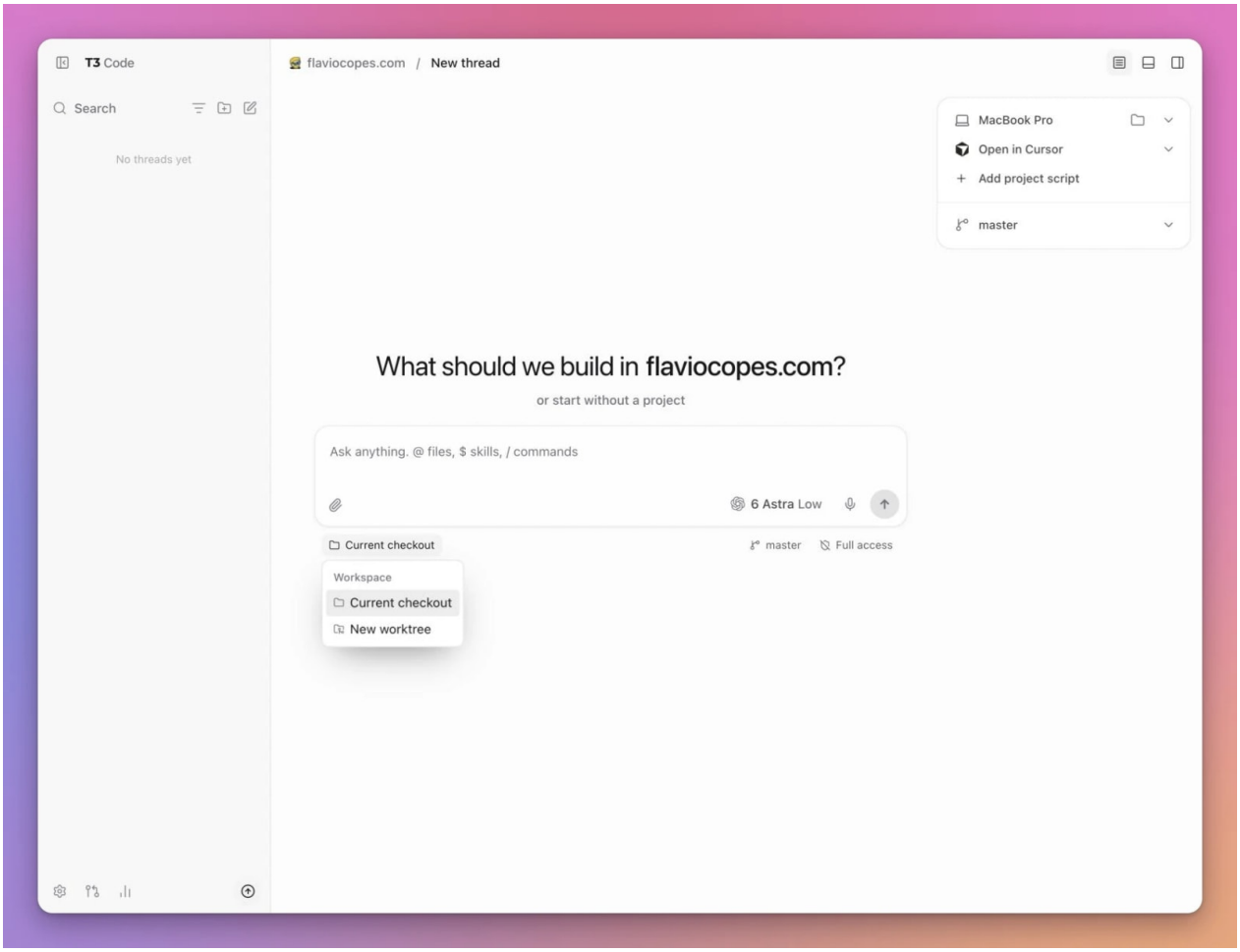
- **On-device: Apple Speech** on macOS 26+ (desktop app), iOS 26+, or Android 12+ with zero audio leaving the device.
- **Self-hosted on an environment:** download a local speech model onto your Mac or Linux server (T3 Code even reuses models already cached by Handy or Hugging Face) with GPU/CPU controls and streaming preview.
- **Cloud opt-in:** OpenAI, Grok Voice Transcribe, or OpenRouter (with zero data retention requested).

You can also maintain a **Project dictionary** so domain names and library jargon transcribe accurately, and turn on **Transcript cleanup** (with a custom correction cue like `"scratch that"`)

backed by a fast text generator.

Workspaces, worktrees, and multi-repo projects

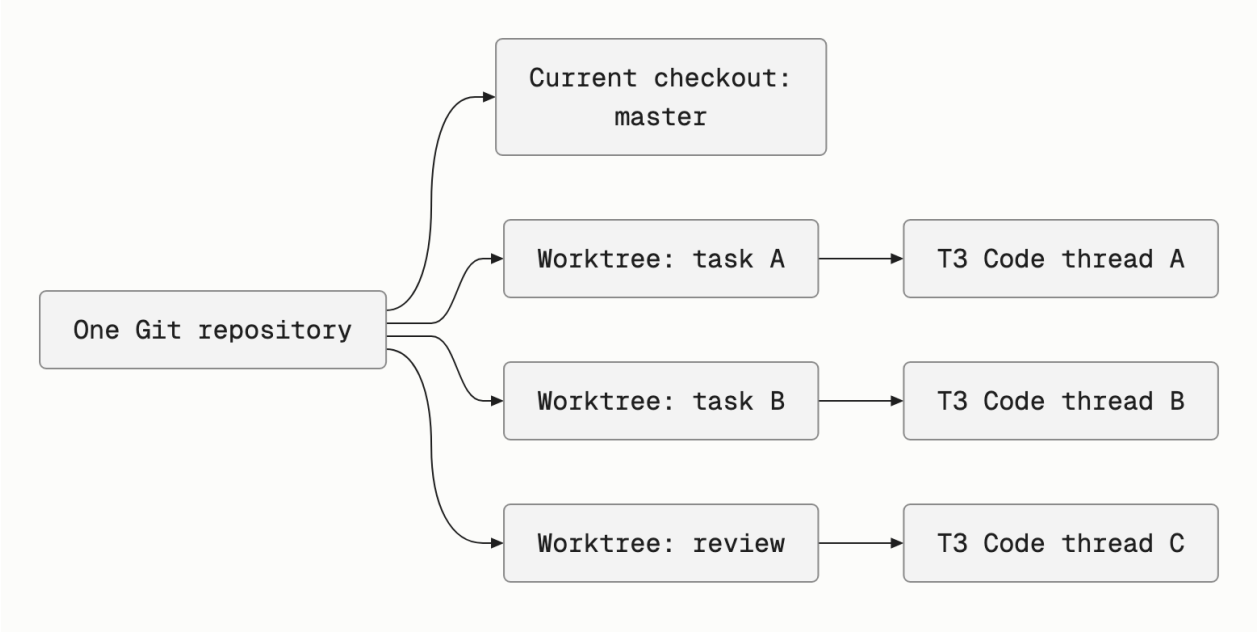
Every thread runs in a workspace: either the **current checkout** or an isolated **new Git worktree**.



Current checkout vs. isolated worktrees

Use **Current checkout** when you want one active writer, need the agent to see uncommitted local edits, or work in a repository whose rules forbid worktrees. Keep other threads in that checkout read-only so two agents never overwrite each other's files.

Use **New worktree** when running independent implementation tasks in parallel. Each worktree gets its own directory and branch while sharing Git's object database:



If Git worktrees are new to you, see my Git worktrees tutorial or the free Git course. Remember that worktrees isolate files and branches, not shared external resources like port 3000, a local Postgres database, or Docker container names.

T3 Code adds several practical worktree controls under **Settings -> Worktrees** and **Settings -> Labs**:

- **Worktree handoff:** if a thread started in the current checkout and turns into a bigger edit, ask the agent to move the thread into a new git worktree (`t3_worktree_handoff`); T3 Code creates the branch and worktree, runs your setup script, and continues there.
- **Fast worktree checkout on macOS (Labs):** uses APFS copy-on-write clones from a warm local cache (Git 2.42+), roughly doubling worktree creation speed and keeping the first `git status` instant.
- **Custom branch naming:** pick a static prefix (`t3/`), automatic semantic prefixes (`feat/`, `fix/`), or custom prompt instructions (such as including issue IDs).
- **Large repositories:** tune `"worktreeSubmodules"` (`recursive`, `top-level`, or `none`) or pass `"worktreeSparsePatternsPath"` in `t3.json` for Git sparse checkouts (Git 2.35+).
- **Automatic cleanup:** delete T3-managed worktrees after a thread settles, merges into the default branch, or goes inactive, while protecting uncommitted tracked/untracked changes.
- **Keep default branch fresh:** enable **Automatically pull** under **Settings -> Source control & issues** to fast-forward clean default-branch checkouts automatically.

Multi-repo projects

A T3 Code project does not have to be a single Git repo. It automatically discovers repositories inside a plain folder (like `shop/api` and `shop/web`), one level deeper inside `src` or `packages`, and inside submodules. You can also attach external repositories on the machine via **Settings -> Projects -> Repositories -> Add folder**, import folders from a `.code-workspace` file, or run:

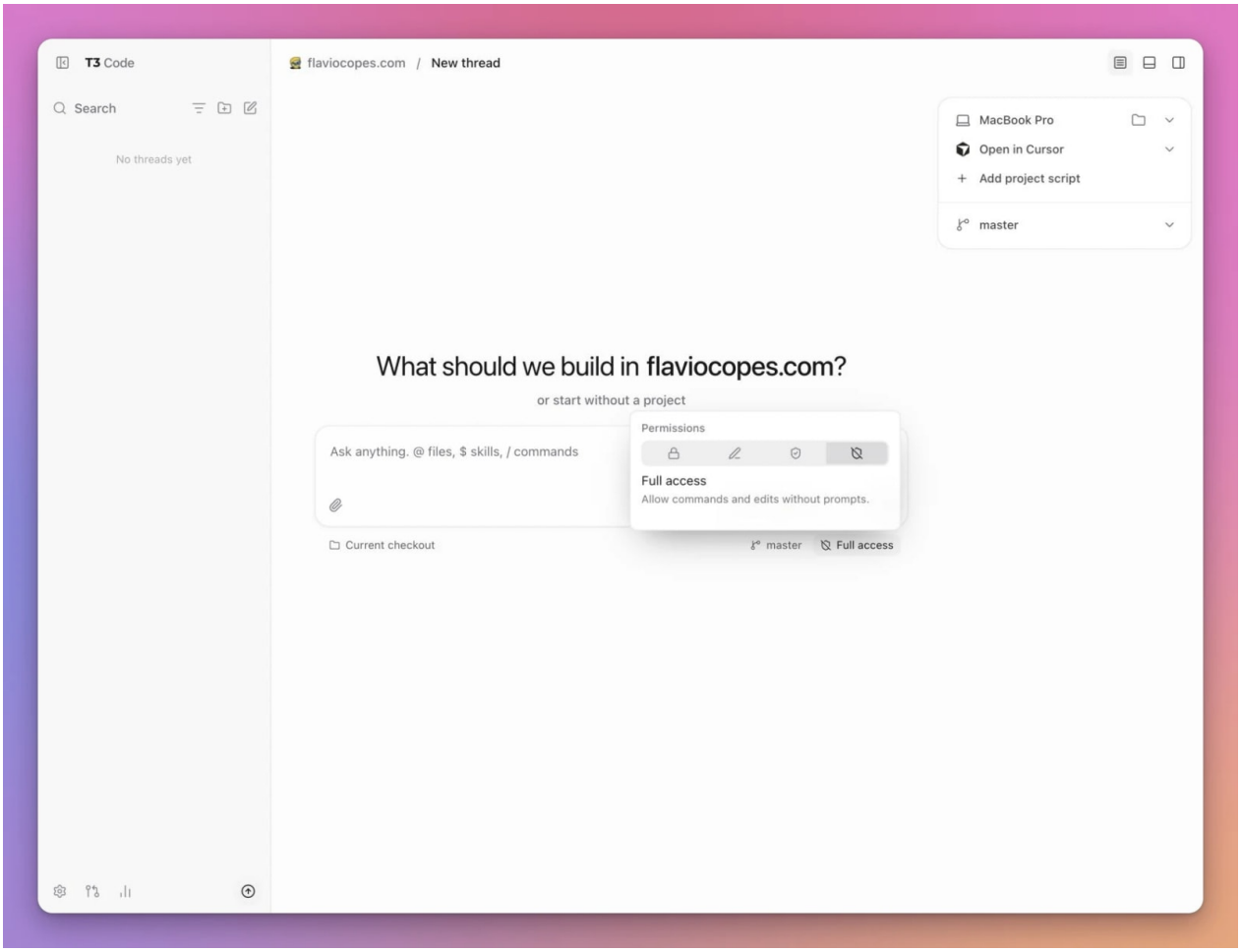
```
t3 project add ~/dev/shop --track ~/dev/shared-types
```

Exclude noisy folders across every machine by adding `"ignoreRepositories": ["vendor/big"]` to `t3.json`. In multi-repo threads, turn diffs group files by repository, the Diff pane adds a repository switcher, and **New worktree** creates matched worktrees across the selected repositories on the

same branch (or calls your custom `worktreeLayout` `create`/`remove` shell hooks in `t3.json`).

Permission modes, approvals, and checkpoints

T3 Code separates **permission mode** (what an agent can run without asking) from **interaction mode** (Normal vs. Plan).



