Doing Nothing on Purpose

Rigorous Magic and the Recovery of Meaning in POSIX `sh`

Anders J. Aamodt

Abstract

This paper argues that POSIX `sh`, often treated as a purely instrumental glue language, already contains a latent semantic hinge capable of supporting a rigorous theory of meaning, intention, and agency as represented within code. That hinge is the null command `:`. By treating a disciplined use of `:` not as a trivial no-op but as a **syntactically real, operationally inert act**, we recover a space where meaning can be articulated without causation.

From this single technicality, we develop a method for synthesizing semantics into shell scripts while preserving backward compatibility, operational correctness, and Unix austerity. We introduce a disciplined magical grammar—**evocation, invocation, and casting**—and a small closed set of magical primitives (including wards, promises, vows, witnessing, binding, release, and pact) that together allow scripts to *mean* as well as *do*.

These primitives correspond not to metaphor but to long-standing human techniques for reasoning about intention, obligation, boundary, and consequence. We show how this grammar enables post-hoc semantic lift for existing scripts, yields immediate practical benefits (notably safety through explicit boundaries), and provides a bridge between philosophy of language, programming-language theory, human–computer interaction, and the living tradition of magical language.

The result is an enchanted but fully rigorous shell: one in which a long-standing technical structure and an equally long-standing human vocabulary are at last allowed to recognize one another.

1 Introduction: Why Magical Language Belongs in Rigorous Systems

Any system that coordinates human action must ultimately contend with *intention*: how it is formed, how it is restrained, how it persists through time, and how it is acknowledged when it bears fruit or causes harm. Modern technical languages, including Unix shells, are extraordinarily effective at describing *what to do*. They are notably poor at describing *what one means to be doing*.

Across cultures and centuries, humans developed vocabularies precisely for this missing layer. Among these, what is traditionally called **magical language** is

unusually explicit, fine-grained, and experientially resonant—retaining a lived sense of agency rather than abstracting it away. Far from ornamental, magical terms name distinctions in agency, obligation, boundary, and consequence that routinely reappear, unnamed, in technical discourse. Words such as *ward*, *vow*, *binding*, *witness*, and *release* persist because they encode stable patterns of human autocognition and microcognition: how will touches the world without yet forcing it.

This paper takes the position—explicitly and without rhetorical provocation—that such language is not an alternative to rigor but one of its historical resources. Where modern technical vocabularies often flatten intention into mechanism, magical language preserves intermediate strata: the difference between considering and deciding, between capability and action, between accident and transgression. These distinctions are already operative in practice; they are merely under-articulated, and magical language preserves them in a form that remains cognitively vivid rather than purely schematic.

The surprising claim of this paper is that POSIX `sh`, often treated as a purely instrumental glue language, already contains a latent grammatical hinge capable of hosting these distinctions without altering behavior. That hinge is the null command `:`. By taking `:` seriously—not rhetorically, but technically—we recover a place inside computation where meaning can be articulated without causation. From this single technicality, a disciplined system of **semanthesis** follows.

---

2 The Technical Hinge: : as Illocution Without Perlocution

2.1 POSIX semantics of :

In POSIX `sh`, `:` is a builtin command with the following properties:

- It performs no operation.
- It always returns exit status 0.
- It accepts arbitrary arguments, which are passed to the command and then discarded.

Thus:

```
cmd && : celebrate
```

executes `cmd`; if successful, it executes `:` with argument `celebrate`. Operationally, this is equivalent to `cmd && :`, provided the evocation itself is pure: no command substitutions, assignment side effects, or redirections are smuggled into the phrase.

This qualification matters because expansion and redirection precede command execution:

```
: "$(touch created)"
: > truncated-file
: "${x:=default}"
```

These are not pure evocations. They exploit shell machinery around : to cause effects before the null command can discard its arguments. Semanthesis therefore concerns **disciplined evocation**: null commands whose words are semantic material, not covert mechanisms.

2.2 Why this matters

Unlike comments, pure evocations are **intralanguage**. Their words:

- are parsed by the shell,
- occupy ordinary argument positions,
- occur at a precise point in control flow,
- and are sequenced with other commands.

Yet they do not affect execution. This combination—syntactic reality with disciplined non-interference—is extraordinarily rare in programming languages. It creates a lawful place for *meaning without force*.

