LocalAgent: A Local-First Agent Runtime That Actually Cares About Safety

February 21, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** LocalAgent (v0.1.1, released Feb 21 2026) is a Rust-based agent CLI that connects to LM Studio, Ollama, or llama.cpp and runs coding/automation tasks with deny-by-default trust controls. Every tool call goes through a policy engine — shell and file writes are disabled unless you explicitly enable them, and even then they can require per-call approval with TTLs. Every run produces a deterministic replay log you can verify later. It's very early (16 stars, single developer), but the design philosophy is exactly right: safety as architecture, not afterthought. Install with `cargo install --path .` or grab a prebuilt binary from GitHub.

📚 **Related:** The Agentic Web · OpenClaw Setup Guide · OpenClaw Security Guide · The 5 Levels of AI Coding · Planning Tool

The agent landscape has a safety problem. OpenClaw has 42,000+ exposed instances with full system access. Cline just had a package injection incident. The r/LocalLLaMA post about the Midwest developer whose local agent fabricated 40.8% of claimed tasks. These aren't edge cases. They're the predictable result of agent frameworks that ship with maximum capability and zero safety defaults.

LocalAgent, released today (v0.1.1, Feb 21 2026) by CalvinSturm, inverts this. It's a Rust CLI for running AI agents locally with deny-by-default permissions, explicit approval workflows, and deterministic replay logs. It connects to LM Studio, Ollama, or llama.cpp — the same backends you're already running — and wraps them in a trust model that's built into the runtime, not bolted on after the fact.

It's very new. 16 GitHub stars. One developer. But the design philosophy is exactly what local AI agents need.

Why This Exists

From CalvinSturm's Hacker News post:

> "I originally tried wrapping trust controls around existing agent CLIs, but tool execution is too native in those products to reliably enforce policy externally."

This is the right diagnosis. When tool calling is deeply woven into an agent framework's core loop, you can't reliably gatekeep it from the outside. A wrapper that intercepts shell commands will miss the edge cases. A policy file that says "don't delete files" is only as good as the enforcement mechanism — and if that mechanism is a Python hook in a framework that wasn't designed for it, you have a suggestion, not a gate.

LocalAgent builds the gate into the loop. Every tool call passes through a policy engine before execution. The defaults are locked down. You unlock capabilities explicitly, per-run, with scoped permissions that expire.

Getting Started

Install

Three options:

```
# Build from source
git clone https://github.com/CalvinSturm/LocalAgent.git
cd LocalAgent
cargo build --release

# Or install globally
cargo install --path .

# Or grab a prebuilt binary from GitHub Releases
# (Linux, macOS, Windows archives available)
```

No Python. No pip. No venv. One binary.

First Run

```
localagent init                          # scaffolds .localagent/ directory
localagent doctor --provider ollama      # verify connectivity
```

`localagent init` creates a `.localagent/` directory with default config files: `policy.yaml`, `instructions.yaml`, `hooks.yaml`, `mcp_servers.json`, and directories for runs, sessions, and evals.

`localagent doctor` checks that your chosen backend is running and responding. Supported providers:

Provider	Default Endpoint	Notes
`lmstudio`	`http://localhost:1234/v1`	OpenAI-compatible API
`ollama`	`http://localhost:11434`	Ollama native API
`llamacpp`	`http://localhost:8080/v1`	Needs `--jinja` flag on llama-server

Chat and Run Modes

```
# Interactive TUI chat
localagent --provider ollama --model qwen3:8b chat --tui

# Single-task run
localagent --provider ollama --model qwen3:8b \
  --trust on --approval-mode interrupt \
  run --prompt "Refactor the error handling in src/main.rs"
```

Chat mode gives you a terminal UI with toggleable panes for tools, approvals, and logs. Run mode executes a single prompt and exits. Both operate under the same trust controls.

The Trust Model

This is what makes LocalAgent different. The trust system is layered and deny-by-default.

Default State: Everything Locked

Out of the box:

- `--trust off` — no trust layer active
- `--allow-shell false` — shell execution blocked
- `--allow-write false` — file writes blocked
- `--enable-write-tools false` — write tools aren't even exposed to the model

That last point is key. LocalAgent doesn't just block write calls — it hides them from the model's tool catalog entirely unless you opt in. The model can't call what it doesn't know exists.

Unlocking Capabilities

```
# Allow shell, require approval for each call
localagent --provider ollama --model qwen3:8b \
  --allow-shell --trust on --approval-mode interrupt \
  run --prompt "Run the test suite"

# Allow everything, auto-approve within this run
localagent --provider ollama --model qwen3:8b \
  --allow-shell --enable-write-tools --allow-write \
  --trust on --approval-mode auto --auto-approve-scope run \
  run --prompt "Fix the failing tests"
```

Three approval modes:

- **`interrupt`** (default): Pauses and asks you. The TUI shows pending requests; you approve with `Ctrl+A` or deny with `Ctrl+X`.
- **`auto`**: Approves within configured scope. Scoped to the current run by default — approvals don't bleed across runs.
- **`fail`**: Denies everything that requires approval. Strict mode for automated pipelines.