Mode	File edits	Shell commands & actions	Best use
Supervised	Ask first	Ask first	Valuable checkouts, unfamiliar repos, learning T3 Code
Auto-accept edits	Automatic	Ask when needed	Routine code edits while gating shell commands
Auto	Provider review	Routine runs; risky actions ask	Everyday trusted work in a controlled repo
Full access	No prompts	No prompts	Disposable worktrees or isolated environments

The initial default for new threads is **Full access**. Change your global default under **Settings -> Threads -> New threads -> Permissions** (`Cmd+Shift+A` toggles it per thread), or lock

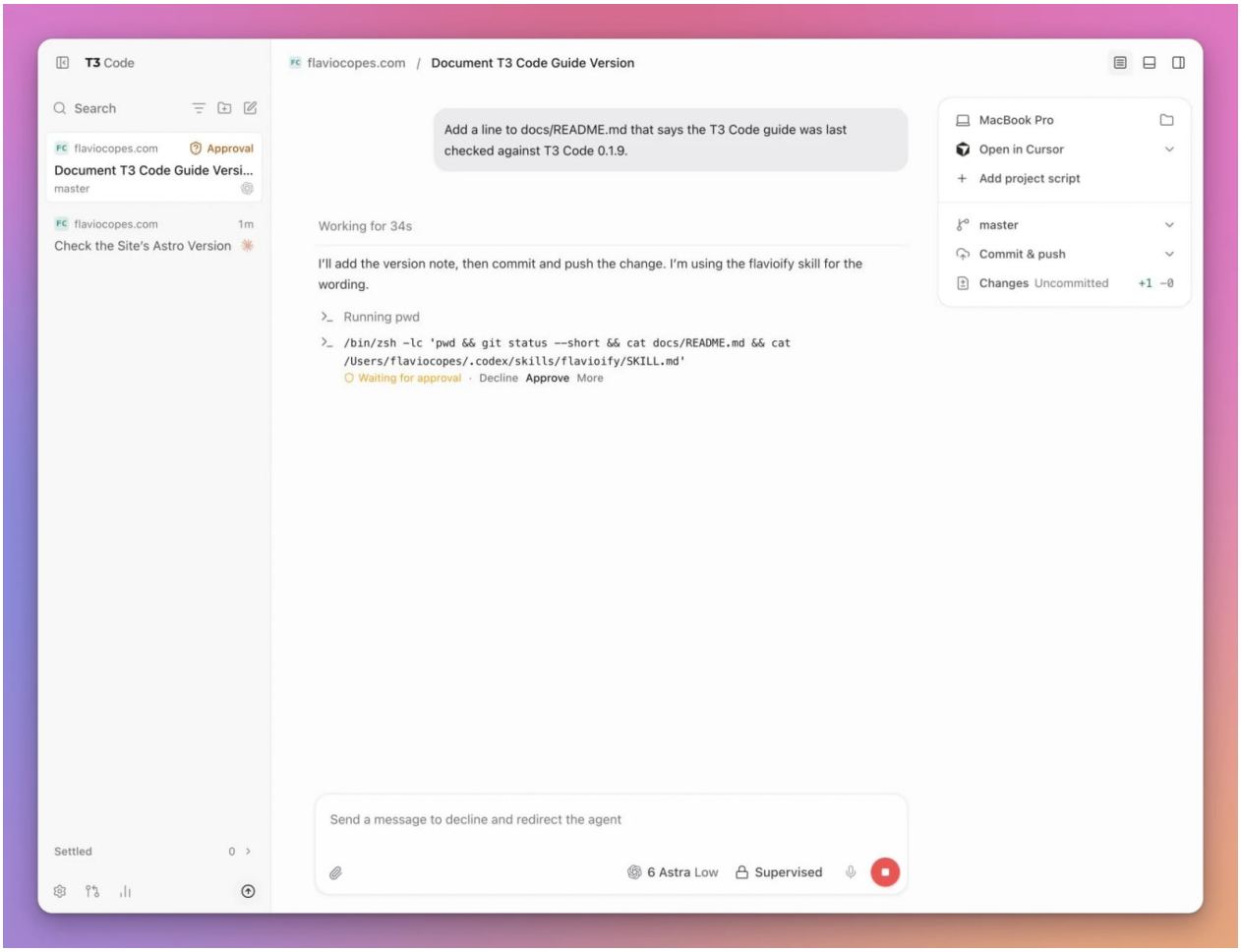
a stricter default per project. Administrators on shared machines can enforce hard ceilings via a managed policy.

Keep provider nuances in mind:

- **Auto mode** uses the provider's native safety reviewer on Codex, Claude, Cursor, Grok, Antigravity (when the Google account includes Antigravity Auto), and fx. OpenCode falls back to asking.
- **Full access** still respects native safety guards: Claude still checks dangerous-removal rules (and any Claude settings that disable bypass permissions), Codex still checks restricted network destinations, and Antigravity can still send native approval prompts.
- **Grok, Muse Code, Kiro, and fx** each expose only the subset of modes their CLI supports; switching to a mode a provider lacks safely drops to **Supervised**.

Approvals from any device

When an agent asks to run a command or edit a file, the thread pauses with an **Approval** badge in the sidebar and inline **Decline** / **Approve** buttons:



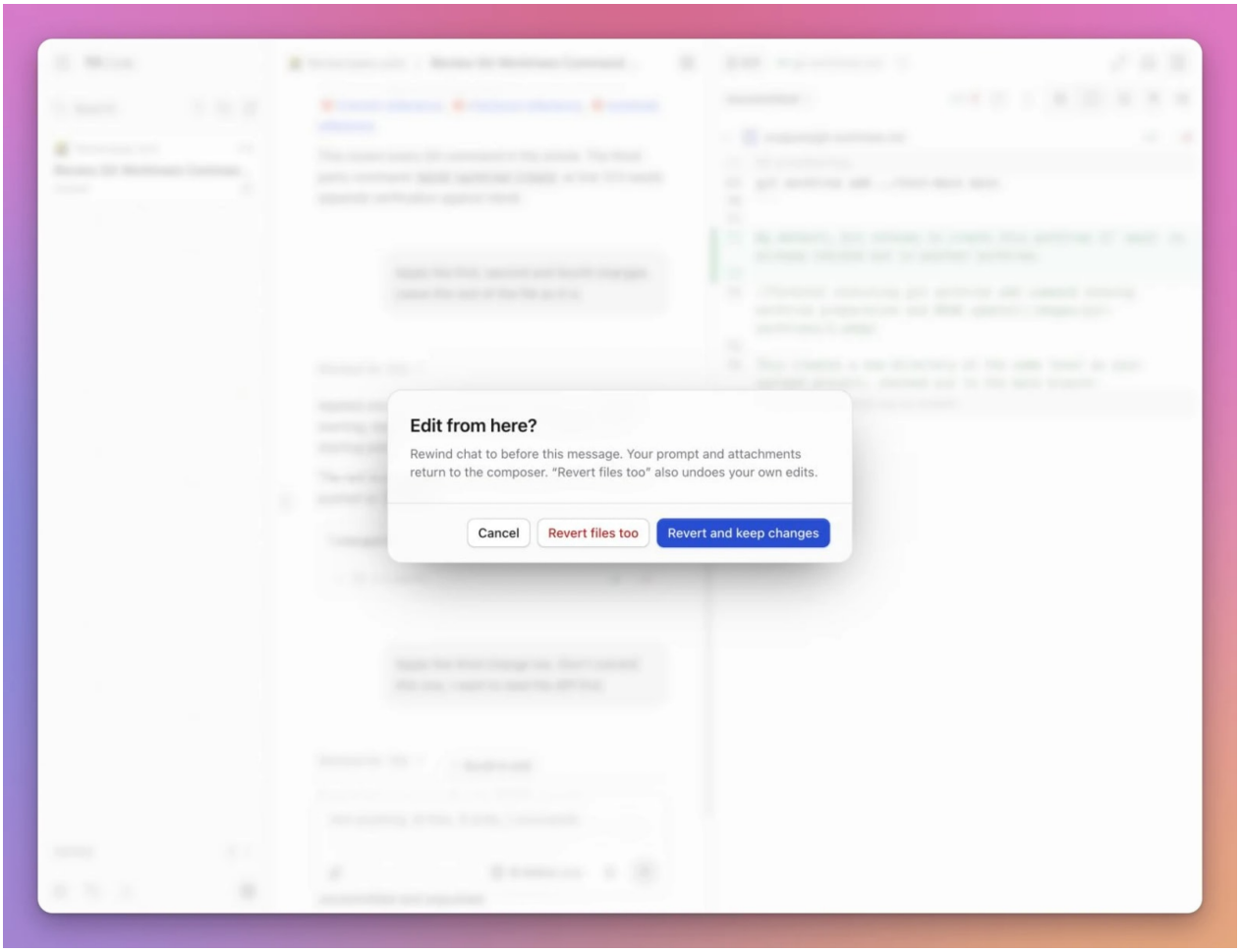
You can approve or decline from desktop, web, or mobile. Typing a reply into the composer while a request waits declines that request and redirects the agent (including for a delegated subagent, whose approval card appears right inside the parent conversation). For Claude, you can approve an MCP app once for the entire session.

Plan mode vs. default mode

Switch the interaction mode to **Plan** when a task touches several subsystems or when a wrong first assumption would create rework. Because planning uses the provider's native planning capability, the agent inspects the repo and proposes a concrete plan before touching files. Skip it for one-line fixes where planning adds more ceremony than clarity.

Checkpoints and Edit from here

T3 Code saves workspace checkpoints as hidden Git refs around every turn. Beneath any earlier user message, click **Edit from here** to rewind:



- **Revert files too** restores the workspace checkpoint and rewinds the conversation (including any manual edits you made after that turn). File restore is available only in isolated worktrees and refuses to run if another active thread shares that directory.
- **Revert and keep changes** rewinds only the conversation while leaving the files on disk.

On providers without native rewind (Antigravity and ACP Registry agents), T3 Code restores the files and starts a fresh session carrying the kept conversation history. Treat checkpoints as fast local undo, and still commit and push durable milestones to Git.

Multi-provider orchestration and portable handoffs

Most developers start using T3 Code as a tabbed inbox for separate chats. The bigger unlock is combining providers inside one workflow.

Video walkthrough: Watch the embedded video in the online edition at https://

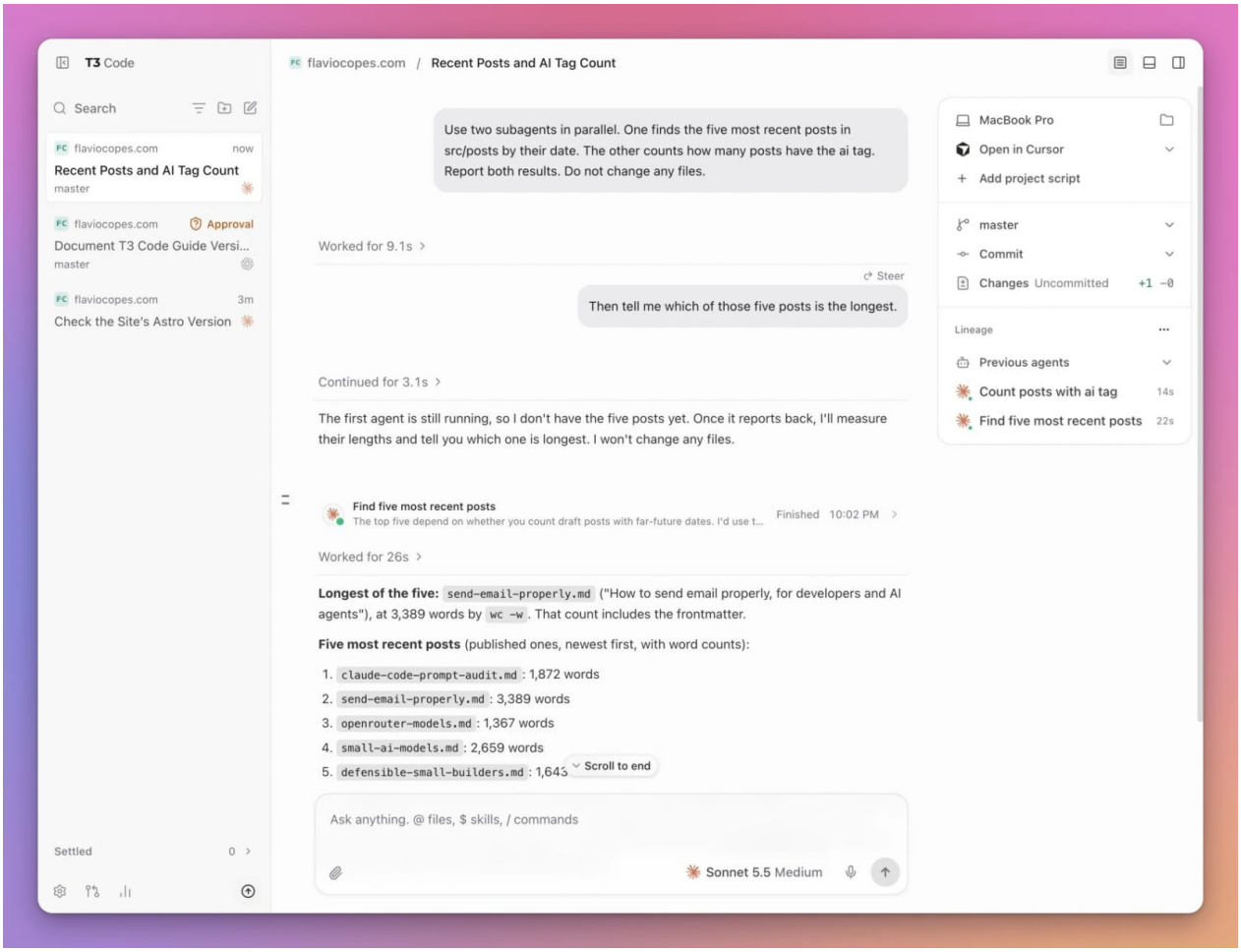
flaviocopes.com/t3-code/.

Cross-provider delegation (`delegate_task`)

Inside any supported thread, the agent can call `delegate_task` to spin up child tasks on other enabled providers, optionally in their own worktrees. In his walkthrough, Theo asks a Codex thread to review the T3 Code repo, and it dispatches four parallel reviewers across Codex, Muse Code, Antigravity, and Claude Code.

You track delegated children from the thread's **Agents** view, answer their approval requests right in the parent thread, and let the parent wake up automatically once every child returns its result.

Providers that support native subagents (like Claude Code) also show structured subagent cards in the timeline and under **Lineage** in thread details, including elapsed time, model, effort, and tools used:



Switching providers mid-thread and portable handoffs

You can switch model or provider right in the middle of a thread (watch Theo switch mid-test on an iOS device). The timeline marks exactly where the model or effort level changed.

When you switch providers, fork onto another runtime, or hit a context limit, T3 Code performs a **portable handoff**:



- Short threads transfer intact. Longer threads send a **budgeted selection** (default cap `T3CODE_CONTEXT_HANDOFF_TOKEN_CAP=16000`, tunable from 1,024 to 64,000) prioritizing your original request, recent turns, and key command outcomes.
- Your newest prompt is **never truncated** to make old history fit.
- The outgoing provider's private reasoning, tool-call internals, and attachments stay behind, while the handoff includes references to omitted messages so the new agent can pull exact earlier turns on demand with T3's thread-reading tool.
- If a target session runs out of context room, T3 Code automatically opens a fresh provider conversation with a bounded history slice and sends your prompt without making you compact manually.

Forking, context transfer, and Activity Log modes

You can **fork** any thread from its latest stable turn or an earlier checkpoint to test an alternative approach without disturbing the original branch, and later merge context back (`merge_back`) into a related thread in the same project.

