Beads System Overview - Report

Viktor Ćurčić

Abstract

In this report, key questions were answered regarding the Beads framework. First, what problem Beads is solving and how it is meant to be used. Then, the high-level system architecture and main components of it were described. The question regarding tasks, issues, and how agent state are stored was answered. Also, how the agent is started, updated, and resumed from an existing state was also discussed. Finally, a little bit more was said regarding the broadness of the framework.

1 What Problem Beads Is Solving

Modern software development increasingly involves collaboration both between humans themselves and between humans and AI agents. While traditional issue trackers are designed around different web services and manual interaction, AI agents require something fundamentally different: persistent, structured, machine-readable state that evolves alongside the codebase.

The core problem Beads addresses is the lack of branch-aware coordination for agent-driven workflows. When an agent performs a task, it must remember what has already been done, what remains to be completed and how current work relates to other tasks. Stateless scripts or external SaaS issue trackers do not provide this kind of deeply integrated memory.

Beads solves this by embedding issue tracking directly into the Git workflow. Issues are not stored remotely on a web server; instead, they are stored locally, versioned with the repository and synchronized using Git itself. This means that task states branch with code, merge with features and remain available even when offline.

Essentially, Beads transforms Git repositories into structured coordination environments where both humans and agents can reason over shared, versioned task graphs.

2 High-Level System Architecture

Beads is organized as a layered system that separates interaction and coordination.

At the top layer is the command-line interface, invoked through the `bd` command. This interface is used by both developers and AI agents. It exposes operations for creating issues, modifying them, querying dependencies and resolving tasks.

The CLI does not communicate directly with storage. Instead, it talks to a long-running workspace daemon. The daemon acts as the central coordinator of system state. It maintains an open database connection, batches write operations for efficiency and monitors the repository for synchronization events.

Beneath the daemon lies the persistence layer. Issue data is stored in a structured database such as SQLite or Dolt for efficient querying and graph traversal. At the same time, this structured state is exported to a JSONL file that is tracked by Git. When changes are committed and merged, issue updates via the JSONL file propagate through standard version control mechanisms, where the daemon re-imports the changes into the database.

This separation ensures that interaction remains lightweight and state coordination remains consistent.

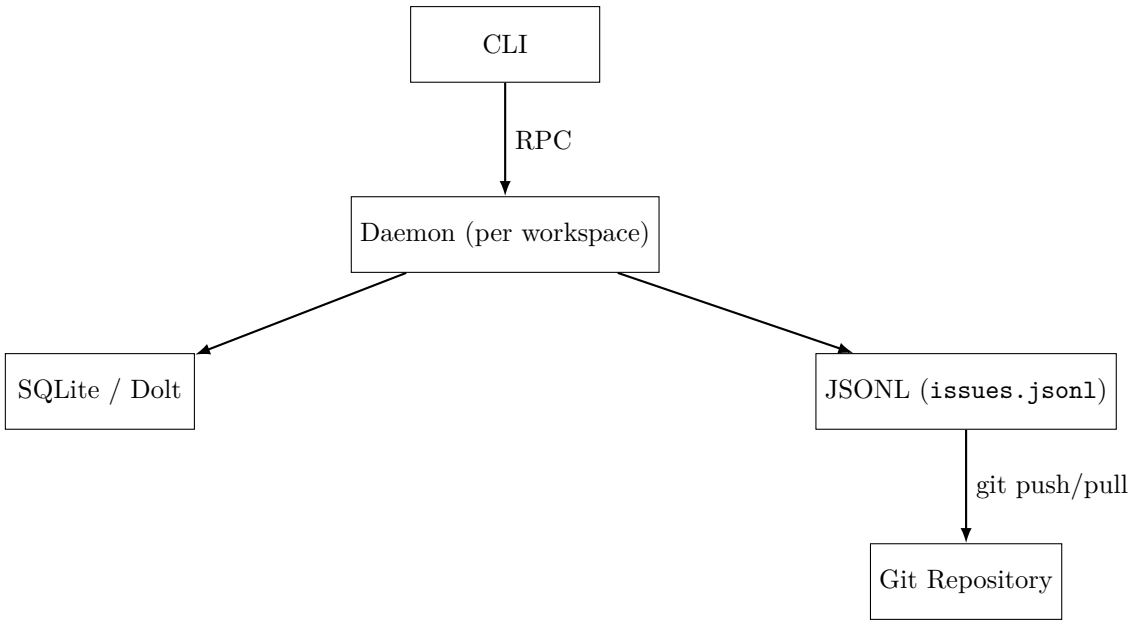

Figure 1: Beads workflow

3 How Issues and Agent States Are Stored

In Beads, work is represented as a graph of issues. Each issue corresponds to a durable unit of work. It contains a unique identifier, descriptive information, status metadata and references to other issues on which it depends.

Dependencies between issues form a directed graph. This structure allows the system to determine which tasks are ready for execution and which are blocked. To make these checks efficient, Beads maintains a cached set of blocked issues rather than recomputing the entire graph each time.

Issues are stored in two complementary forms. First, they reside in a local structured database, enabling fast queries and updates. Second, they are exported to a JSONL file tracked by Git. This file acts as the synchronization surface between repositories.

Deletion is handled using tombstones rather than physical removal. Instead of erasing an issue, the system marks it as deleted. This ensures that modifications across branches merge deterministically and that integrity is preserved.

Agent state is encoded directly in issue updates. When an agent performs work, its progress and status changes become structured updates to issue records. This means that agent memory is not transient. It becomes part of the versioned project state.

4 Agent Lifecycle

An agent interacting with Beads operates over the issue graph as its working memory.

When an agent starts, it queries the issue graph through the CLI to determine which tasks are ready for execution. Because the entire state is stored locally, this process does not require network access.

During execution, the agent performs work based on the selected issue. This may involve generating code, modifying files or performing analysis. Once progress is made, the agent updates the issue's status or metadata. These updates are passed through the CLI to the daemon, which writes them to the database and eventually exports them to the JSONL synchronization file.

If the agent process is interrupted, no critical information is lost. Since all relevant state is stored in structured issue records, the agent can resume by reloading the issue graph and continuing from the last recorded status.

Branching further enhances this lifecycle. Because issue state is versioned with Git, agents can operate independently on different branches. Each branch represents a consistent snapshot of code and task state. This makes parallel development and experimentation naturally supported.

5 Interesting Aspects and Framework Scope

Several design decisions make Beads particularly notable.

First, its Git-native synchronization model ensures that issue state evolves together with code. This eliminates the disconnect between repository history and task history that exists in traditional systems.

Second, the offline-first design allows full functionality without any central service. This makes the system resilient and suitable for distributed or privacy-sensitive environments.

Third, the use of tombstones and structured exports ensures deterministic merges. Rather than relying on manual reconciliation in a web interface, Beads uses Git's existing merge functionalities.

Finally, the framework is bigger than a conventional issue tracker. Because it models work as a dependency graph and exposes machine-readable interfaces, Beads functions as a workflow orchestration engine. It can coordinate not only developers but also autonomous agents operating over long-running tasks.

In this sense, Beads extends Git from a version control system into a coordination platform for hybrid human-agent development workflows.