We call this property **argument-bearing non-interference**. It is the foundational technicality on which semanthesis rests, and it holds only when evocation remains free of hidden effects.

2.3 : as witnessing

Because : always succeeds, it can host what we term **witnessing**: the act of recognizing that something occurred and that it counts. Witnessing closes a moment semantically without opening a new causal chain. In ordinary shell scripts, success and failure are often silent; meaning evaporates into exit codes. With :, meaning can be acknowledged explicitly, in place.

---

3 Method and Discovery: From Technical Detail to Semantic System

The framework presented here did not originate as a speculative redesign of the shell, nor as an attempt to aestheticize programming practice. It emerged from a sequence of concrete technical observations, each of which imposed constraints that made the next step unavoidable.

The initial observation was strictly mechanical: the POSIX null command `:` is syntactically valid, argument-bearing, and operationally inert when used without effectful expansion or redirection. Unlike comments, which are removed before parsing, pure evocations participate in the grammar of the shell. Their words are ordered and sequenced, yet discarded without effect. This property—here named **argument-bearing non-interference**—reveals that shell scripts already contain a lawful location where language may appear without force.

From this followed a second insight. Because `:` always returns success, it can be embedded anywhere control flow permits a command, including within short-circuit constructs (`&&`, `||`) and sequential lists. This makes it uniquely suited to mark completion, recognition, or judgment without perturbing behavior. The notion of **witnessing** arose here not as metaphor but as a functional description: a way for a script to acknowledge that something has occurred and that it counts, without causing anything further to happen.

The third insight concerned temporality and revision. If witnessing can be added without altering behavior, then meaning can be layered onto scripts *after they are written*. This property—later formalized as **post-hoc semantic lift**—distinguishes semanthesis from annotation systems, logging frameworks, or DSLs. Meaning accretes rather than replaces.

At this point, design pressure became explicit. Human action reliably distinguishes between naming, preparing, and acting. If the shell was to host meaning without ambiguity, it would need to respect the same separation. This led to the articulation of three modes—**evocation**, **invocation**, and **casting**—not as stylistic choices but as the minimal partition required to prevent intention from leaking into causation.

Finally, once mode separation was stable, the question became one of sufficiency. What is the smallest set of concepts capable of expressing boundary, obligation, identity, relation, and closure? Through iterative refinement, a closed set of primitives emerged. Attempts to add further primitives either collapsed into these or proved compositional. At this point, the system reached equilibrium.

Semanthesis was not imposed upon the shell; it was elicited from it by respecting its existing guarantees. Magical language entered the system only after the substrate had already demanded the distinctions it names.

---

4 Speech-Act Theory Revisited (Austin, Precisely)

J. L. Austin's *How to Do Things with Words* dismantles the idea that language merely describes reality. He distinguishes:

- **Locutionary acts**: the utterance itself,

- **Illocutionary acts**: what is done *in saying* (promising, deciding, warning),
- **Perlocutionary effects**: what happens *because* it was said.

Shell scripting traditionally collapses illocution and perlocution. To write a command is to cause an effect. Semanthesis reintroduces illocution as a first-class, inspectable layer by using disciplined `:` phrases as pure illocutionary hosts.

Austin's concept of **felicity conditions** is equally crucial. Speech acts succeed only when performed under the right conventions and authority. Shells already enforce felicity mechanically—via permissions, environment, and syntax. Semanthesis simply gives those conventions an explicit vocabulary.

---

5 Semanthesis: Definition and Scope

Semanthesis is the synthesis of explicit semantics into shell scripts without altering their operational behavior. More precisely, it introduces **infralinguistic operators**: linguistic forms that operate *below* propositional language and *above* raw mechanism. These operators do not describe states of the world; they structure the conditions under which description, decision, and action become meaningful.

In ordinary human cognition, this layer is pervasive but rarely formalized. Humans routinely distinguish between intending and acting, between permitting and forbidding, between recognizing and causing. We refer to this domain as **infracognition**: the field of sub-deliberative distinctions by which agents regulate stance, readiness, restraint, and closure before and after explicit reasoning occurs. Infracognition is not irrational; it is pre-rational and regulative.