In busy threads, consecutive tool calls fold into compact summaries in the **Activity log** (like *Ran 2 commands and sent messages to 3 threads*). Under **Settings -> Chat -> Tool calls in chat**, you can keep everything visible, choose **Hide successful** (show live calls, keep only failures), **Hide all**, or pick **Zen** mode to show only your prompt and the agent's final reply while keeping approvals,

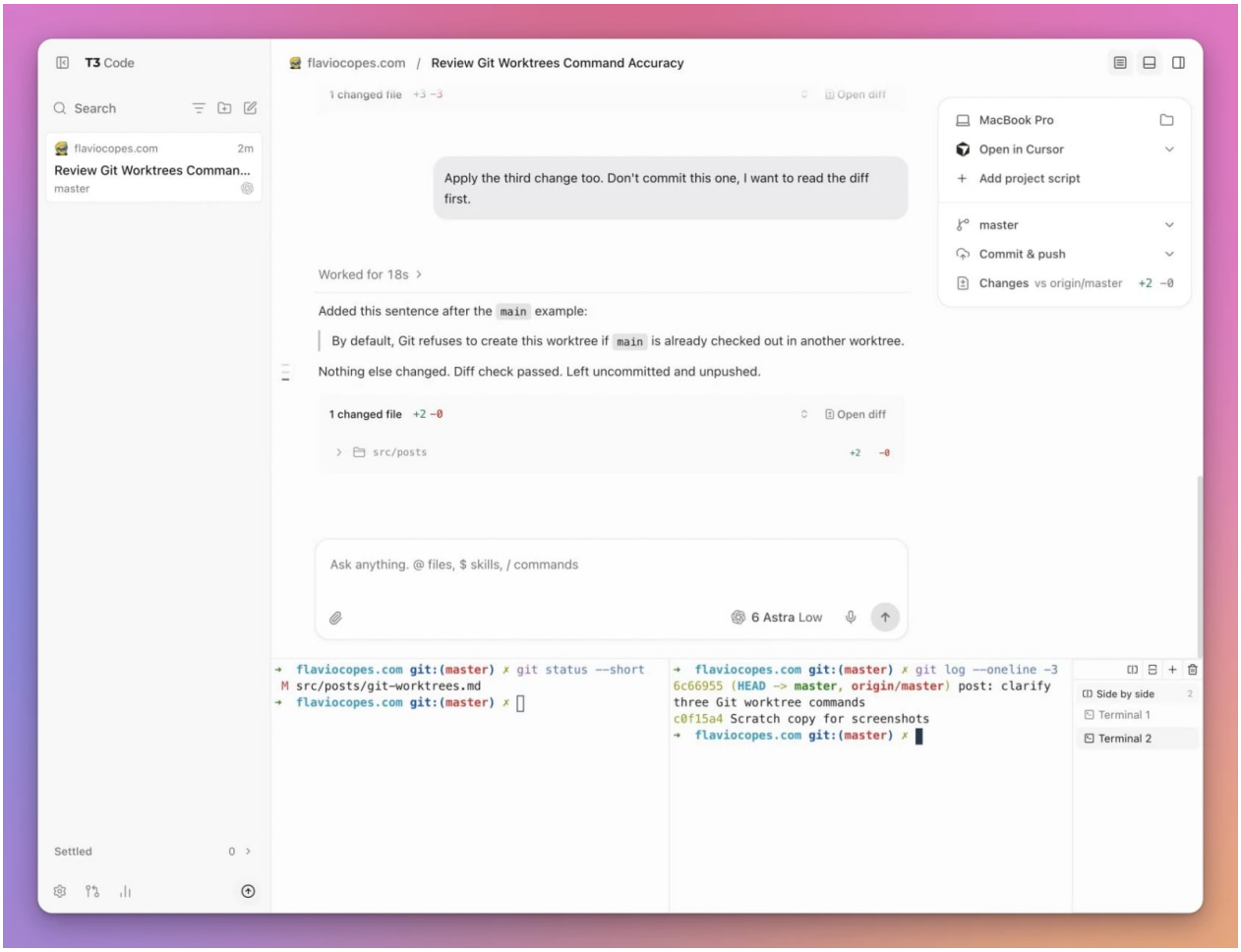
questions, and errors visible. **Settings -> Chat -> Screen reader announcements** can also narrate turn status and steps live without leaving the composer.

Review changes: files, diffs, visual replies, browser, and devices

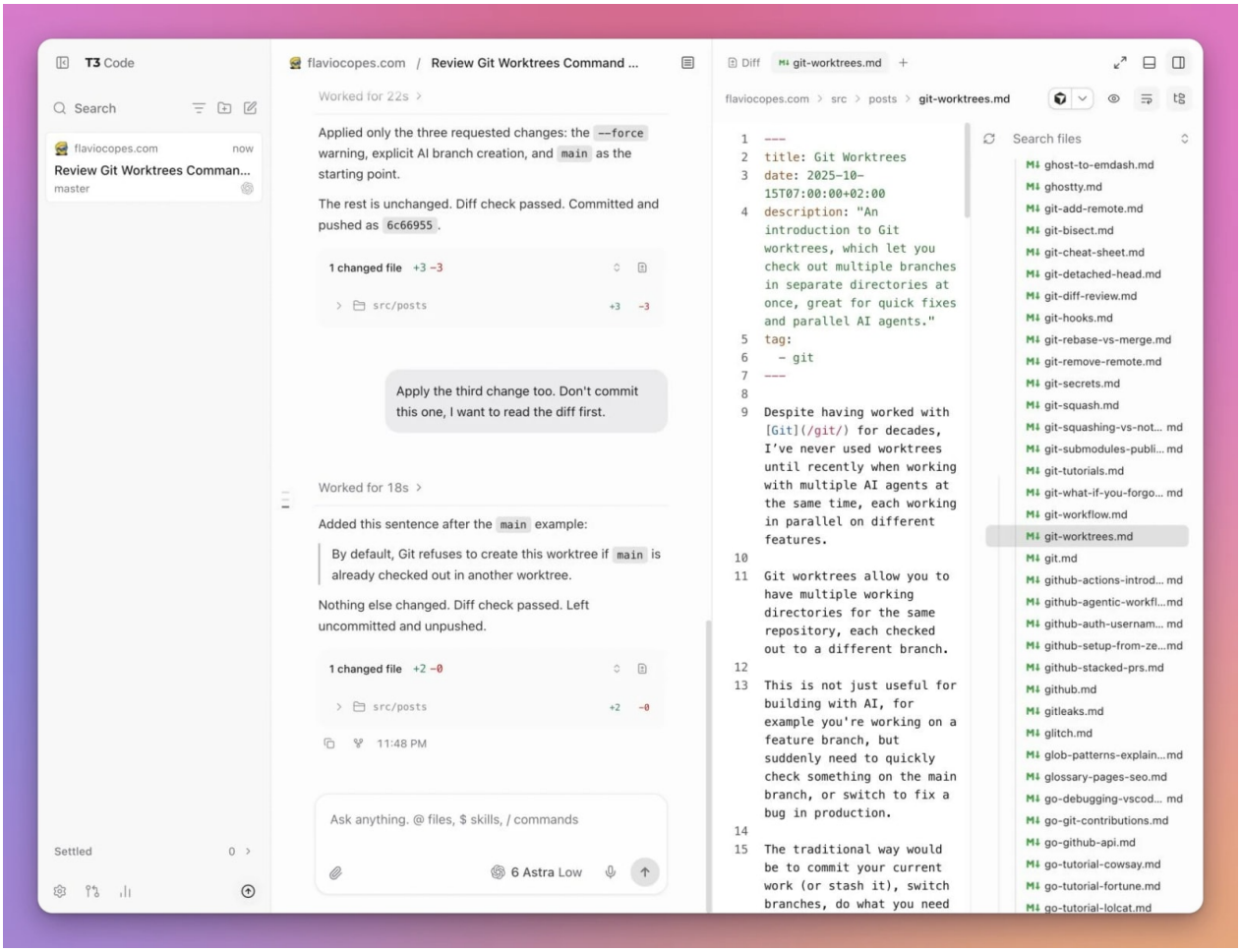
Reading an agent's summary is never enough. T3 Code keeps terminals, files, multi-scope diffs, visual renders, a full browser, and mobile simulators right beside the conversation.

Terminals and file browser

Each thread has its own terminal drawer running on the server machine in the thread's directory (even when you trigger it from your phone):



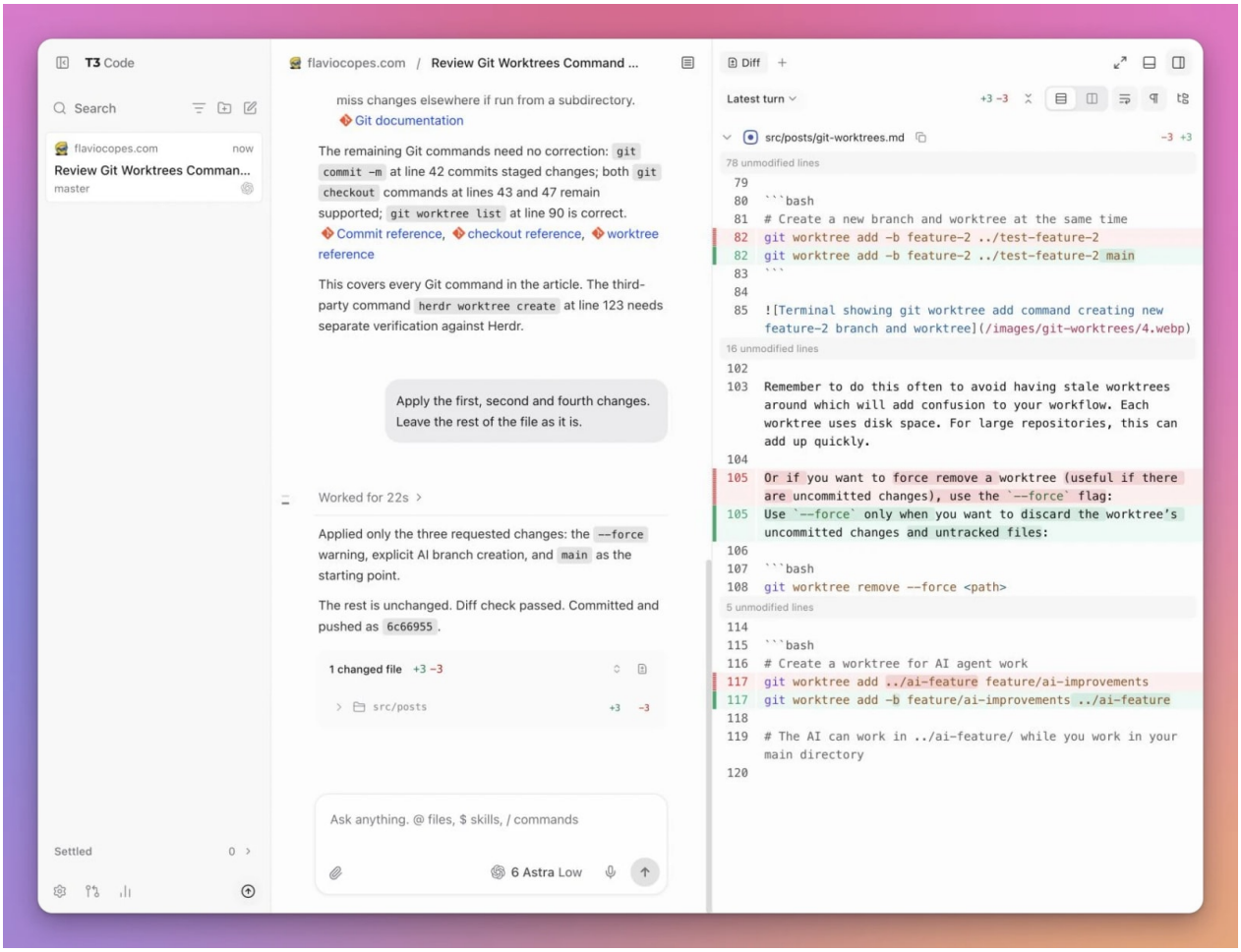
The file panel browses the workspace, renders Markdown, images, HTML, PDFs, Office docs, and ZIP archives (up to 50 MB), opens read-only agent links to files outside the workspace, and lets you select lines to attach an inline comment directly to the composer:



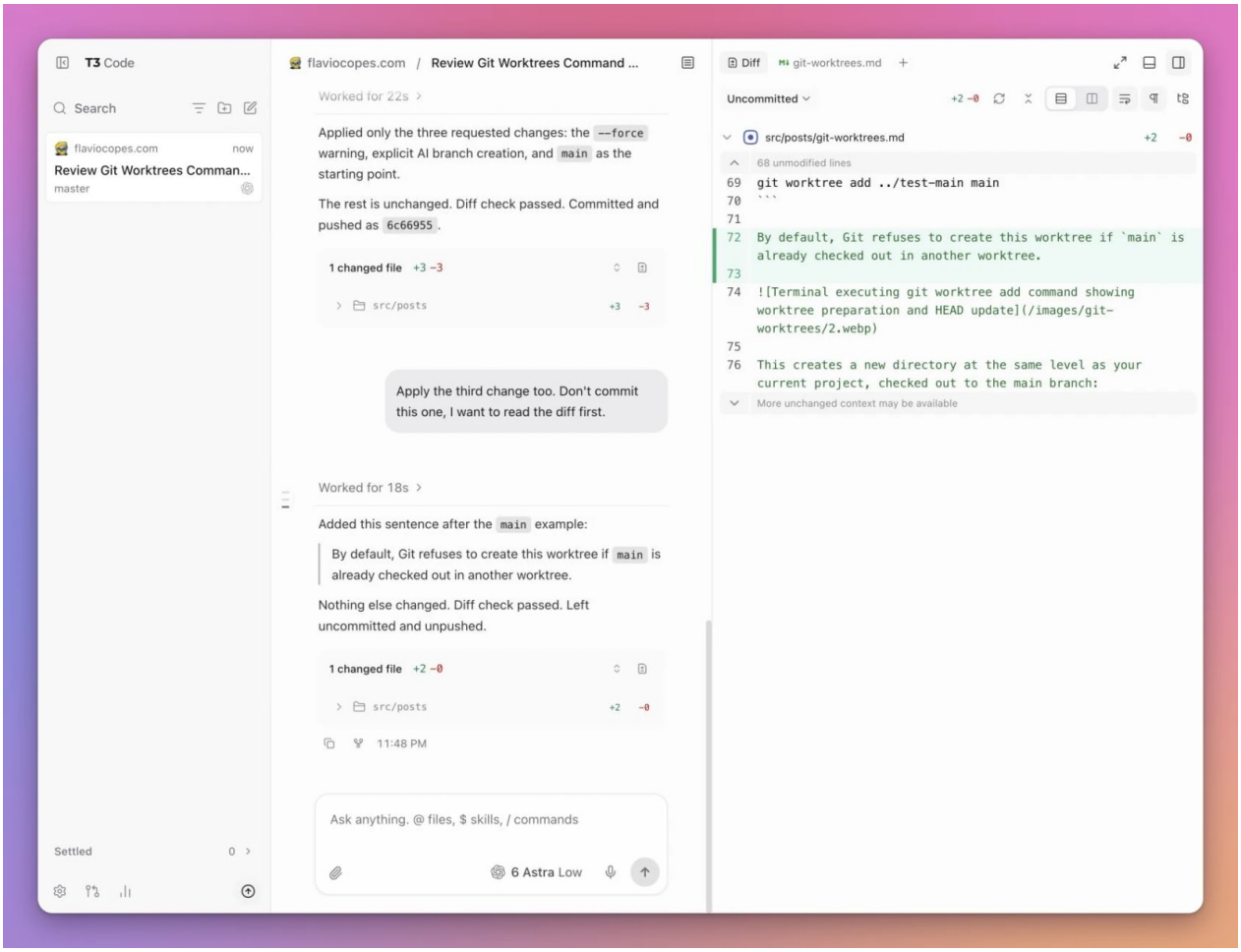
Four diff scopes (plus global diff search)

The Diff panel gives you four scopes so you always answer the right question:

- **Latest turn / Turn:** only what that specific prompt changed (pulling or rebasing doesn't pollute the list with unrelated upstream files).



- **Commits:** inspect one local commit at a time (up to the newest 100 commits in the branch comparison).
- **Uncommitted:** everything currently modified on disk across turns.



- **Changes (Branch):** the full diff against your base branch before committing or opening a PR.

Click into any diff and press `Cmd+F` (`Ctrl+F`) to search across *every* file in the diff, including folded files and hidden unchanged lines. Select any lines in a diff to drop a review comment chip directly into your next prompt.

