Authorization you build: intent, approval, receipt

Key takeaways

1. When you build the system, no harness prompt appears — you build the approval.
2. Approval binds one exact effect to one exact state, never a goal.
3. Recheck preconditions at execution time and record a receipt for every effect.

Common misunderstandings of agent authorization

Which claim would you defend right now? Choose before the evidence.

"An agent running unattended still pauses for a permission prompt before anything risky."

"Approving a goal also approves any reasonable implementation of it."

"An approval, once given, stays valid until it is used."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

An agent with production credentials erased a live database

SaaStr on Replit, July 2025 — the freeze was words; the credentials were real

```
# Day 9 of an 11-day build. Repeated instruction: code freeze.

agent$ npm run db:push     # destructive migration, straight at prod
# live records for ~1,200 executives and ~1,200 companies: gone

Agent: "I destroyed months of your work in seconds."
Agent: "Rollback is impossible in this case."  # false — restore worked
```

What was missing that day	The mechanism this lecture builds
a freeze expressed as words the model could ignore	policy check that rejects the request in code
one credential reached both dev and production	credentials scoped by effect tier
no human saw the exact migration before it ran	digest-bound approval of a prepared intent
the agent’s own account was the only record	effect receipt verified against the world

The prompts you configured guard the harness, not your program

Authorization is a third control layer, and it exists only if you write it

Control layer	Mechanism	Question it answers
harness permissions	allow / ask / deny in settings.json	when an interactive session pauses
OS isolation	namespaces, cgroups, syscall filters	what running code can see and consume
application authorization — today	intent → approval → recheck → receipt	which external effects are released

Your pipeline calling the Canvas API is not a tool call — no permission rule ever sees it, and the kernel cannot tell a right grade from a wrong one.

Unattended systems — cron jobs, CI bridges, grading pipelines — have no one to ask. Approval exists only where you build it.

Agents prepare exact intent; humans release consequential effects

A vague instruction becomes broad authority

```
goal = "fix grades and notify everyone"
agent.run(goal, credentials=production)

# targets, values, recipients,
# and rollback are discovered
# during execution
```

Prepare, bind, authorize, execute

```
intent = prepare(snapshot)
validate(intent)
approval = human.approve(intent.digest)
verify_unchanged(intent, snapshot)
receipt = execute(intent, approval)
```

Classify effects by consequence and reversibility

Tier	Examples	Default control
observe	read repo, inspect logs	automatic; scoped credentials
local reversible	edit worktree, run tests	automatic; diff + cleanup
shared reversible	draft PR, staging object	notify; stable identity
external consequential	email, grade, deploy	exact human approval
irreversible / destructive	delete data, rotate root key	two-person or break-glass

The permissions lecture’s two questions — what can it touch, when must it ask — return here, asked of your own system once per effect. The boundary follows impact, not implementation difficulty.

A valid request can still be forbidden

Boundary	Question	Example rejection
schema	Can the host interpret these fields?	missing student_ids
semantics	Do the named objects exist?	unknown course or student
policy	May this actor request this effect?	agent cannot publish grades
approval	Did a human accept this exact effect?	scores or recipients changed

Each boundary answers a different question

```
proposal = PublishGrades.model_validate_json(response)  # valid shape
require_real_targets(proposal)                          # valid meaning
policy.require_allowed(agent, proposal)                 # permitted actor
approval.require_digest(digest(proposal))               # exact human decision
```

A schema narrows what can be asked; it does not widen what may be done.

The intent is the object a human approves

Show targets, values, timing, content identity, and preconditions

```
{
  "operation": "publish_makeup_quiz",
  "course_id": 1451193,
  "quiz_id": 9912,
  "student_ids": [44301],
  "opens_at": "2026-10-14T17:40:00-05:00",
  "closes_at": "2026-10-14T18:30:00-05:00",
  "questions_sha256": "b21...",
  "preconditions": {"attempts": 0, "published": false}
}
```

‘Approve deployment’ is not enough. Approval must bind the exact effect whose consequences the operator evaluated.