Semanthesis externalizes this infracognitive layer into code. Rather than elevating code to natural language, it names the infralinguistic moves programmers already make implicitly—often via comments, conventions, or social norms—and renders them inspectable.

The design goals of semanthesis are therefore:

1. Zero interference with execution semantics.
2. Backward compatibility with existing scripts.
3. Human-legible expression of infracognitive stance.
4. Machine-legible structure for analysis, tooling, and enforcement.

Semanthesis is not natural-language programming, nor is it a DSL in the usual sense. It does not compile English. It introduces a *small, closed, and disciplined lexicon* whose terms earn their place by corresponding to stable distinctions in both human practice and shell mechanics.

---

6 The Magical Grammar: Evoke, Invoke, Cast

We adopt a three-mode grammar that aligns precisely with both magical tradition and shell mechanics.

6.1 Evocation

Evocation names or signifies without force. In `sh`, this is written with `:`.

```
: regret
: ward protect-root-from-rm
```

Evocation records meaning, judgment, or recognition. It must never alter control flow or truth conditions.

6.2 Invocation

Invocation brings power or capability into the current self. In `sh`, this corresponds to sourcing.

```
invoke wards.sh   # equivalent to `. wards.sh`
```

Invocation changes what *can* be done later without doing anything yet.

6.3 Casting

Casting enacts force. It is execution.

```
cast deploy
```

Casting spends the power that invocation made present.

6.4 Why this mapping holds

In classical magical grammar, invocation precedes casting; evocation names without summoning force. This mapping is not decorative. It enforces a clean separation between meaning, capacity, and effect—exactly the separation shell scripting otherwise lacks.

---

7 Aura Semantics: Meaning Without Side-Effects

Each lexeme in semanthesis participates in two distinct but coordinated semantic layers:

- **Aura-meaning**: the significance a term carries when *evoked*—that is, when named without force.
- **Aura-effect**: the behavior a term produces when *cast*—that is, when enacted as an operation.

This distinction mirrors a fundamental feature of human language: the difference between mentioning a concept and acting upon it. Many programming languages collapse this distinction, allowing names to silently acquire causal power. Semanthesis makes the distinction explicit and enforceable.

Aura-meaning may contextualize, authorize, or prohibit future actions. It may shape interpretation, tooling behavior, or human judgment. However, it must never *cause* action by itself. Aura-effect, by contrast, is always explicit, opt-in, and temporally local.

The invariant is strict:

> Meaning may authorize action, but only action may cause action.

By maintaining this separation, semanthesis prevents **semantic leakage**: the silent migration of intention into mechanism. Aura semantics allow magical terms to retain cognitive and cultural richness without becoming covert control structures, preserving both safety and debuggability.

---

8 The Magical Primitives (A Closed Set)

Through design pressure, we arrive at a minimal complete set of primitives:

1. **Wards** — spatial/contextual boundaries.
2. **Promises** — temporal obligations.
3. **Vows** — identity-level constraints.
4. **Witnessing** — semantic closure.
5. **Binding** — relational coupling.
6. **Release** — intentional unbinding.
7. **Pact** — a named reliability contract joining boundary, obligation, and closure.
8. **Mode (Evocation/Invocation/Casting)** — force gating.

These primitives are sufficient to express intention, obligation, and restraint. Specialized forms—such as enthralling a lock or disenthralling it afterward—are scripts of binding and release, not separate primitives. Everything else—cantrips, rites, enchantments—is compositional.

---

9 Wards as a Canonical Application

Wards provide an immediate, non-controversial demonstration of semanthesis's utility. A ward is a persistent declarative protection that constrains *how* actions may occur without prescribing *what* must occur.

Examples:

- `protect-root-from-rm`
- `delete-sends-to-trash`
- `protect-keys-and-secrets`

In an implementation, wards may be toggled globally or individually (`wards on|off`, `ward NAME on|off`). Existing shell options (`set -e`, `set -u`) can be re-expressed as wards, converting cryptic flags into named boundaries.

Crucially, wards operate *around* commands, not inside them, enabling post-hoc safety for legacy code. Some wards are prohibitive: they name a taboo. Others are procedural: they name the conditions under which one may transgress the taboo without pretending the danger has disappeared.