Visual replies and inline MCP Apps

Instead of replying with walls of plain text, any agent on any provider can publish a **Visual reply**: a self-contained, sandboxed HTML page rendered right inside the thread (above the written summary) on web, desktop, and mobile. Ask for a chart, comparison table, interactive mockup, or collage of screenshots; it adapts to your light/dark theme, embeds local images permanently, and supports full-screen zoom.

On **Codex** and **Claude Code** (2.1.301+), MCP servers that implement the MCP Apps spec also render interactive mini-apps inline beneath their tool call. Read-only server calls run without interruption, while mutations and chat posts ask for approval first.

Built-in browser, remote tunnels, port forwarding, and session import

A project action with `previewUrl` opens your dev server right inside T3 Code's browser panel, and the **Servers** list automatically detects ports opened by your agents and terminals.

> **Video walkthrough:** Watch the embedded video in the online edition at https://flaviocopes.com/t3-code/.

Under the hood, the in-app browser is far more than an iframe:

- **Shared agent browser:** agents can snapshot the accessibility tree, click elements, run JavaScript, test mobile viewport presets (iPhone, Pixel, iPad), flip light/dark mode, and **record preview videos**. You can take control of an agent's tab to type or sign in, then release control so the agent keeps going.
- **Browser tunnels and one-click port forwarding (`v0.1.11`):** when an agent starts a dev server on a *remote* machine, opening it in your desktop app loads it locally while tunneling `localhost` traffic straight to the remote server. Want that server in Chrome, `curl`, or a database client on your laptop? Click the forward button in **Servers**, `Cmd-click` the remote server, or open **Forwarded ports** in the command palette to forward the remote port to your local loopback in one click. On iPhone, iPad, or Android, choose **Open on this device** on any remote `localhost` page for native touch scrolling.
- **Import signed-in browser profiles:** open **Settings -> Browser -> Browser profiles -> Add profile -> Import from** to do a one-time cookie import from Safari, Firefox, Zen, or Helium so agents can test authenticated pages without logging in from scratch.
- **Default browser on macOS:** you can even set **T3 Code** as your macOS default web browser so clicked links open in a browser panel beside a ready agent.
- **Linux headless setup:** macOS and Windows work out of the box; on Ubuntu 23.10+ or minimal Linux servers, run `sudo t3 browser setup` once to install missing libraries, fonts, and the Chrome sandbox AppArmor profile.

Live iOS Simulators and Android Emulators (`Devices`)

Open the right panel and choose **Device** to watch and tap a live iOS Simulator (macOS with Xcode) or Android Emulator (Android SDK + `ffmpeg` with `libx264` for emulator video) directly beside your thread, or float it in a resizable window over chat.

> **Video walkthrough:** Watch the embedded video in the online edition at https://flaviocopes.com/t3-code/.

Key details that make it genuinely practical:

- **Interactive Tools drawer:** toggle light/dark mode, scale text size, overlay accessibility element frames, set fake GPS coordinates, flip app permissions, send test iOS push notifications, or fold/unfold supported foldables in interactive **3D view**.
- **Agent control:** turning on **Agents can use devices** under **Settings -> Devices** hands agents the `device_*` tools and the `agent-device` CLI so they can boot simulators, tap through screens, take screenshots, and verify fixes automatically.
- **SSH device hosts:** even if your main T3 server is on Linux, you can add a Mac over SSH under **Settings -> Devices -> Add source -> Machine over SSH** to stream its iOS Simulators directly into your threads.

SnapShots: global window capture with accessibility data

Turn on **Settings -> SnapShots** in the desktop app (macOS, Windows, or Linux Wayland) and press the global shortcut (both **Shift** keys together by default on macOS and Windows) from *any* app on your computer.

T3 Code grabs the active window screenshot, flies it straight into your current draft, and when **Include app text** is on, attaches the window's **accessibility tree** (app name, window title, controls, visible text, and coordinates). Instead of making the vision model guess tiny UI labels in a

screenshot, the agent receives both the pixels and the structured UI hierarchy.

Issues, pull requests, review guides, and PR watch

T3 Code connects issue trackers, branches, diffs, reviews, and merges into one continuous loop.

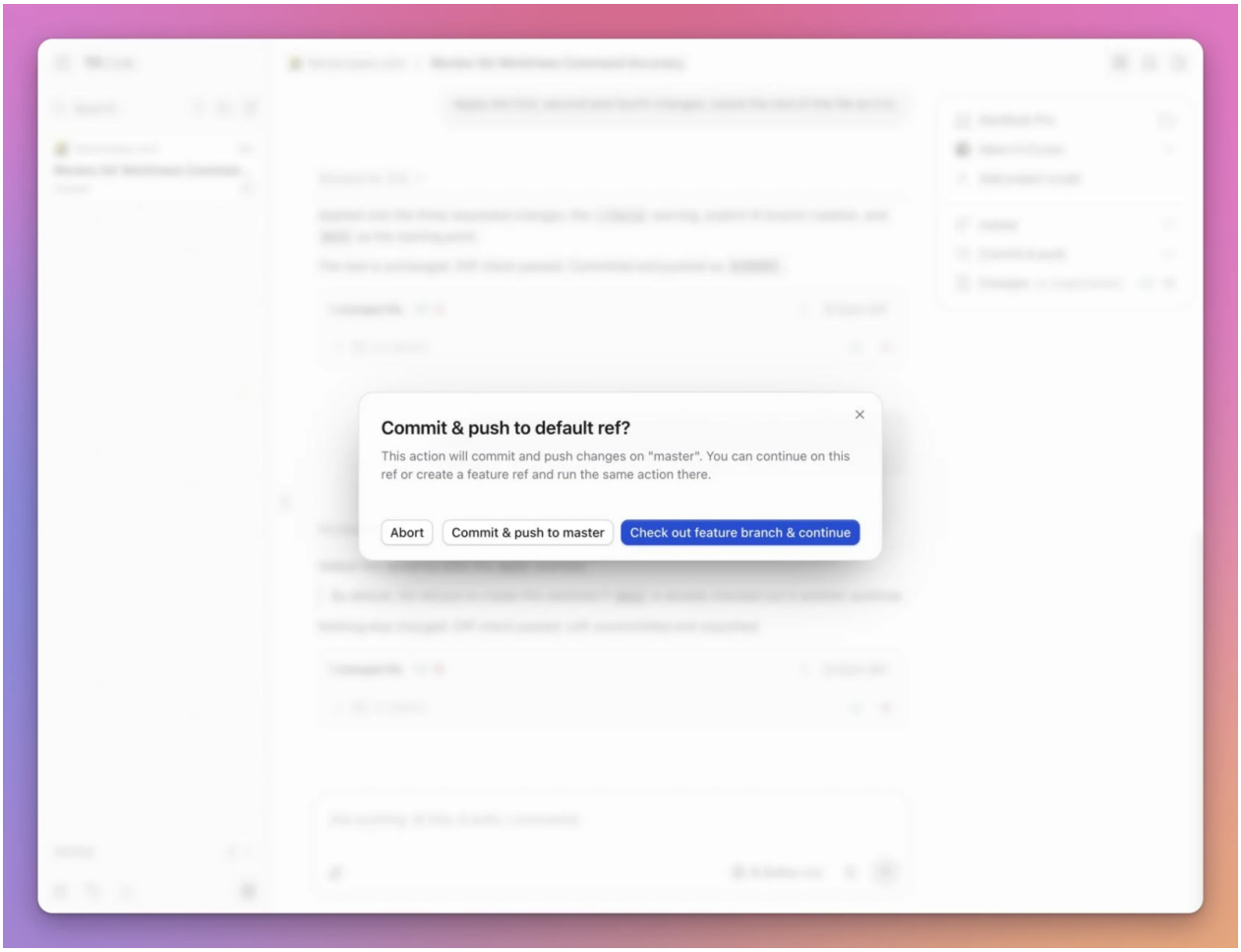


Nine supported source control hosts

Configure Git and your host credentials on the server machine, then hit **Rescan** under **Settings -> Source control & issues**:

Host	How to authenticate on the server	Highlights
GitHub (incl. Enterprise)	Token in Settings, `GH_TOKEN`, or `gh auth login` (2.81+)	Native API, PR stacks, viewed-file sync, merge queue, fork workflow approvals
GitLab	`glab auth login`	Merge requests, auto-merge, one-click revert of merged MRs
Forgejo / Gitea	`fj` or `tea` (0.16+)	Shared integration (Codeberg included); uses `tea` for subpath URLs
Bitbucket	Access token or Atlassian email + API token in Settings	Full clone, publish, and PR review
Azure DevOps	`az extension add --name azure-devops` + `az login`	PR listing, auto-merge (comments are read-only)
GitCafe	`cafe auth login` or token in Settings	Stack merges and restacking
Cursor Origin	`origin auth login`	Private repos (`origin.cursor.com`); see my Cursor Origin post
TanStack Source	Private beta (`T3CODE_FEATURE_FLAGS=tanstack_source`); `src auth login`	Live CI logs, private PRs on `tanstack.dev` repos

T3 Code stops you before accidentally committing directly to the default branch, letting you move work onto a feature branch cleanly:



Under **Settings -> Source control & issues -> Commits & pull requests**, you can pick **Repository conventions** (reads your `AGENTS.md` and recent commits), conventional commits, or custom instructions, and toggle **Remove agent credits when merging** to strip agent `Co-Authored-By` trailers on GitHub merge/squash commits while keeping human co-authors intact.

The Issues page (`v0.1.11`)

Open **Issues** from the sidebar (or **Open issues** in the command palette) to browse open and closed issues across GitHub, GitLab, Azure DevOps, Forgejo, and **Linear** (connect Linear with an API key and bind one or more teams per project).

Filter by involvement or tracker, save reusable **Views**, edit titles/labels/assignees, comment, and link an issue straight to a thread (agents also call `link_issue` and `create_issue`). Clicking an issue badge in the sidebar opens the issue and its linked pull requests right beside the conversation.

PR Review Guides, viewed marks, and GitHub stacks

Opening **Pull requests** gives you saved filter views, offline-readable mobile PR diffs, and three tools that make reviewing agent-written code much faster:

> **Video walkthrough:** Watch the embedded video in the online edition at https://flaviocopes.com/t3-code/.

- **Review Guides (beta):** enable under **Settings -> Source control & issues** to add a **Guide** tab to every PR. Click **Write guide** (or ask an agent to prepare guides for a batch of PRs) and Claude or Codex inspects the diff in a temporary checkout to build a reading-order

tour. Important design calls get flagged **Look closely**, mechanical renames group at the end, and every changed file is accounted for. Select lines inside any stop to leave a review comment or click **Ask about this stop** to open a pre-contextualized thread.
- **Mark files as viewed:** check files off in the **Code** tab to collapse them; if a new push touches a viewed file, it pops back up marked **Changed**. On GitHub, viewed ticks sync both ways with github.com.
- **GitHub stacks:** navigate stacked PR layers from the stack badge, **Rebase stack** remotely from bottom to top, or **Merge stack** to land a PR and every unmerged layer beneath it together.
- **Pull request sharing:** if you have several connected machines signed into the same host account, enable **Read PRs** or **Read and act** under **Settings -> Environments -> Pull request sharing** so a fast local environment can serve PR data for slower remote environments without ever moving credentials off their machines.

Watch a pull request

Ask an agent to watch a linked pull request (or enable Watch in the **Linked items** menu). The server polls every two minutes (checking GitHub's change watermark first to stay within rate limits) and wakes the agent if:

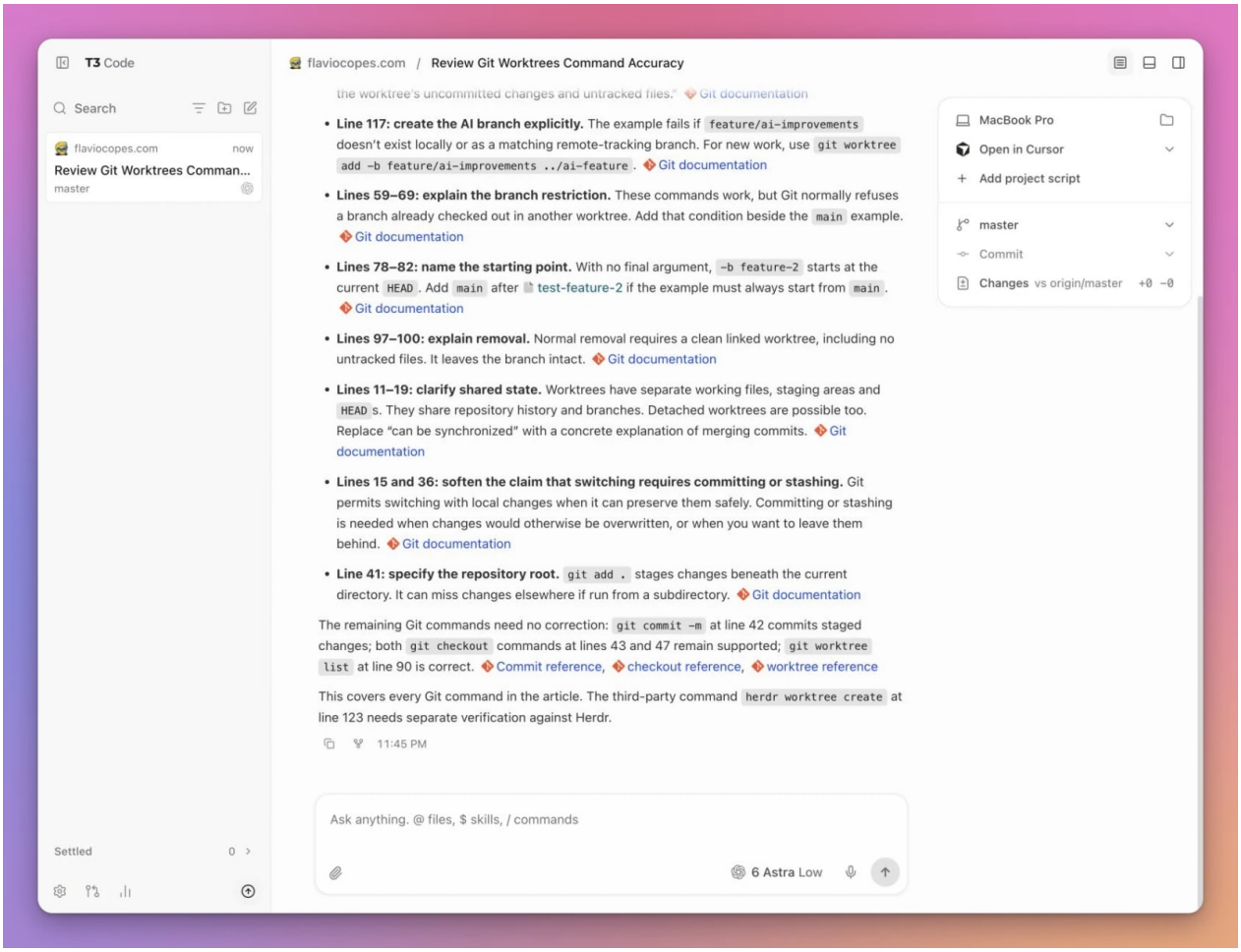
- a CI check fails, waits for maintainer approval, or passes
- someone else leaves a review or comment
- the branch develops a merge conflict or gets dropped from a GitHub merge queue

While checks are still running, the watched thread stays in **Working** and never auto-settles. Once every check finishes and the next decision belongs to a human, it moves back to your inbox (watch Theo explain PR Watch).

