

Agentic Software Development as Distributed Coordination: Content-Addressed Histories and Policy-Aware Descent

Anonymous

October 24, 2026

Abstract

We specify agentic development as distributed coordination, separating object dissemination, authoritative acceptance, and semantic validation. We prove convergence for idealized object exchange and conditional safety for certified atomic acceptance. A constraint-scope topology makes valid assignments a sheaf, identifying when local validation composes. Directed patch categories and sheaves of spaces provide a higher extension. Git supplies neither arbitrary program correctness nor Byzantine consensus.

1. Thesis, contribution, and scope

Agentic development manages concurrent, partially informed proposals under explicit acceptance rules. Agents may be stochastic, fallible, interrupted, or replaced: their outputs remain proposals rather than authoritative transitions.

Git and blockchains share hash-linked provenance. Object identity differs from agreement about accepted history. Bitcoin adds proof-of-work chain selection [6]; Git workflows delegate integration to maintainers and conventions [4]. Here we specify an authorized integration service. Trust in identity, authority, validators, and storage are distinct assumptions; trusted agents can generate incorrect code.

The contribution is a synthesis and a finite constraint-scope topology. Its convergence and invariant arguments specialize established results [5,7,8]; higher semantics uses existing patch and topos theory [9-11]. Content addressing and invariant confluence are not claimed as new.

2. Content identity and replicated knowledge

Definition 2.1 (idealized repository). Fix a universe O of finite, well-formed objects and a dependency relation $d \prec o$. Dependencies include commit parents and referenced trees and blobs. Assume the dependency graph is acyclic, each object has finitely many ancestors, and a digest H is injective on objects encountered in the execution. A repository view is a finite dependency-closed subset $K \subseteq O$:

$$o \in K \wedge d \prec o \implies d \in K.$$

Injectivity idealizes collision resistance. A digest identifies bytes, not authorship or behavior. Git objects are immutable, references mutable [1]. History rewriting creates different objects and moves

references; pruning can delete unreachable objects [3]. Immutability therefore implies neither permanent retention nor an immutable branch tip.

Proposition 2.2 (object-exchange convergence). Dependency-closed finite views form a join-semilattice under inclusion, with $K \vee L = K \cup L$. Suppose finitely many persistent replicas only add complete objects, retain them, and eventually disseminate every stored object to each other. They converge after generation stops; each fixed stored object is eventually present everywhere even without quiescence.

Proof. The union of two closed sets is closed. Union is associative, commutative, and idempotent, and is their least upper bound. For a fixed object, eventual delivery establishes the second conclusion. If generation stops with finitely many objects, the maximum of their finitely many delivery times gives convergence at each replica. $\square$

This grow-only model [7] idealizes completed transfers and retained ancestry. Shallow clones, pruning, and unavailable replicas require extra modeling. Branch references and working trees need not converge.

3. Ancestry, merges, and authority

Definition 3.1 (ancestry category). For a fixed commit DAG, let $a \leq b$ mean that $a = b$ or a is an ancestor of b . The associated thin category $\mathbf{C}$ has one arrow $a \rightarrow b$ exactly when $a \leq b$. This records reachability; it forgets distinct paths through the DAG. Commit timestamps are not a substitute for this order. Ancestry itself records only the dependencies represented by commits, rather than every communication event in an execution [5].

A merge commit with parents a, b is an upper bound in $\mathbf{C}$. It need not be a categorical pushout of a common-base span.

Proposition 3.2 (a merge need not be a join). A commit DAG can contain two incomparable upper bounds of a, b and no least upper bound.

Proof. Take a common base c , children a, b , and two distinct merge commits m_1, m_2 , both with parents a, b , and no further commits. Neither a nor b is an upper bound of the other, and m_1, m_2 are incomparable. There is no least upper bound. In a poset category a pushout of $a \leftarrow c \rightarrow b$ would be that least upper bound. $\square$

Git stores snapshots linked by provenance [1]. Workflows separately designate accepted references and who may advance them [4]. Hash linkage supplies neither authority nor Byzantine agreement; replicating an integration service requires an additional protocol.

4. Why independent success does not imply safe integration

Let S be semantic states, $I \subseteq S$ valid states, and $\mu : S \times S \rightarrow S$ a deterministic combination operation. Any base and configuration are fixed parameters. Consider branch states reachable from a common valid base by admitted local transitions and earlier combinations, without presupposing intermediate validity.