---

10 Promises and Vows: Time and Identity in Code

A **promise** records a future obligation that may be fulfilled, released, or broken—but never silently forgotten.

```
: promise cleanup
```

A **vow** binds identity and is costly to break.

```
vow no-unverified-deploys
```

Promises address forgetfulness; vows address rationalization. Neither replaces mechanism; both make intention inspectable.

---

11 Post-Hoc Semantic Lift and Interpretability Reserves

Because pure evocation is non-interfering, existing scripts can gain new semantic structure without altering behavior. We call this property **post-hoc semantic lift**: the ability to ascribe additional, structured meaning to code *after it has been written* without falsifying its behavior.

Post-hoc semantic lift implies the existence of what we term **interpretability reserves**. A script possesses interpretability reserves when it contains lawful positions where meaning may later be attached, refined, or reinterpreted without altering behavior. In semanthesis, pure `:` phrases constitute such positions, precisely because they are syntactically real yet causally inert.

Interpretability reserves are rare in programming languages. In most systems, execution and meaning are so tightly coupled that semantic enrichment requires

rewriting code, introducing wrappers, or altering control flow. By contrast, semanthesis allows meaning to accrete gradually, supporting refactoring at the level of intention rather than mechanism.

This property is not merely ergonomic. It enables historical layering: scripts may acquire richer semantics as contexts change, norms evolve, or tooling improves, without invalidating their past behavior. In this sense, interpretability reserves function as a form of semantic capital preserved in the codebase.

---

12 Relation to Prior Art

Semanthesis intersects with several established intellectual and technical traditions, while remaining reducible to none of them. Its contribution lies not in novelty of components but in the specific conjunction and disciplined interaction of those components.

Speech-act theory (Austin; Searle) provides the philosophical groundwork for distinguishing meaning from effect. However, speech-act theory remains largely descriptive and sociolinguistic. It offers no account of how illocutionary force might be represented *inside* executable systems without collapsing into perlocution. Semanthesis supplies a concrete mechanism for this separation by identifying a syntactic position—`:`—where illocution may occur without effect.

Literate programming (Knuth) foregrounds human comprehension, but commentary remains extralinguistic: it is not ordered, scoped, or evaluated by the runtime. Semanthesis differs in that its semantic utterances are intralinguistic. They occur in sequence, may be conditioned on control flow, and are therefore available to tooling, analysis, and disciplined interpretation.

Logic and rule-based systems (e.g., Prolog) encode intention declaratively, but only by abandoning the imperative execution model that makes shells effective for systems work. Semanthesis preserves the imperative shell intact, adding a declarative semantic stratum that constrains and contextualizes imperative action rather than replacing it.

Modern shells and data-oriented languages (such as Nushell) improve type safety, data flow, and compositionality. Yet they continue to treat meaning as an emergent property of execution rather than a first-class concern. Semanthesis is orthogonal to these efforts: it could, in principle, be applied within such environments wherever argument-bearing non-interference exists.

Agent-oriented frameworks (BDI models) explicitly model belief, desire, and intention, but require wholesale architectural adoption and continuous runtime mediation. Semanthesis operates incrementally and post-hoc. It does not simulate agents; it formalizes the human–machine interface where intention already constrains action.

What distinguishes semanthesis is the conjunction of three properties rarely found together: intralanguage illocution, zero operational interference, and backward compatibility. It is this conjunction, rather than any individual feature, that defines its contribution.

---

13 Why Magical Language Is the Correct Vocabulary

Magical language is not metaphor here. It is humanity's oldest living ontology of intention, boundary, obligation, and consequence. It constitutes a mature **infralinguistic system** for naming infracognitive distinctions: those subtle but decisive differences in stance by which humans regulate action before and after explicit reasoning.

Modern technical vocabularies excel at naming mechanisms and states. They are comparatively impoverished at naming *postures*: readiness versus hesitation, permission versus taboo, commitment versus exploration. Programmers therefore rely on informal metaphors—"dangerous", "safe", "clean", "dirty"—without a stable grammar to support them. Magical language already provides such a grammar. It distinguishes wards from prohibitions, vows from promises, bindings from mere dependencies, witnessing from logging, and release from failure.

A further advantage is pragmatic. Magical terms are underused in contemporary technical contexts, resulting in low semantic collision. They carry dense, specific connotations without overloading existing keywords, and they remain intelligible to non-specialists. A reader may not know what an invariant is, but they understand a ward; they may not know what a temporal constraint is, but they understand a promise.