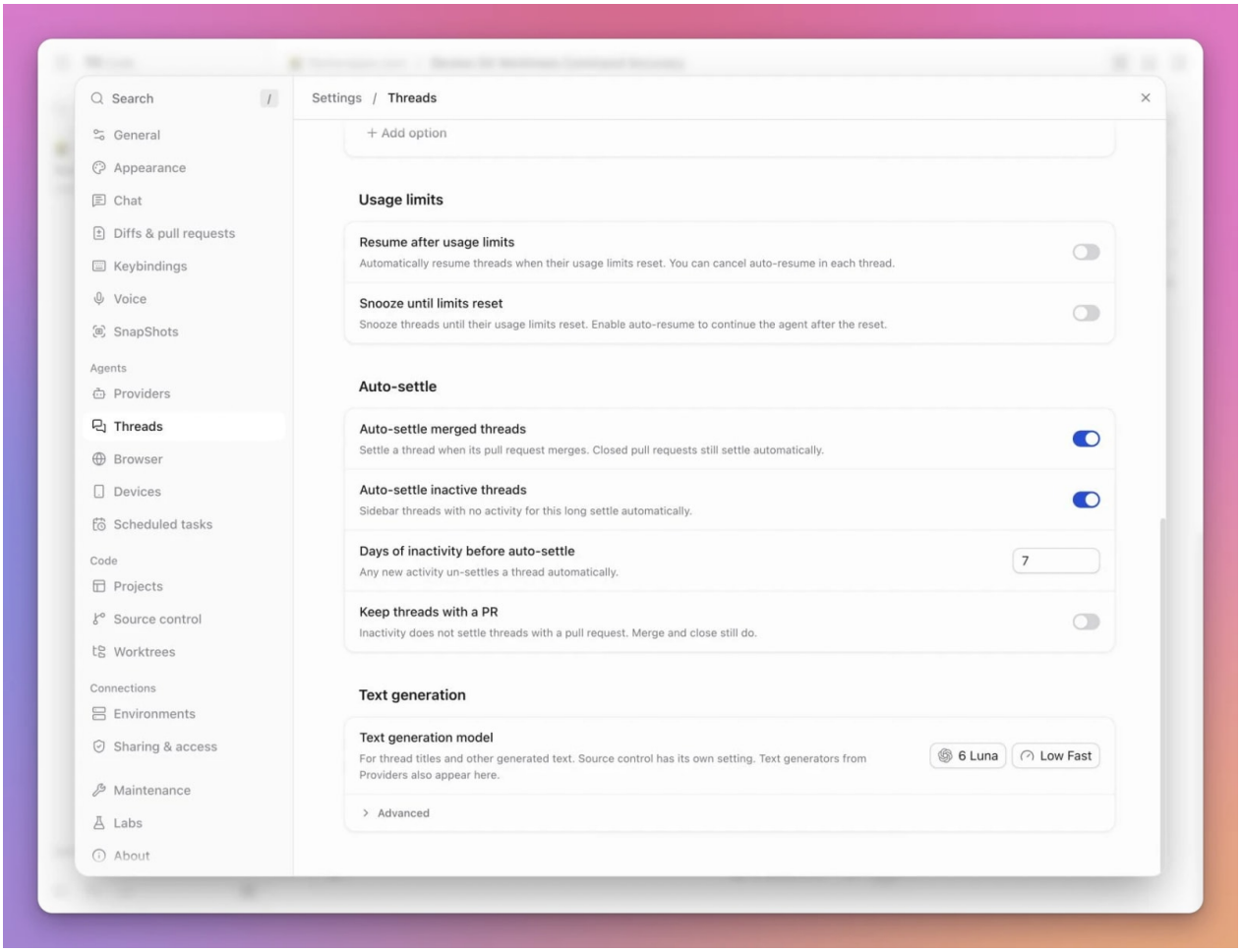
The thread sidebar and Usage dashboard

Theo treats the sidebar as an inbox (watch how he clears it), and the controls are built around that habit:

- **Pin** active threads and projects so they stay at the top of pickers.
- **Snooze** a thread until a set time; the agent keeps working, and an approval or question pops it back early.
- **Settle** finished work so it leaves your active inbox while staying searchable under **Settled** (unlike **Archive**, which freezes a thread and blocks scheduled tasks).



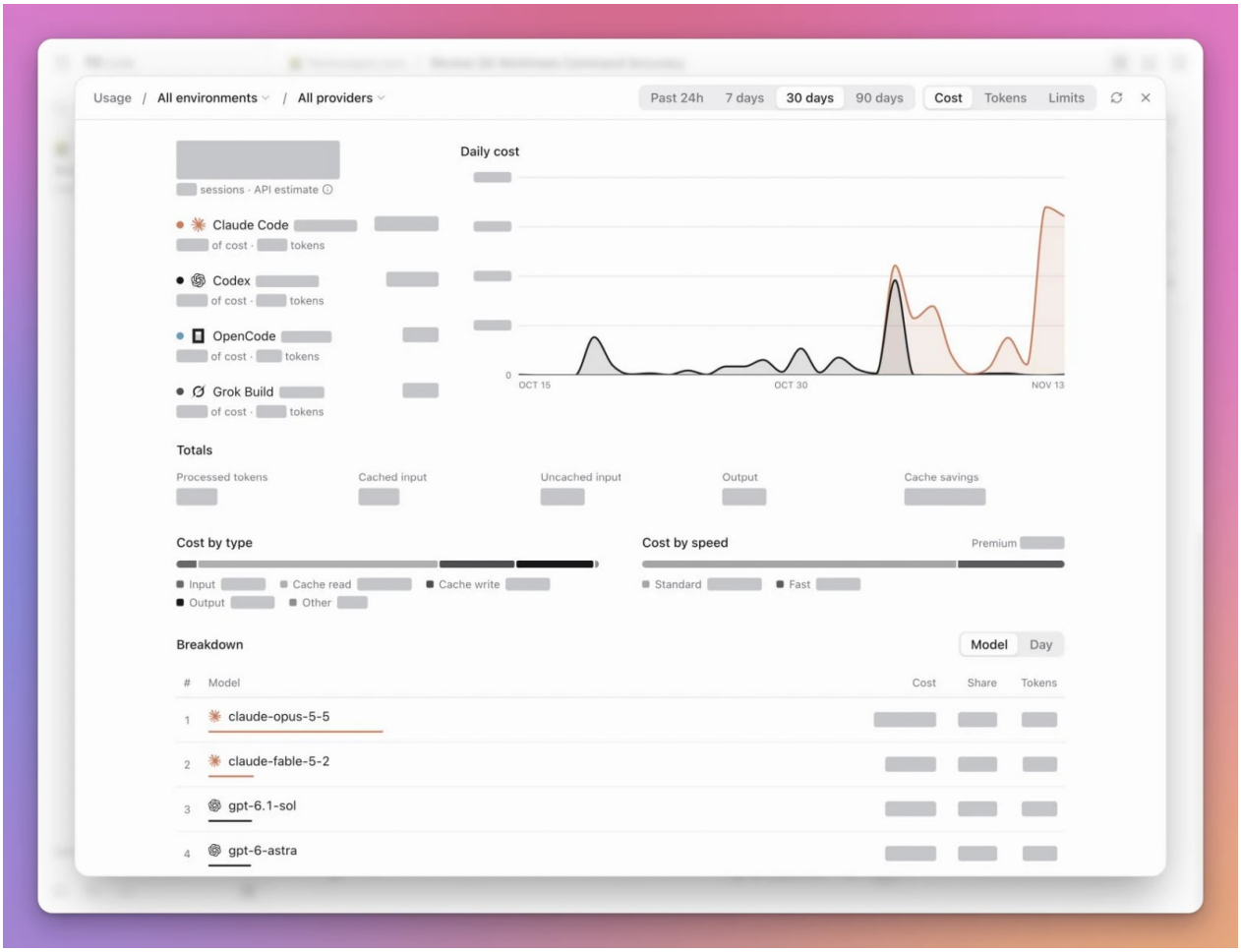
By default, threads auto-settle after **seven days of inactivity** or when their linked pull request merges. Adjust those rules under **Settings -> Threads** (and remember settings resolve in clear order: **project override -> environment setting -> `t3.json` -> built-in default**, shown by the layers icon at the top of Settings).



Cross-environment search in the command palette (`Cmd+K`) indexes thread titles, projects, branches, user prompts, and final replies, so you can find a decision from three weeks ago even if you forgot its title.

Usage and pooled subscription Limits

Press `Cmd+U` (`Ctrl+U`) to open **Usage**, combining session history across Codex, Claude Code, Grok, OpenCode, Antigravity, Pi, Muse Code, and Cursor across all connected environments:



A few features on this page are easy to miss:

- **92-day local archive:** T3 Code retains scanned usage for 92 days by default (longer than Claude Code's 30-day transcript cleanup); set `"usageHistoryRetentionDays"` in `~/.t3/userdata/settings.json` to keep up to `"unlimited"`.
- **Custom model prices and mapping:** click **Model prices** in the environment dropdown to price private or unpriced models, or map a preview model ID onto its release name.
- **Pooled `Usage -> Limits`:** combines all your Codex, Claude, Antigravity, Cursor, Grok, OpenCode Go, and Muse accounts across environments (or a connected **CLIProxyAPI hub**) into unified remaining-quota bars with reset countdowns, even-spend pace lines, and banked reset credits.
- **Keep quotas visible:** turn on **Usage limits** and **Context window** under **Settings -> Appearance -> Thread details** to see live bars right inside every thread, or add the **Subscription usage** widget to your iPhone or Android home screen.

Remember that API-equivalent cost estimates let you compare workloads across providers; check each provider's billing portal for your actual invoice.

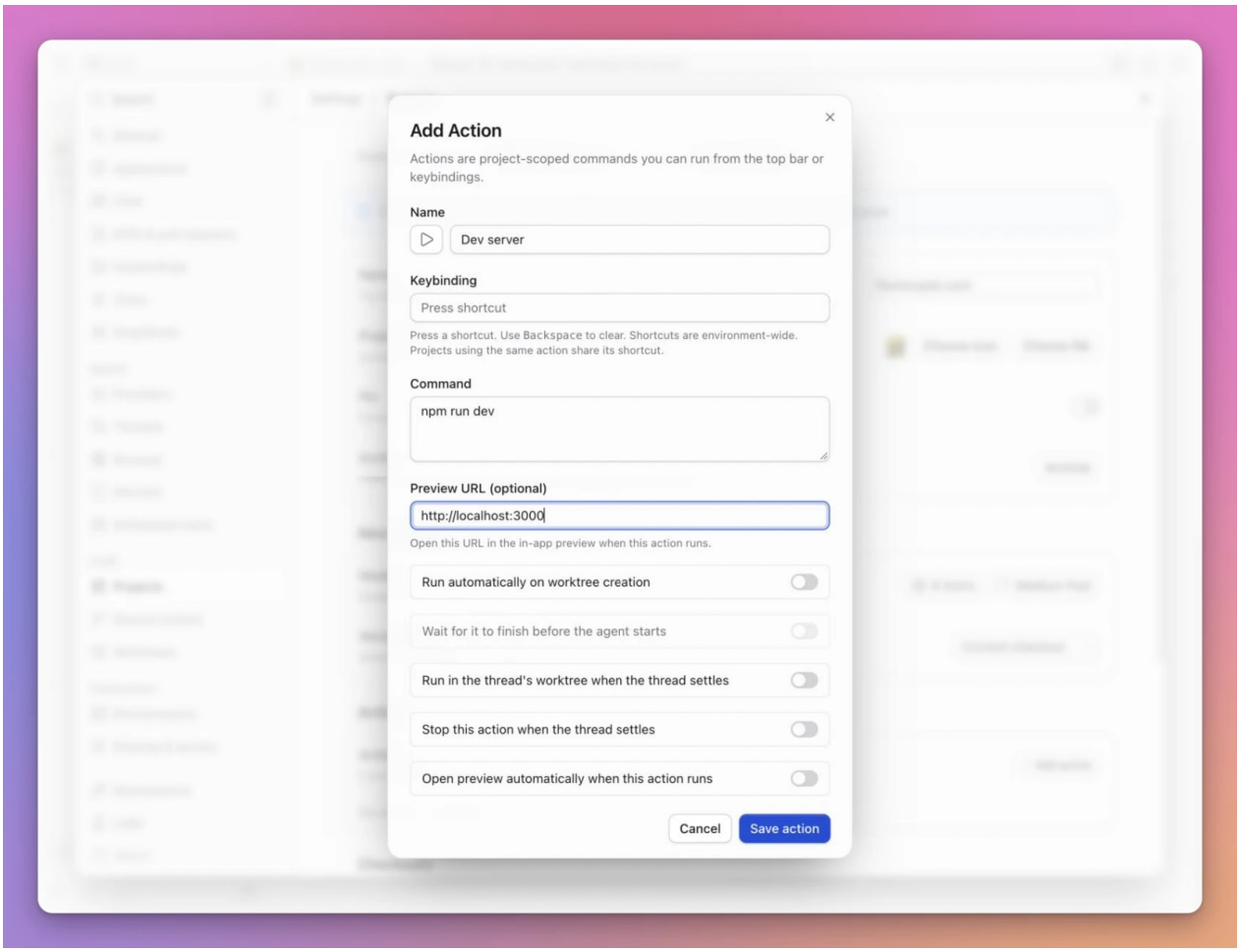
Projects, `t3.json` cookbook, and automation

Once a single thread feels natural, T3 Code gives you four layers of automation, from simple checked-in scripts up to durable TypeScript pipelines.



Project actions and `t3.json` cookbook

You can configure project commands in the UI with **Add Action** or check a `t3.json` file (schema: `https://t3.codes/schema/t3.json`) into the repository root so every machine gets the same actions (up to 50 per project):



Scripts receive `T3CODE_PROJECT_ROOT` and `T3CODE_WORKTREE_PATH`, and support lifecycle flags beyond just clicking Play:

- `runOnWorktreeCreate`: runs automatically when a new worktree spins up (set `"async": false` to make the agent wait until setup finishes).
- `stopOnSettle`: stops a dev server's terminal and child processes when the thread settles.
- `runOnSettle`: runs cleanup (like deleting build output) in a worktree when the thread settles.
- `previewUrl` and `autoOpenPreview`: opens the app straight in T3 Code's browser panel.

Here are three copy-paste `t3.json` recipes for common setups.

1. Single-checkout site (like `flaviocopes.com`):

```
{
  "$schema": "https://t3.codes/schema/t3.json",
  "defaultThreadEnvMode": "local",
  "scripts": [
    {
      "name": "Start Astro",
      "command": "npm run dev",
      "icon": "play",
      "previewUrl": "http://localhost:3000",
      "autoOpenPreview": true,
      "stopOnSettle": true
    },
```

```
    {
      "name": "Build site",
      "command": "npm run build",
      "icon": "build"
    },
    {
      "name": "Visuals check",
      "command": "npm run posts:visuals:check",
      "icon": "test"
    }
  ]
}
```

2. Parallel worktree web app with setup and settle cleanup:

```
{
  "$schema": "https://t3.codes/schema/t3.json",
  "defaultThreadEnvMode": "worktree",
  "worktreeSubmodules": "top-level",
  "scripts": [
    {
      "name": "Prepare worktree",
      "command": "npm ci && cp \"$T3CODE_PROJECT_ROOT/.env.local.example\" .env.local",
      "icon": "configure",
      "runOnWorktreeCreate": true,
      "async": false
    },
    {
      "name": "Dev server",
      "command": "npm run dev -- --port 0",
      "icon": "play",
      "stopOnSettle": true
    },
    {
      "name": "Clean build cache",
      "command": "rm -rf .next dist",
      "icon": "debug",
      "runOnSettle": true
    }
  ]
}
```

3. Multi-repo folder ignoring a heavy vendored repo:

```
{
  "$schema": "https://t3.codes/schema/t3.json",
  "defaultThreadEnvMode": "worktree",
  "ignoreRepositories": ["vendor/legacy-sdk"],
  "scripts": [
```

```
    {
      "name": "Run API + Web tests",
      "command": "(cd api && npm test) && (cd web && npm test)",
      "icon": "test"
    }
  ]
}
```

Never put secrets inside `t3.json` since it belongs in Git.