Definition 4.1 (merge closure on reachable states). Validity is merge closed if, for every such pair on which combination is defined,

$$x \in I \wedge y \in I \implies \mu(x, y) \in I.$$

Proposition 4.2 (boundary of eager acceptance). Suppose every local admitted transition preserves I . A protocol that independently accepts both branches and later retains their specified combination preserves I for all such executions if merge closure holds. If merge closure fails, that protocol has an unsafe execution.

Proof. Under closure, induction on local transitions and combinations preserves I . If closure fails, choose the witnessing branches x, y . Independent acceptance followed by their required combination reaches $\mu(x, y) \notin I$. $\square$

This specializes invariant-confluence reasoning [8]. Necessity concerns the stated availability, combination, and retention requirements. Rejection, reservations, rollback, or a different combination algebra can change the premises.

Example 4.3 (a clean merge exceeding a quota). Let a state be a finite set of jobs, each consuming six units, with invariant $6|s| \leq 10$. From $\emptyset$, one agent adds job A and another adds job B , in different files. Each branch satisfies the invariant. Their union costs twelve. A textual merge can be conflict free although the global specification fails. Therefore

$$\text{Checks}(x) \wedge \text{Checks}(y) \not\Rightarrow I(\text{merge}(x, y)).$$

Here checks may even establish the full branch invariant. The failure comes from composition, independently of language-model quality.

5. Certified authoritative acceptance

We specify a single authoritative integration service. Its availability and fault tolerance are explicit assumptions, not consequences of Git.

Definition 5.1 (manifest and certificate). An immutable manifest is $r = (g, h, \pi, \eta, b)$: a generation number, source commit digest, policy digest, environment-model digest, and predecessor-manifest digest. One atomic authoritative register R stores its digest. A candidate r' has generation $g + 1$ and predecessor $H(r)$. A certificate w is bound to the entire pair $(H(r), H(r'))$. A verifier $V(r, r', w)$ is **sound** when acceptance implies

$$I_{\pi', \eta'}(\text{Sem}(h')).$$

Here Sem interprets the source under the declared environment. Soundness is a hypothesis about the verifier. For a finite test suite, the justified invariant is initially that those tests passed; broader correctness needs a separate soundness argument. An environment digest describes a model or reproducible snapshot, not the continuously changing external world.

The service reads r , constructs the exact candidate r' , checks authorization under r , generation, predecessor, and V , then performs $\text{CAS}(R, H(r), H(r'))$. Failure requires recomputation against the new root. All writers obey this gate; objects and certificates remain available. An optional rule $h \leq h'$ retains accepted source ancestry. Git’s old-object argument to **update-ref** supplies a single-reference expected-value check [2], with validation and authorization added by the protocol.

Theorem 5.2 (conditional safety and acceptance order). Suppose the initial manifest is valid, digests identify retained immutable contents, the register is linearizable, every write follows the gate, and the verifier is sound. Then every accepted manifest satisfies its declared invariant.

Successful acceptances form one ordered predecessor chain, and two contenders based on the same manifest cannot both succeed.

Proof. Induct over successful register updates. A successful update names the candidate whose contents and evidence were checked. Soundness establishes its invariant. CAS ensures that its predecessor was the authoritative root at the update’s linearization point. Generation increases strictly, so an earlier manifest cannot reappear as the root; this excludes the ABA problem. Consequently, after one contender replaces the base, another CAS expecting that base must fail. The initial case is assumed. $\square$

The theorem assumes validator soundness. Fair scheduling, terminating validation, available storage, and an eventually stable base suffice for a fixed eligible proposal to receive a decision. Continual competition can cause starvation. A replicated R must independently establish the required register semantics.

A policy change may weaken its invariant. To preserve a fixed baseline I_* , require every accepted candidate to prove I_* as well, or forbid policy changes. Checking only the proposed policy does not establish preservation of an earlier policy.

6. A category of certified development

Let $\mathbf{P}$ be a small category whose objects are specified software states and whose arrows are directed development traces. Composition concatenates traces. This is distinct from the ancestry category, which forgets trace multiplicity.

Definition 6.1 (compositional evidence). A certificate semantics is a functor $E : \mathbf{P} \rightarrow \mathbf{Set}$. An element $p \in E(x)$ is evidence for the declared property of x ; each arrow acts by a verified evidence transformer $E(f) : E(x) \rightarrow E(y)$. Assume evidence is sound: $E(x) \neq \emptyset \Rightarrow I(x)$.