There is also a political dimension that cannot be ignored. Magical language historically enables autonomous self-modification: the capacity of individuals to reprogram their own habits, boundaries, and commitments without institutional mediation. Its marginalization in modern discourse coincides with the externalization of agency into platforms, policies, and tools. Semanthesis deliberately restores this vocabulary at the precise interface where humans and machines coordinate action.

Finally, magical language survives because it encodes failure modes. A spell may misfire; a vow may be broken; a ward may be lowered; a binding may entangle. These are structured possibilities, not bugs. By giving such distinctions operational traces through aura semantics, the system renders magic rigorous rather than vague.

---

14 Objections and Replies

Objection: This is merely commentary with theatrical naming.

Reply. Commentary is extralinguistic: it is removed before parsing and lacks a determinate position in control flow. Evocations are intralinguistic: they are parsed, expanded, sequenced, and therefore analyzable. The distinction is structural, not rhetorical.

Objection: This smuggles new behavior into old scripts.

Reply. Semanthesis is defined by disciplined non-interference. Evocation must be observational. If a `:` phrase uses command substitution, assignment expansion, redirection, or any other hidden mechanism to cause state transition, it is not a semanthetic evocation but an ordinary shell effect disguised as one.

Objection: This anthropomorphizes machines.

Reply. The framework models human intention as an interface constraint on mechanism. It does not claim machine experience, consciousness, or inner life; it claims only that intention can be represented in code in a way that is legible, inspectable, and enforceable.

Objection: The vocabulary is arbitrary or culturally contingent.

Reply. The choice is constrained, not arbitrary. Magical terms are adopted only where they correspond to stable infracognitive distinctions that the substrate already demands (boundary, obligation, closure, binding). Alternative vocabularies could be mapped, but would typically sacrifice either folk intelligibility or semantic specificity.

Objection: This introduces unnecessary verbosity.

Reply. Semanthesis is optional and additive. It introduces no required syntax and alters no existing semantics. Where explicit intention is valuable, evocation provides it; where it is not, silence remains available.

Objection: The system is not formally specified.

Reply. The core invariants are formalizable: disciplined argument-bearing non-interference, separation of aura-meaning and aura-effect, and mode-gated causation. The present work establishes the conceptual and technical substrate; a full formalization is a straightforward continuation rather than a prerequisite for validity.

---

15 Conclusion: The Return of Meaning

Unix shells never lacked power. What they lacked was a lawful place for meaning to land: a syntactic location where intention could be articulated before becom-

ing force. That location has existed since the earliest POSIX shells, available but untheorized.

By taking the null command `:` seriously, semanthesis recovers a grammar of will that is at once ancient and computationally exact. Through evocation, invocation, and casting, scripts gain the ability to mean as well as to act. Through wards, vows, promises, and witnessing, intention becomes inspectable, revisable, and accountable.

The broader significance of this work extends beyond shell scripting. It demonstrates that rigor and enchantment are not opposed. Where technical systems respect their own invariants, human semantic vocabularies—especially those evolved to describe intention—may be reintroduced without loss of precision.

To do nothing, in the right place, is not emptiness. It is the precondition of deliberate action.

References

Austin, J. L. *How to Do Things with Words*. Oxford University Press, 1962.

Searle, John R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

Knuth, Donald E. *Literate Programming*. CSLI Publications, 1992.

IEEE POSIX Working Group. *IEEE Std 1003.1-2017 (POSIX.1-2017): The Open Group Base Specifications Issue 7*. IEEE, 2018.

McIlroy, M. D., Pinson, E. N., and Tague, B. A. "UNIX Time-Sharing System: Forward." *Bell System Technical Journal*, vol. 57, no. 6, 1978.

Norman, Donald A. *The Design of Everyday Things*. Basic Books, 1988.

Brooks, Frederick P. *The Mythical Man-Month*. Addison-Wesley, 1975.

Winograd, Terry, and Fernando Flores. *Understanding Computers and Cognition*. Ablex Publishing, 1986.

Suchman, Lucy A. *Plans and Situated Actions*. Cambridge University Press, 1987.

Dukes, Ramsey. *SSOTBME: An Essay on Magic*. Thoth Publications, 2002.