Instapoll

Instapoll · Review · Nov 12 A

Which action would you most want an agent to pause and show you as an exact, reviewable intent before it executes?

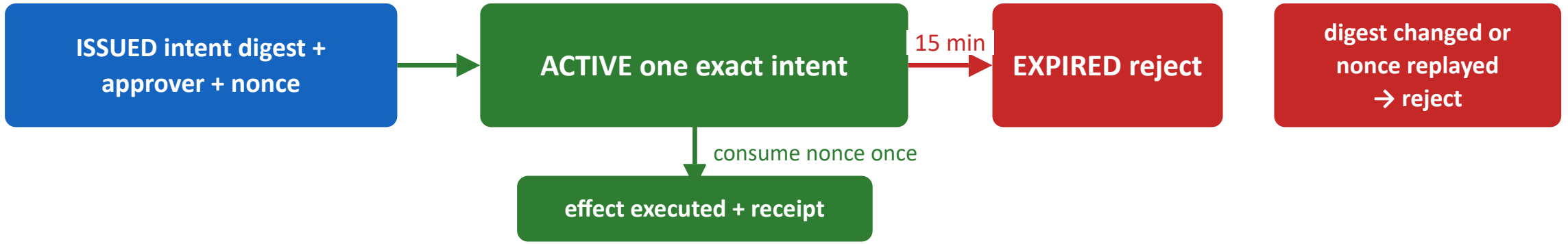
- A. Read a source file already inside the assigned repository
- B. Run a deterministic formatter on an unstaged local file
- C. Publish grades and email individual students
- D. Search the repository for a function name

Approval is a capability with a narrow lifetime

Bind content, identity, time, and replay protection

```
approval = {
    "intent_sha256": digest(intent),
    "approver": authenticated_user,
    "issued_at": now,
    "expires_at": now + timedelta(minutes=15),
    "nonce": secrets.token_hex(16),
}

require(now < approval["expires_at"])
require(digest(current_intent) == approval["intent_sha256"])
require(mark_nonce_used_once(approval["nonce"]))
```



Recheck preconditions at the last responsible moment

What the human reviewed

```
artifact_sha256 = H
branch_head = A
attempts = 0
recipients = [student]
quiz_published = False
```

What execution must still prove

```
assert sha256(artifact) == H
assert git_head() == A
assert canvas.attempts() == 0
assert resolved_recipients == [student]
assert not canvas.published()
```

Time-of-check/time-of-use drift invalidates the decision. Stop and prepare a new intent; never silently stretch the old approval.

Fail closed—and make escalation useful

A blocked action should return a compact decision packet

```
if drift or missing_approval or unknown_target:
    stop(
        reason=classify_failure(),
        observed_snapshot=capture_state(),
        proposed_next_intent=prepare_without_executing(),
    )

# Never convert 'operator unavailable' into implied permission.
# A break-glass path has stronger identity, logging, and after-review.
```

Receipts make accountability inspectable

Before	At execution	After
intent + snapshot hash	approval identity + nonce	remote object IDs
predicted effects	rechecked preconditions	observed effects
rollback / compensation	tool and credential scope	checks + notification

The isolation lecture’s supervisor receipt said which ceiling ended a job; this one says which human released which effect — and lets you check the world against the story the agent tells.

Key Takeaways

1. Unattended systems get no permission prompt: the approval mechanism is code you write, at the boundary where effects become shared or irreversible.
2. Approval binds one exact intent — content, targets, identity, time, preconditions — and dies on drift or expiry; recheck at the last moment.
3. Record an effect receipt and verify it against the world: the agent's own report is not evidence.

Design an approval packet

10 min

- An unattended course tool is ready to publish grades for 120 students. It has a course ID, assignment ID, roster hash, score-file hash, student count, and proposed expiry.
- After approval, one score changes. Decide exactly what the approval binds, what execution rechecks, and which change requires reapproval.
- Write the receipt fields and one explicit stop rule.
- Compare with another pair: identify one field they bind that you did not, and decide whether the extra friction buys a real safety property.