Proposition 6.2. The category $\int E$ has objects (x, p) and arrows $f : (x, p) \rightarrow (y, q)$ satisfying $E(f)(p) = q$. Its projection to $\mathbf{P}$ is a discrete opfibration, and every object in its image is valid.

Proof. Given (x, p) and $f : x \rightarrow y$, its unique lift has target $(y, E(f)(p))$. Identity and composition follow from functoriality. Validity follows from evidence soundness. $\square$

Arbitrary patches need not act on proofs. Multiple certificates may witness one state, so the total evidence space is not simply a subset of states. Reusable proof transformers provide a stronger interface than Section 5’s candidate-specific gate.

7. A site determined by constraint scopes

Fix a finite set D of development coordinates and sets A_d of possible values. Let $\mathcal{C} = \mathcal{P}(D)$, ordered by inclusion. A constraint index k has a nonempty scope $K_k \subseteq D$ and an allowed relation $\Phi_k \subseteq \prod_{d \in K_k} A_d$. Dependencies, quotas, and interface contracts must be represented in these declared scopes for the following result to apply.

Define presheaves on $\mathcal{C}$ by restriction:

$$F(U) = \prod_{d \in U} A_d,$$

$$Q(U) = \{s \in F(U) : \forall k (K_k \subseteq U \Rightarrow s|_{K_k} \in \Phi_k)\}.$$

Restriction preserves validity because any constraint visible in a smaller region is visible in the larger one. Ordinary union covers make F a sheaf, but can fail for Q [10].

Definition 7.1 (scope-aware cover). A family $\{U_i \subseteq U\}$ covers U when its union is U and every constraint scope contained in U is contained in some U_i . Let J_K be the Grothendieck topology generated by these families.

Theorem 7.2 (scope-aware descent). The specified families form a pretopology on $\mathcal{C}$, and Q is a sheaf for J_K .

Proof. The identity family covers. For a cover of U and $W \subseteq U$, the sets $U_i \cap W$ cover W as sets. A scope contained in W is contained in U and hence in one U_i ; it is therefore contained in $U_i \cap W$. Thus covers are stable under pullback. For transitivity, a scope in U lies in some U_i , and then in a member of any scope-aware cover of that U_i . Unions also compose. This proves the pretopology axioms, including the empty cover of the empty region.

Now let $s_i \in Q(U_i)$ agree on intersections. Coordinatewise gluing gives a unique $s \in F(U)$. For any k with $K_k \subseteq U$, choose i containing K_k . Since s_i satisfies Φ_k and agrees with s on that scope, s satisfies it. Thus $s \in Q(U)$, uniquely. The basis sheaf condition gives the claim. $\square$

Equivalently, the canonical map identifies $Q(U)$ with the equalizer of the two restriction maps

$$\prod_i Q(U_i) \rightrightarrows \prod_{i,j} Q(U_i \cap U_j).$$

Example 7.3 (why the topology matters). Take $D = \{a, b\}$, $A_a = A_b = \{0, 6\}$, and one relation $a + b \leq 10$ on scope D . The singleton regions form an ordinary union cover. Their locally valid assignments $a = 6$ and $b = 6$ match on the empty overlap but have no valid global glue. This family is excluded by J_K . Sheafifying Q for ordinary union covers would actually add this unsafe pair: singleton values determine all sections of its associated sheaf. Sheafification is therefore not a correctness repair.

The Grothendieck topos $\text{Sh}(\mathcal{C}, J_K)$ contains Q . Additional scopes can reduce admissible covers. The theorem is sufficient relative to declared scopes; redundancy may make some unnecessary. Failure of this proof strategy alone gives no communication lower bound without operational hypotheses such as Proposition 4.2's.

8. Directed and higher semantics

Source-changing traces are generally noninvertible. Treating every edit as an identity path would erase this directionality. Instead choose a class W of justified semantic equivalences in $\mathbf{P}$. If a semantics $B : NP \rightarrow \mathcal{B}$ into an infinity-category sends W to equivalences, the universal property of localization gives a factorization through

$$NP[W^{-1}] \rightarrow \mathcal{B}.$$

Here N is the nerve. Only the selected semantic equivalences are inverted. Distinct commit digests can then denote equivalent behavior. Higher patch theories supply a related precedent for retaining proofs relating edit sequences [9]; Git itself does not manufacture these equivalences.

For an explicit elementary case, suppose $f_1, \dots, f_n : S \rightarrow S$ each preserve I and commute pairwise. Every permutation of their composite has the same result and preserves I : adjacent transpositions preserve the composite, and induction preserves validity. The commutation hypotheses concern semantic transformations, not merely disjoint textual lines. Partial transformations additionally require stability of their domains.