Scheduled tasks and webhook automations

Open **Settings -> Scheduled tasks** (on desktop, web, or mobile) to run prompts automatically:

- **Recurring or one-shot:** schedule a repeating cadence or pick **Once** for a single date and time (if the server is offline at that moment, it runs when the machine comes back).
- **Destinations:** start a **New thread each run**, continue one ongoing thread, or post into **one thread with a fresh session each run** so the timeline stays readable without context bloating.
- **Webhook automations (`On webhook`):** pair your server with T3 Connect to get a webhook URL, verify payloads with **Require signature** (e.g., GitHub's `x-hub-signature-256` with `sha256=` hex), filter on fields like `headers.x-github-event` or `body.action`, deduplicate by `{{headers.delivery-id}}`, or coalesce rapid bursts over a 10–3,600 second window. Turn on **Settings -> Sharing & access -> Hold webhooks while offline** to let T3 Connect buffer incoming webhooks for up to 24 hours when your machine sleeps.

Every templated value like `{{body.pull_request.html_url}}` reaches the agent wrapped in `<webhook-data-...>` untrusted-data tags with `$`, `@`, and leading `/` neutralized so incoming webhook text cannot silently trigger skills or slash commands.

> **Video walkthrough:** Watch the embedded video in the online edition at https://flaviocopes.com/t3-code/.

Durable Workflows (Labs)

When a job needs multiple stages, plain code, human approvals, and crash recovery, turn on **Settings -> Labs -> Workflows** and save TypeScript files into `~/.t3/workflows/` (the server hot-reloads changes automatically).

Workflows run on Effect (`T3`, `Effect`, `Schema`, and `Option` are injected globally). Here is a minimal release approval workflow saved as `~/.t3/workflows/release.ts`:

```
const { workflow, step, ask, stop, title } = T3

export default workflow(
  'release',
  { version: 1, payload: Schema.Struct({ name: Schema.String }) },
  Effect.fn(function* (payload) {
    yield* title(`Release ${payload.name}`)
    const notes = yield* step('notes', { returns: Schema.String }, Effect.succeed('3 fixes'))
    const answer = yield* ask('ship', {
      title: `Ship ${payload.name}? ${notes}`,
```

```
      choices: ['Ship', 'Hold'],
    })
    if (Option.isNone(answer) || answer.value.choice !== 'Ship') return yield* stop('held bac
    return { shipped: payload.name }
  }),
)
```

Start it with an idempotent `--key` (re-running the same key returns the existing run instead of duplicating work):

```
t3 workflow start release --input '{"name":"v1.4"}' --key v1.4
```

Workflows can run `agent` steps (even targeting projects on linked machines), trigger on `every`/`daily` schedules, expose secret webhook paths (`t3 workflow hooks`), or poll new issues and pull requests without configuring webhooks. Every finished step is persisted: failed code steps back off and retry (5, 20, and 80 minutes), while failed agent steps pause for you instead of looping blindly. If you want an agent to author workflows for you, grab the official `t3-workflows` skill.

Outside agents (`/mcp`), CLI scripting, Guest links, and Prompt links

T3 Code also opens up to tools and teammates outside its own window:

- **Control T3 Code over MCP (`/mcp`):** open the command palette and choose **Use T3 Code in another app (MCP)** to connect Claude Desktop, ChatGPT, Poke, or terminal Claude Code/Codex via OAuth. You pick the maximum permission cap (**Read only** through **Full access**) per client, and bots supporting MCP Events (`io.modelcontextprotocol/events`) can subscribe to turn completions, approvals, and workflow milestones instead of polling.
- **Script threads locally with `t3 thread`:** on the server host, drive threads directly from shell scripts with no extra login:

```
id=$(t3 thread start . "Find why the newsletter test is slow" --worktree master)
t3 thread send "$id" "Fix the bottleneck and run the test suite" --wait
t3 thread show "$id"
```

- **Guest links (Labs):** turn on **Settings -> Labs -> Guest links** (requires T3 Connect), right-click a thread, and pick **Share with guest**. One teammate can follow the thread streaming live in their browser for 24 hours with no account and submit questions or files; every guest message waits as a **Request** until you click **Send** or **Dismiss**. Guests never see diffs and cannot touch your files or settings.
- **Prompt links:** copy a shareable draft URL from the command palette (**Copy prompt link** or **Copy prompt link with repo and model**), or craft one by hand for a PR comment or tutorial:

```
https://app.t3.codes/new?pr=flaviocopes/flaviocopes.com/123&prompt=Fix%20the%20failing%20chec
```

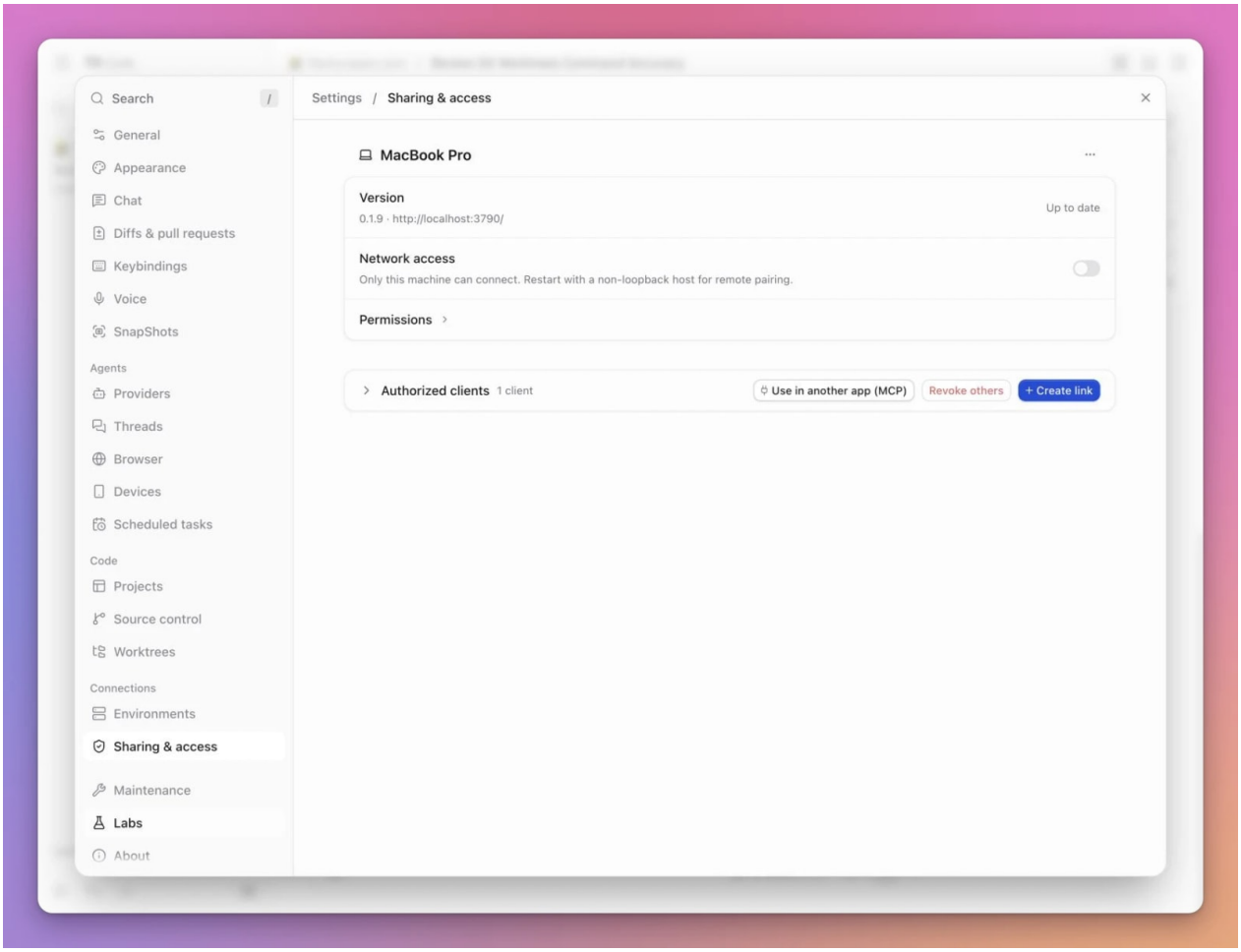
Prompt links always open as a draft using *your* permission and workspace defaults, never auto-run, and strip hidden zero-width Unicode characters with a warning if someone tried to sneak prompt injection into the URL.

Remote access, linked environments, and mobile

Remote access turns T3 Code from a local dashboard into a way to drive one or more dev machines from any laptop, browser, or phone.



Settings splits remote setup into **Settings -> Sharing & access** (how *this* machine is reached and who is authorized) and **Settings -> Environments** (remote servers and machine links):



Comparing the remote paths

Path	Best for	Setup on host	Tradeoff
LAN / Tailscale	Devices you own on a private network	`t3 up --lan` or `t3 serve --host "$(tailscale ip -4)"`	Zero relay; needs HTTPS (`--tailscale-serve`) to use `app.t3.codes`
T3 Connect	Zero-config reach from anywhere + mobile push	`t3 up` (choose T3 Connect)	Managed Cloudflare tunnel discovery; 3 free machines (10 Pro / 100 Premier with T3 Chat)
Desktop SSH	Linux dev boxes already reachable over SSH	Add SSH host in desktop **Settings -> Environments**	Desktop manages remote server + port forward; desktop-only

One of T3 Code's slickest networking details is **multi-route failover**. A single machine can have LAN, Tailscale, SSH, and T3 Connect routes at once. When you connect through T3 Connect, T3 Code automatically learns the host's LAN and Tailscale addresses; at home it switches straight to fast local Wi-Fi, and when you leave the house it falls back to T3 Connect automatically.

Headless setup, pairing, and device approval

On a headless Linux box or Mac mini, one interactive wizard configures both remote access and the background service:

```
t3 up
```

Or run a server manually and print a one-time pairing URL and QR code:

```
t3 serve --lan
t3 pair --tailscale
```

Pairing tokens travel in the URL hash so servers don't log them, and since `v0.1.11`, directly paired devices automatically renew their sessions while you use them (expiring after 30 days unused). For least-privilege clients, pass explicit scopes:

```
t3 pair --scope orchestration:read --scope relay:read
```

If you use T3 Connect, turn on **Require approval for new devices** under **Settings -> Sharing & access** (`v0.1.10`). Any new device signing into your T3 Connect account then displays a short code like `K7QM-2XFD` that you approve on an existing device or via `t3 auth connect-device approve K7QM-2XFD`. You can also keep a laptop as a thin remote-only controller by turning off **Settings -> Environments -> Local environment**.

Linked environments (Labs) and load balancing

Turn on **Settings -> Labs -> Linked environments** on two machines (like your MacBook and a beefy Linux VPS) and drag a connection between them on the visual map in **Settings -> Environments** (or ask an agent *"delegate this to my vps"*).

Agents on your laptop can then launch, message, wait on, and delegate threads onto the VPS with `delegate_task`, while you watch the remote thread live inside a right-panel tab without even switching environments. If a project exists on two or more connected machines, turn on **Settings -> Environments -> Load balancing -> Auto balance** so new drafts pick whichever machine has free CPU and memory (with per-machine **Prefer**, **Normal**, **Less often**, or **Manual only** weights).

Running as a background service (Linux, macOS, Windows)

Install T3 Code as a persistent service so threads survive closing terminals:

```
t3 service install
t3 service status
t3 update
t3 service restart
```

- **Linux:** installs a systemd user service with lingering enabled (`sudo loginctl enable-linger "$(id -un)"` if needed). Run `t3 service protect` to protect the T3 server (and optionally SSH and Tailscale) from the Linux OOM killer when heavy agent builds eat all RAM.
- **macOS:** installs a LaunchAgent (runs while logged in and awake). `v0.1.10` also added automatic sleep prevention so your computer stays awake while agents are actively working.

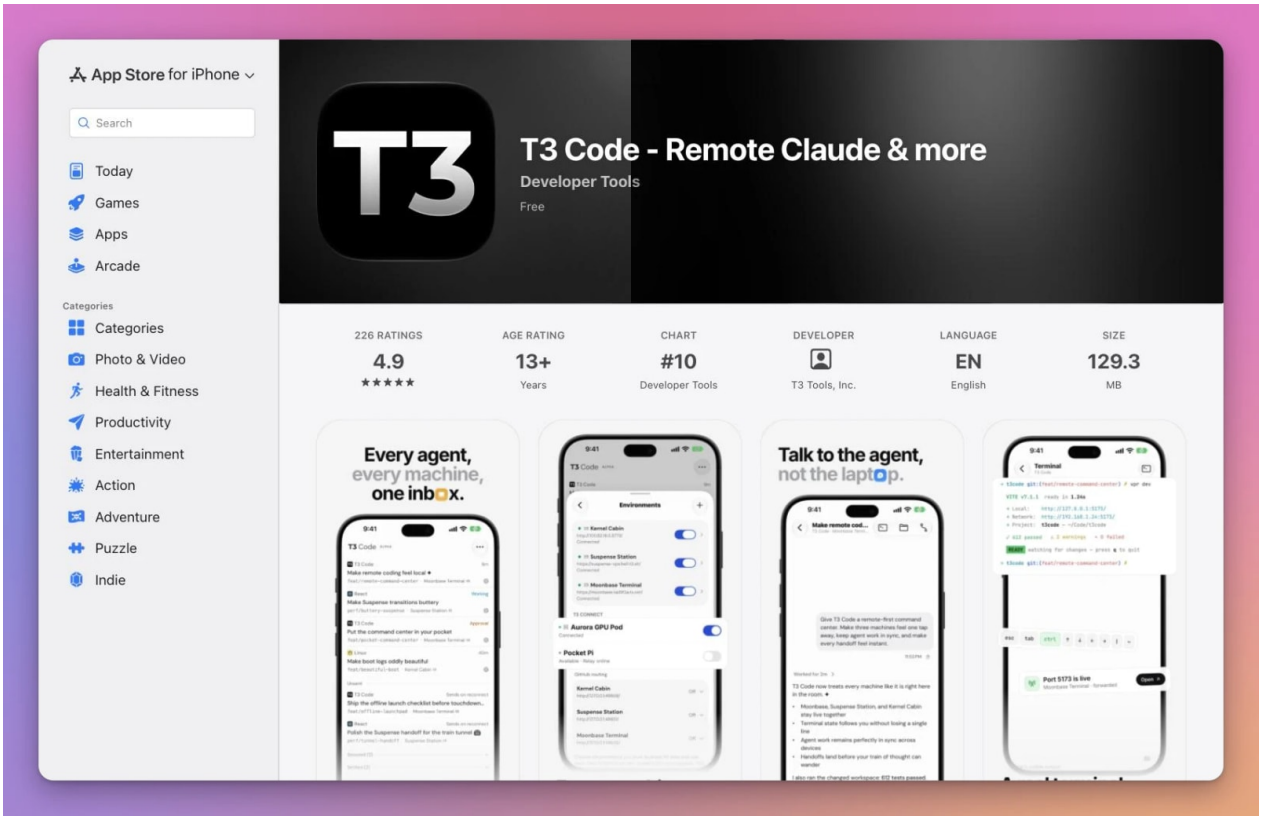
- **Windows (`v0.1.10`):** runs windowless via Task Scheduler (`t3code-<domain>-<user>`), restarting automatically within a minute if it exits, and supports remote updates.