Policy Files

The scaffolded `policy.yaml` is deny-by-default:

```
version: 2
default: deny
rules:
  - tool: "list_dir"
    decision: allow
  - tool: "read_file"
    decision: allow
  - tool: "shell"
    decision: require_approval
    reason: "shell execution requires explicit approval"
  - tool: "write_file"
    decision: require_approval
    reason: "file writes require explicit approval"
```

Read-only operations are allowed. Everything else needs approval. You can customize with glob patterns (`mcp.playwright.*`), conditional rules that check argument values, and composable policy includes. The `localagent policy doctor` command validates your policy, and `localagent policy test` runs deterministic test cases against it.

Approval TTLs

Approvals aren't permanent:

```
localagent approve <id> --ttl-hours 24 --max-uses 10
```

An approval expires after 24 hours or 10 uses, whichever comes first. Expired approvals are pruned automatically. This prevents the "I approved shell access once and forgot" problem that plagues every other agent framework.

Replay and Verify

Every run produces a JSON artifact in `.localagent/runs/<run_id>.json` containing the full message transcript, every tool call with arguments and results, every policy decision with reasons, and configuration hashes.

```
# Replay a recorded run
localagent replay <run_id>

# Verify run integrity
localagent replay verify <run_id>
localagent replay verify <run_id> --strict
```

Verification checks whether the config hash, policy hash, hooks hash, and tool schemas match the recorded run. If anything changed since the run was recorded — someone edited the policy, updated a tool, changed the hooks — the verification flags it.

Why this matters: when an agent does something wrong (and they will), you need to know exactly what happened. Not a log file you hope captured the right events. A complete, verifiable record of every decision the agent made, every tool it called, and every result it received.

The `--repro on` flag goes further, capturing a full environment snapshot including OS, provider, model capabilities, and a reproducibility hash. For teams sharing agent workflows, this is how you answer "it worked on my machine" — you replay the exact run on theirs.

How It Compares

	LocalAgent	OpenClaw	CrewAI/LangGraph	Claude Code
Language	Rust	Python	Python	TypeScript
Install	Single binary	pip + deps	pip + deps	npm
Default trust	**Deny-by-default**	Allow-all	Allow-all	Approval prompts
Policy engine	Built-in, file-based	None	None	Permission modes
Replay/verify	**Full deterministic**	None	None	None
Providers	Local only	Local + cloud	Cloud-first	Anthropic only
Maturity	v0.1.1, 16 stars	Established	Established	Established
Community	1 developer	Large	Large	Large

The honesty column: LocalAgent is v0.1.1 with one contributor. OpenClaw, CrewAI, and Claude Code have thousands of users, active development teams, and battle-tested code paths. LocalAgent's safety design is better in principle — but it hasn't been tested by thousands of users trying to break it yet.

The other honesty: OpenClaw's security posture is genuinely dangerous at scale. CrewAI and LangGraph assume cloud-first architectures. Claude Code requires Anthropic's API. None of them produce verifiable replay logs. LocalAgent's architecture solves real problems that the established tools haven't addressed.

Honest Limitations

Very early. One developer, 33 commits, released today. Expect rough edges, breaking changes, and missing features. This is not production-ready software — it's a v0.1 with excellent design principles.

Rust means fewer contributors. Most AI developers work in Python. The Rust choice is correct for a safety-critical runtime (memory safety, no GIL, single binary), but it limits the contributor pool. If this project needs community momentum to survive, the language barrier matters.

Model compatibility varies. Tool calling requires models that support structured function calls. Not every model on Ollama handles this well. The eval framework helps test which models work, but you'll need to experiment.

No GUI. The TUI is functional but it's a terminal interface. If you want a web UI for agent management, this isn't it.

Single-provider per run. You connect to one backend at a time. No routing between providers or automatic model selection (yet).

Who Should Use This

Now: Developers who build with local agents and care about auditability. If you've been burned by an agent that deleted files it shouldn't have touched, or you need verifiable records of what your agent did, LocalAgent's replay system alone is worth the install.

Watch for: Anyone running agents in environments where safety matters — shared development machines, automated pipelines, distributed node setups. The trust model is designed for exactly these cases.

Skip for now if: You need a mature, well-documented tool with community support. Come back in six months and check the star count.

The Bigger Picture

The agentic web is being built by Coinbase, Stripe, Cloudflare, and OpenAI simultaneously. Agents are getting wallets, execution environments, and access to the open web. The infrastructure for autonomous software is shipping faster than the safety tooling to contain it.

LocalAgent is a small project with the right idea at the right time. Deny-by-default permissions, scoped approvals that expire, deterministic replay with cryptographic verification — this is what agent safety should look like. Not "are you sure?" prompts. Not YAML files that the runtime ignores. Policy enforcement in the execution loop, where it can't be bypassed.

Whether LocalAgent itself becomes the standard doesn't matter as much as whether its design patterns spread. Every agent framework should have these primitives. Most don't. One Rust developer in February 2026 shipped what billion-dollar companies haven't gotten around to building.

GitHub: CalvinSturm/LocalAgent