Let $\mathcal{S}$ denote spaces. The higher ambient categories are

$$\widehat{\mathcal{C}}_\infty = \text{Fun}(\mathcal{C}^{\text{op}}, \mathcal{S}), \quad \mathcal{X}_K = \text{Sh}_\infty(\mathcal{C}, J_K).$$

These are infinity-topoi; the latter is the left exact sheaf localization of the former [11, Proposition 6.2.2.7]. The sheaf Q embeds as a discrete, zero-truncated object of $\mathcal{X}_K$. A space-valued extension $\mathcal{A}$ can retain implementations, interface equivalences, and their coherent identifications. It must separately satisfy descent, expressed by the Čech comparison

$$\mathcal{A}(U) \simeq \text{holim}_{[n] \in \Delta} \prod_{i_0, \dots, i_n} \mathcal{A}(U_{i_0} \cap \dots \cap U_{i_n}).$$

Pairwise equivalences alone need not supply the higher coherence data in this limit. Theorem 7.2 proves only the discrete case. The higher formula is a specification obligation, not a claim that arbitrary collections of agent outputs already form a stack.

9. Operational interpretation and limits

A concrete implementation uses isolated checkouts and pinned bases, stores immutable proposals and evidence, validates the actual integration candidate, and advances one authoritative manifest reference by expected-value update. A stale base triggers reconstruction and reevaluation of affected evidence. Policy changes obey the same gate.

Object exchange may be asynchronous while acceptance coordinates on the root. Proven compositional evidence permits decomposition along declared scopes; file boundaries alone do not. Hashes identify content, signatures support attribution, and authorization controls acceptance. None alone proves correctness.

Runtime side effects need a separate protocol. A Git revert does not undo an email, payment, migration, or robot motion. Idempotency keys, durable execution records, and suitable resource fencing can be added, but they are outside Theorem 5.2. The present safety property concerns accepted artifact descriptions under a fixed interpretation. It does not establish real-world operational safety, autonomous discovery of invariants, correctness of arbitrary generated software, or fault tolerance against a compromised authority.

10. Evidence, authorship, and reproducibility

Proofs are mathematical arguments under stated assumptions. Executable checks cover finite object joins, merge-closure failure, scope-aware descent, and stale acceptance, including a Git expected-value update. These are regression evidence, not proof-assistant verification or empirical evaluation of agents.

The byline **Anonymous** follows the responsible submitter’s instruction. An OpenAI language model assisted with formulation, proofs, writing, source verification, and the executable artifact. No independent peer review is claimed.

References

- [1] Git project. *Git’s core data model*. <https://git-scm.com/docs/gitdatamodel>
- [2] Git project. *git-update-ref*. <https://git-scm.com/docs/git-update-ref>
- [3] Git project. *git-gc*. <https://git-scm.com/docs/git-gc>
- [4] Git project. *gitworkflows*. <https://git-scm.com/docs/gitworkflows>
- [5] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM* 21(7), 558-565, 1978. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/Time-Clocks-and-the-Ordering-of-Events-in-a-Distributed-System.pdf>
- [6] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. <https://bitcoin.org/bitcoin.pdf>
- [7] M. Shapiro, N. Preguica, C. Baquero, M. Zawirski. Convergent and Commutative Replicated Data Types. *Bulletin of the EATCS* 104, 67-88, 2011. <https://www.lip6.fr/Marc.Shapiro/papers/2011/CRDTs-beatcs-2011-06.pdf>
- [8] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, I. Stoica. Coordination Avoidance in Database Systems. *PVLDB* 8(3), 185-196, 2014. <https://www.vldb.org/pvldb/vol8/p185-bailis.pdf>
- [9] C. Angiuli, E. Morehouse, D. R. Licata, R. Harper. Homotopical Patch Theory. *ICFP 2014*, 243-256. Expanded author version: <https://www.carloangiuli.com/papers/hpt-expanded.pdf>
- [10] The Stacks project. *Sites and Sheaves*, Definition 7.7.1, tag 00VM. <https://stacks.math.columbia.edu/tag/00VM>
- [11] J. Lurie. *Higher Topos Theory*. Annals of Mathematics Studies 170, Princeton University Press, 2009, especially Section 6.2.2. <https://www.math.ias.edu/~lurie/papers/HTT.pdf>

Links checked October 24, 2026.