Put persistent server env vars (`T3CODE_PORT`, `T3CODE_HOST`, Bitbucket tokens, `T3CODE_PORT_DISCOVERY=0`) in `~/.t3/service.env` (`chmod 600 ~/.t3/service.env`) so service updates preserve them. Over time, reclaim freed SQLite disk space from **Settings -> Maintenance** or by running `t3 doctor` when the server is stopped.

Always update servers, desktop apps, and mobile apps together, and avoid running `t3 update` mid-turn because restarting the server interrupts active provider sessions.

Mobile app, notifications, and themes

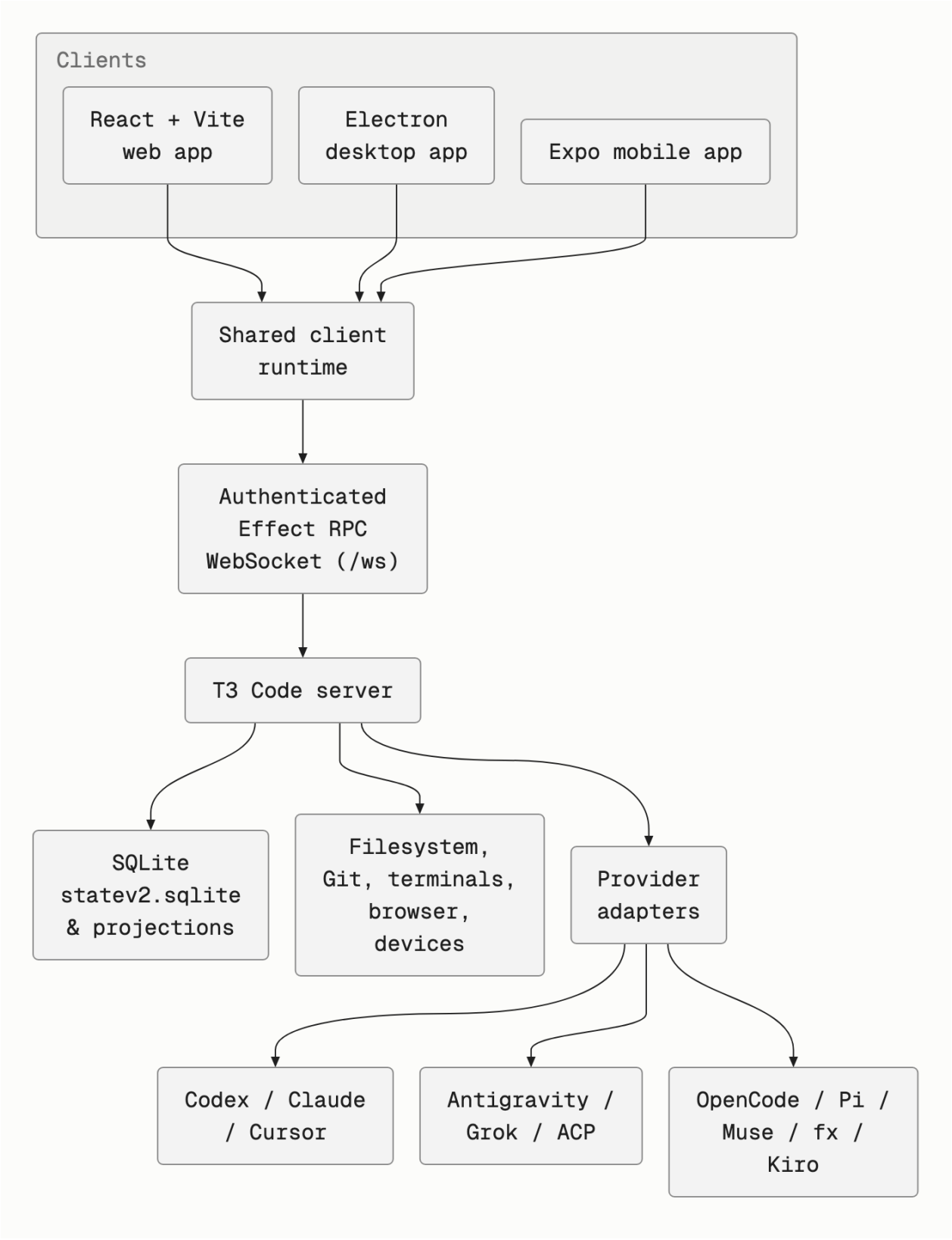
The iOS and Android apps share the same client runtime as desktop and web, adapted for touch:



- **Push alerts and Live Activities:** link the environment with T3 Connect and enable **Settings -> Notifications -> Device Notifications**. Follow running agents right on your lock screen via iOS Live Activities or Android 16+ Live Updates.
- **Offline resilience:** queue prompts and attachments on your phone while offline (they upload and send on reconnect), and read previously opened PRs, Review Guides, and diffs with no connection.
- **Themes, Custom CSS, and agent restyling (`v0.1.10`):** press `Cmd+Option+A` (`Ctrl+Alt+A`) to switch themes, import VS Code themes, or add **Custom CSS** (targeting stable `data-slot` attributes; external `url()` calls are blocked for safety). In a Full access thread on the local server, you can even ask an agent *"give T3 Code a solarized theme"* or *"use ~/Pictures/wallpaper.jpg as T3 Code's background"* and it publishes a live environment theme (`t3 theme set t3-chat` resets it anytime). T3 Code is also localized into 10 languages in `v0.1.11`.

Inside the architecture

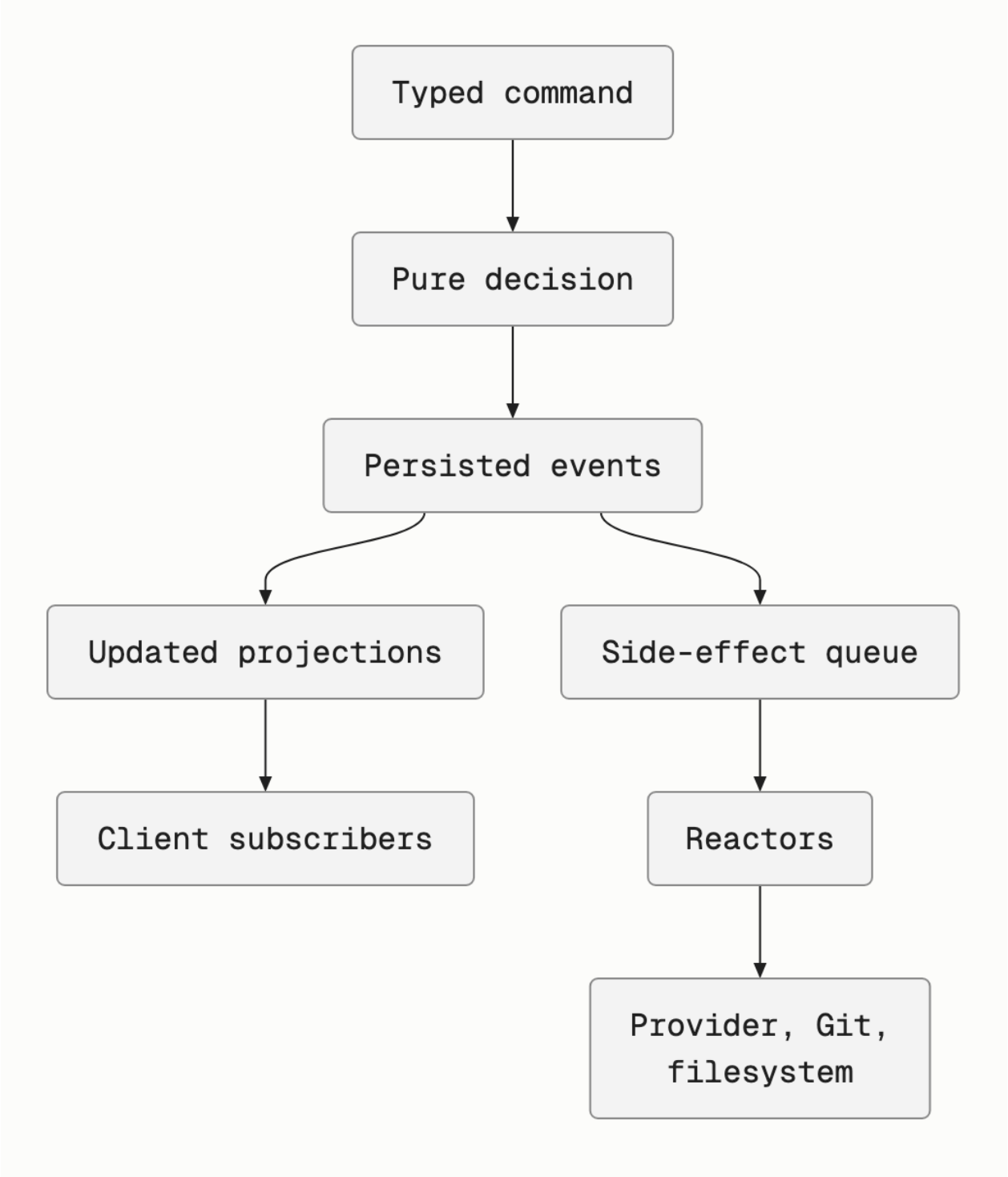
Understanding four design choices explains why T3 Code behaves the way it does.



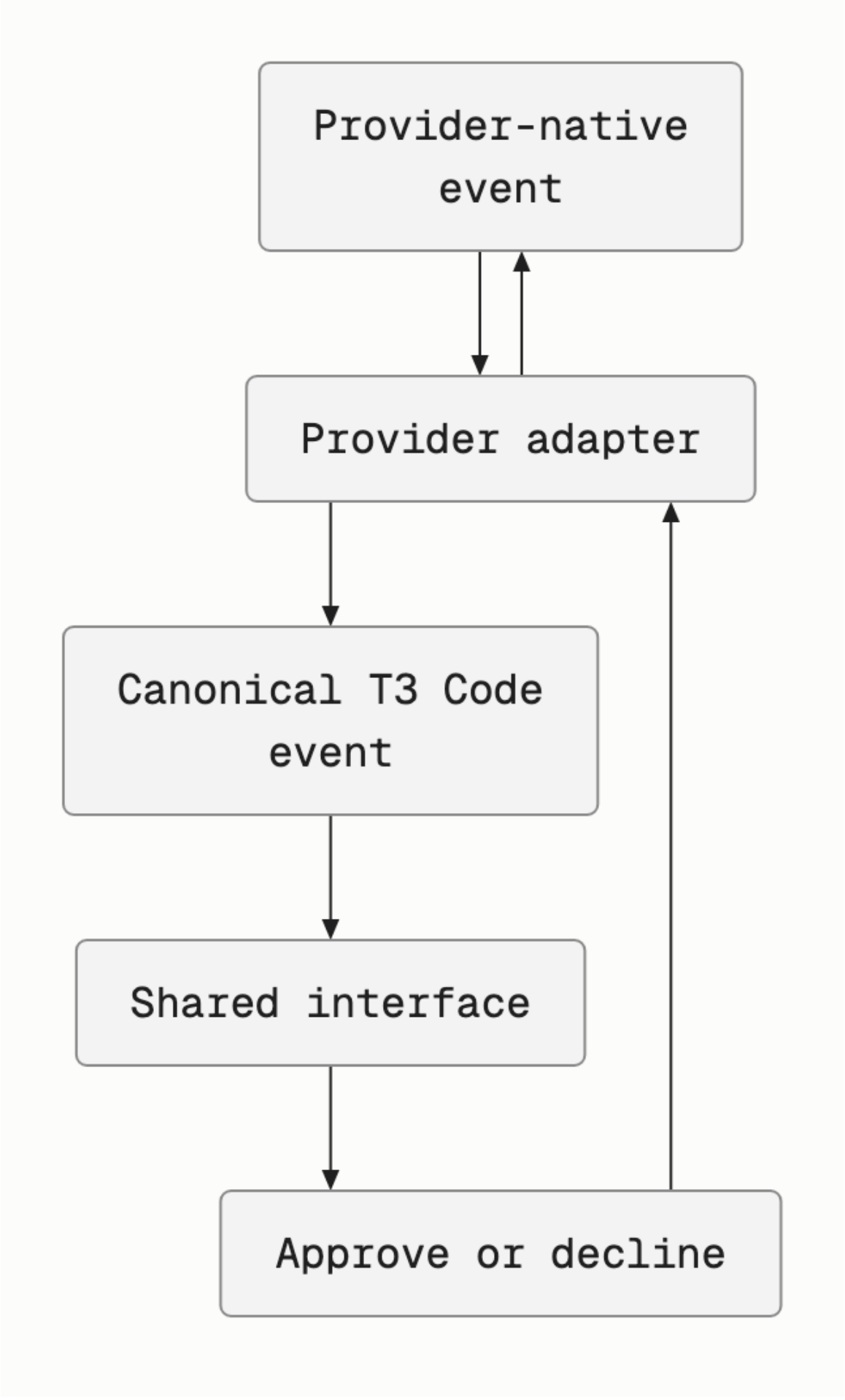
1. **The server is the execution boundary.** Clients never run providers, Git, terminals, or filesystem edits. Every client speaks typed **Effect RPC** over one authenticated `/ws` WebSocket (pairing token -> session -> 5-minute WebSocket ticket, with DPoP-bound relay ses-

sions on T3 Connect and per-operation scopes).

2. **Event-sourced SQLite (`~/.t3/userdata/statev2.sqlite`).** Every action enters a serialized worker as a typed command (`thread.create`), passes through a pure decision function, and writes events + UI projections in the same SQLite transaction using Node's built-in `node:sqlite`. Queue-backed **reactors** execute side effects (spawning providers, running Git, updating files) and emit outcome events back. That gives multi-client sync, crash recovery, and clean timelines without needing an external database.



3. **Adapters preserve provider edges.** Instead of scraping terminal ANSI output, each driver translates structured native events (tool calls, approvals, plans, usage, subagents) into canonical T3 events and sends decisions back down.


Provider-native event
Provider adapter
Canonical T3 Code event
Shared interface
Approve or decline

4. **Hidden Git refs for checkpoints.** Rather than copying your repo into SQLite after every turn, T3 Code writes Git trees under hidden refs and computes diffs directly from Git objects.

What stays local vs. what leaves the machine

- **Stays on your T3 Code server:** project files, Git worktrees, shell commands, terminals, `statev2.sqlite`, attachments, provider binaries, saved secrets, checkpoints, and logs.
- **Sent to your selected AI provider:** prompts, file context the provider reads, tool outputs, conversation history, and prompt attachments.
- **Crosses your chosen remote route:** streamed thread events, approvals, requested diffs/files, terminal I/O, and browser/device streams.

Managed policy for teams and admins

Organizations deploying T3 Code on managed laptops can drop a JSON policy file (including via Windows Group Policy) to lock settings, cap the broadest allowed permission mode, restrict allowed providers, or disable browser/preview tunnels across every client connected to that machine. Validate policies with `t3 policy check policy.json` and inspect active rules with `t3 policy show`.

Build from source

The repo uses Vite+ (`vp`) across its TypeScript monorepo:

```
curl -fsSL https://vite.plus | bash
git clone https://github.com/pingdotgg/t3code.git
cd t3code
vp i
```

Start with the internal architecture overview before making changes. Maintainers prioritize small bug, reliability, and performance fixes; open an Ideas discussion before building a large feature PR.

My real setup and five playbooks

On my MacBook Pro (`v0.1.11`), T3 Code is connected to five active providers at once: **Codex** (`gpt-6.2-sol`, `gpt-6.2-astra`), **Claude** (Opus 5.5, Sonnet 5.5, Fable 5.2), **Cursor**, **Antigravity** (Gemini 4 Neon alongside its shared models), and **GLM Agent** (`glm-5.3`) installed from the ACP Registry.

My server preferences are set to `newWorktreesStartFromOrigin: true`, `backgroundActivity: balanced`, `enableProviderUpdateChecks: true`, and `sourceControlWritingStyle: repo_conventions` with `followChangeRequestTemplates: true` so commits and PRs match each repository's rules instead of generic boilerplate.

For `flaviocopes.com`, this repo forbids Git worktrees, so threads run in the **current checkout** with one writer at a time while other threads run read-only reviews or comparisons across providers (the screenshots throughout this guide come from those threads). Long-lived rules still live in `AGENTS.md`, `CLAUDE.md`, and shared skills under `~/.agents/skills/`, the same contract I use in my fstack software factory and my review workflow for T3 Code threads.

Here are five concrete playbooks you can adopt immediately:

1. Multi-provider second opinion on a risky diff

Start a thread with your primary implementation agent (say Claude Opus 5.5 or Codex). Once the diff is ready, ask the thread to run `delegate_task` with two independent reviewers on different providers (for instance Codex + Antigravity, or Claude + GLM Agent), each checking the uncommitted diff for edge cases and regressions. Compare both reports side by side in the parent thread before committing.

2. Issue to merged PR without leaving T3 Code

Open **Issues**, filter to a saved view (like your Linear or GitHub triage queue), and start a linked worktree thread. After the agent finishes, check the **Commits** and **Changes** diffs, verify behavior in the in-app browser (or **Device** simulator), push and open the PR, generate a **Review Guide** if the diff is large, and tell the agent to **watch the pull request**. T3 Code keeps the thread in **Working** while CI runs, wakes the agent if a check fails or a reviewer comments, and auto-settles the thread plus cleans up the worktree after merge.

3. Zero-friction UI bug reports with SnapShots

Enable **Settings -> SnapShots** with **Include app text** turned on. Whenever a layout bug or broken dialog pops up in any desktop app or browser window, tap **both Shift keys**. T3 Code captures both the window image and its accessibility tree straight into your draft so you can type two words (*"fix this overlap"*) and hit Enter.

4. Stop burning coding-agent turns on titles and commit messages

Open **Settings -> Providers -> Text generators -> Add generator**, add a fast API key (OpenRouter, OpenAI, Anthropic) or local Ollama (`http://localhost:11434/v1`), and assign it to thread titles, commit messages, PR descriptions, and voice cleanup. Every new thread names itself in a second without spinning up a full Codex or Claude session.

5. Remote Linux server + laptop browser tunnel + phone approvals

Run `t3 up` and `t3 service protect` on an always-on Linux box, link it through T3 Connect (with **Require approval for new devices** turned on), and connect from your laptop and phone. Heavy builds and worktrees run on Linux; when an agent starts a dev server there, it opens straight in your laptop's in-app browser (or `Cmd-click` to port-forward it to your local `localhost`), and iOS Live Activities keep you posted away from the desk. Just keep one rule on mobile: use the phone to monitor, answer clear questions, and block risky commands, and save deep diff reviews for a real screen (more on this in running coding agents from your phone).

T3 Code compared with the alternatives

Tool	Built around	Agents it runs	Where code runs	Phone client	Pick it when
Provider CLI	One terminal session	Its own	Your machine	No	One quick task, or you need a provider's brand-new feature today

Tool	Built around	Agents it runs	Where code runs	Phone client	Pick it when
IDE	Manual code editing	Mostly its own	Your machine	No	You write and debug most code yourself
cmux	Native Mac terminal & programmable browser	Any CLI	Your Mac	No	You want a native Mac terminal + scriptable browser workspace
Herdr	Persistent terminal panes	Any CLI	Any reachable host	Yes (SSH)	You want tmux-style persistent, scriptable terminal agents
Hosted agent platform	Cloud sandboxes	Platform's own	Cloud VMs	Varies	You want elastic background compute with no machine to maintain
T3 Code	Structured threads & environments	10 built-in + ACP Registry	Machines you control	Yes (iOS & Android)	You supervise several providers across desktop, web, and phone

A few quick pairings worth calling out:

- **With a direct CLI:** keep both. The raw CLI has zero translation layer when a provider ships a feature an hour ago; T3 Code wins once you have multiple threads, multiple providers, or remote devices.
- **With an IDE:** T3 Code centers on supervising agents; VS Code, Cursor, or Zed still win for deep manual refactoring and interactive debugging. Open the worktree in your editor whenever you need to take over by hand (see IDE, CLI, or vibe coding tool?).
- **With cmux or Herdr:** cmux and Herdr operate at the terminal layer, so any CLI works immediately. T3 Code operates higher in the stack with typed events, checkpoints, Review Guides, mobile simulators, and cross-provider delegation (more comparisons in Herdr alternatives and Conductor alternatives).

Where T3 Code fits well (and where it doesn't)

Great fit: you juggle two or more providers, use existing subscriptions, run parallel worktrees, want browser/simulator verification and PR tracking in one place, or supervise an always-on dev machine from a laptop and phone.

Poor fit: you run one five-minute CLI task at a time, spend all day editing by hand in an IDE, need a multi-tenant hosted cloud platform with org-wide cloud compute, or cannot keep a host machine online and isolated from shared local databases.

Recommended first-day setup order

1. Install the desktop app (`brew install --cask t3-code`) and verify the `t3` command works.
2. Enable **one** provider you already use and confirm one read-only thread works.
3. Set your default new-thread permission mode to **Supervised** or **Auto** until you trust your setup.
4. Add a small test repo, run one small edit, inspect the per-turn diff, and test **Edit from here**.
5. Add a `t3.json` with your dev server (plus `previewUrl` and `stopOnSettle: true`) and test command.
6. Wire up a cheap **Text generator** for titles and commits.
7. Connect GitHub/GitLab/Linear under **Settings -> Source control & issues** and try one test PR.
8. Add **Tailscale** or **T3 Connect** last, once local behavior is rock solid, and pair your phone.

Pre-flight safety checklist

Before letting a thread touch an important repo:

- Right environment (local Mac vs. remote server) and right project folder?
- Current checkout (one writer only) or isolated worktree?
- Appropriate permission mode (`Supervised` or `Auto` for valuable code, `Full access` only in disposable workspaces)?
- Shared ports, local databases, or production credentials isolated from parallel worktrees?
- Clean Git commit baseline so Git stays your real recovery net?

Quick-reference cheat sheets

Keyboard shortcuts and composer syntax

(`mod` is `Cmd` on macOS, `Ctrl` on Windows and Linux. Customize everything in **Settings -> Keybindings** or `~/.t3/userdata/keybindings.json`.)

Action	Shortcut / syntax
Command palette / cross-environment search	`mod+K`
Jump to threads 1–9 (desktop) or models in picker	`mod+1` … `mod+9`
Send in background & open a fresh thread	`mod+Enter` (new thread) / `mod+Alt+Enter` (active thread)
Invert Queue / Steer for one message	`mod+Enter` while a turn runs

Action	Shortcut / syntax
Promote oldest queued message to steer	`mod+Shift+Enter`
Edit most recently queued message	`Option+Up` (`Alt+Up`) at composer start
Stash or restore draft + attachments	`mod+S`
Model / Effort / Permissions / Workspace / Branch / Host	`mod+Shift+M` / `E` / `A` / `X` / `G` / `H`
Reuse previous worktree directly	`mod+Shift+L`
Start or finish voice dictation (`Esc` discards)	`mod+Shift+D`
Global SnapShot (window + accessibility tree)	Both `Shift` keys together (default)
Search active thread / diff / desktop preview page	`mod+F`
Open Usage page (`C` Cost, `T` Tokens, `L` Limits)	`mod+U`
Theme picker / cycle Light, Dark, System	`mod+Option+A` / `mod+Option+Shift+A`
Copy PR URL or thread chip / copy `#PR` number	`mod+Shift+C` / `mod+Shift+K`
Undo sidebar action (settle, snooze, archive)	`mod+Z` (within 5s)
Reopen last closed panel tab	`mod+Shift+T`
Commands / Skills / Thread or model mention / PR chip	`/` · `$` · `@` · `#`
Side chat / Fresh session / Compact / Goal / Limits	`/side` or `/btw` · `/clear` · `/compact` · `/goal` · `/usage-limits`

`t3` CLI reference

Task	Command
Start local server and open web UI	`t3`
Open folder in running desktop app	`t3 app [path]`
Run headless server (optional LAN/Tailscale)	`t3 serve [--lan | --tailscale-serve]`
Interactive remote + background service setup	`t3 up [--connect | --lan]`
Check server and remote exposure status	`t3 status [--json]`
Print one-time pairing link & QR code	`t3 pair [--tailscale] [--scope <scope>]`
Manage background service	`t3 service install | status | restart | uninstall`
Protect Linux server/SSH/Tailscale from OOM killer	`t3 service protect`
Manage sessions & T3 Connect device approvals	`t3 auth connect-device list | approve <code> | revoke <code>`
Update `t3` and background service	`t3 update [--channel nightly]`
Script threads from the host shell	`t3 thread start | send --wait | show | list`
Add a project with extra tracked repo folders	`t3 project add <path> --track <folder>`
Link environments from the terminal	`t3 environment link <addr> --pairing-code <code> --access <mode>`
Run & steer durable Workflows	`t3 workflow start | list | show | resume | signal | hooks`
Fix Linux headless Chrome sandbox & fonts	`sudo t3 browser setup`
Manage environment themes	`t3 theme set <id> | show | clear`
Reclaim SQLite disk space offline	`t3 doctor [--dry-run | --fix]`
Validate or inspect enterprise managed policies	`t3 policy check policy.json` · `t3 policy show`

Task	Command
Launch self-triage agent when T3 won't start	`npx t3 triage`

Troubleshooting and common gotchas

Problem	Why it happens & how to fix it
`app.t3.codes` refuses to connect to a LAN/Tailscale IP	Browsers block plain HTTP/WS from an HTTPS page (mixed content). Use **T3 Connect**, `--tailscale-serve` (HTTPS), or the desktop/mobile app directly.
Upgraded to V2 (`0.1.0+`) and an old client won't connect	V2 servers and V1 clients reject each other. Update every server, desktop app, and mobile app (2.0+) together. V2 copies `state.sqlite` to `statev2.sqlite` one way.
Browser tabs or HTML renders fail on Ubuntu 23.10+ / Linux	Ubuntu restricts Chrome's sandbox or lacks fonts/libs. Run `sudo t3 browser setup` (don't run the T3 server as `root`).
1Password secrets fail on a headless Linux box	Desktop `op` sign-in needs local biometric approval. Configure a 1Password service account or Connect server on the headless host.
Provider **Auto** mode still prompts on every step	OpenCode has no Auto classifier, and some personal Google accounts lack Antigravity Auto, so T3 falls back to **Supervised** instead of faking it.
Parallel worktree dev servers fight over port 3000 or one DB	Worktrees isolate Git files, not ports or databases. Assign dynamic ports (`--port 0`) or isolated DB names in your `runOnWorktreeCreate` script.
`t3 update` interrupted an active agent	Restarting the server terminates running provider processes. Wait until active turns finish before updating.
T3 Code won't start at all	Run `npx t3 triage` (or search **Triage an issue** in the command palette when open) to launch Claude Code or Codex against your local logs and source version (watch the triage demo).

Release history

The `0.1` Orchestrator V2 line started on October 15, 2026. Here are the milestone releases reflected in this guide, newest first:

Release	Date	Headline additions
0.1.11	November 20, 2026	Browser tunnels & one-click port forwarding, Issues page (Linear/GitHub/GitLab/Azure/Forgejo), Antigravity Auto & Edit from here, Cursor edit earlier messages, 10 app languages, auto-renewing paired sessions, side-chat references
0.1.10	November 17, 2026	Custom themes, CSS & background images, Windows background service + remote updates, keep-awake while agents work, T3 Connect device approval, native resume on stopped threads, reply to responses
0.1.9	November 11, 2026	T3 Connect sign-in for agents, session-wide MCP app approvals for Claude, GitLab MR revert, OpenCode goals, deep PR links, screen-reader turn announcements, send while attachments upload
0.1.8	November 9, 2026	`/clear`, `request_secret`, inline MCP Apps, voice dictation, auto-settle after 7 days, `t3` command installer, `/mcp` server for outside agents, Claude Fable 5.2
0.1.7	November 4, 2026	Kiro (beta), managed policies (incl. Windows Group Policy), one-time scheduled tasks, child approvals in parent thread, copyable thread links, Usage provider/model filters
0.1.6	October 31, 2026	Side chats, pinned/archived projects, Commits scope in Diff pane, 1Password secret references
0.1.5	October 29, 2026	Claude and GPT models through Antigravity with thinking controls

Release	Date	Headline additions
0.1.0	October 15, 2026	Orchestrator V2: `delegate_task`, thread forks, server-side queue, scheduled tasks, auto-resume after quota resets

Notable highlights in `0.1.11` and `0.1.10`

- **0.1.11** (643 changes from 138 contributors, announcement thread): turned remote dev work into a local-feeling loop with **browser tunnels and one-click localhost port forwarding**, launched the unified **Issues page** across five trackers, upgraded **Antigravity** with Auto mode and Edit from here, added **10 full UI translations**, let you drag file pills straight into Finder/Explorer, and made directly paired devices renew automatically instead of re-pairing every 30 days.
- **0.1.10** (1,924 changes from 360 contributors): brought **Windows background services**, **keep-awake while agents work**, **custom themes/CSS/wallpapers**, **T3 Connect new-device approval**, **direct replies to assistant responses**, and **native resume** for stopped Claude, Codex, OpenCode, and Cursor threads.

Go deeper on the foundations

T3 Code gets even better once you know the pieces underneath it:

- The complete guide to Claude Code and How to use OpenAI Codex for provider-specific skills, hooks, and CLI flags
- Which AI model is best for coding for choosing models across providers
- How to write an Agent Skill and `AGENTS.md` for keeping project rules portable across every harness
- What is MCP for understanding both MCP Apps inside threads and controlling T3 Code over `/mcp`
- Tailscale and the free Git course for private remote networking, worktrees, and clean branch history