Facsímil 05

Agentes y orquestación

Capítulo 08

Permisos, autonomía y supervisión humana

686f6c61 · 2026

Capítulo 08: Permisos, autonomía y supervisión humana

Autonomía no significa carta blanca

En el capítulo 01 dijimos que un agente no es “un prompt largo”, sino un sistema que puede observar, decidir y actuar. En el capítulo 03 aprendimos que una tool no es una función cualquiera: tiene contrato, permisos, errores y efectos. En el capítulo 06 pusimos harness alrededor del agente. Y en el capítulo 07 vimos cómo los SDKs exponen permisos, hooks, trazas y aprobaciones.

Ahora toca una pieza que suele decidir si un sistema agentic es publicable: **quién puede hacer qué, cuándo, con qué evidencia y bajo qué revisión**.

Un agente no debería vivir entre dos extremos: “no puede hacer nada” o “puede hacerlo todo”. La autonomía útil se diseña por capas. Puede leer sin preguntar, proponer cambios, preparar una acción, pedir aprobación antes de ejecutarla, ejecutar automáticamente acciones pequeñas dentro de un margen y detenerse cuando algo sale del contrato.

Qué no es supervisión humana

Supervisión humana no es poner un botón de “aceptar” al final de una pantalla. Si la persona no entiende qué va a ocurrir, qué datos se usaron, qué herramienta se ejecutará y cómo volver atrás si hace falta, no está supervisando: está firmando a ciegas.

Tampoco es revisar todas las acciones. Eso convierte el sistema en una cola lenta y enseña a la gente a pulsar “sí” sin leer. La revisión debe aparecer donde aporta juicio: acciones con efecto externo, coste alto, incertidumbre, impacto sobre otra persona, modificación persistente o falta de evidencia.

Y no es delegar responsabilidad al modelo. El modelo puede sugerir. La política decide. El harness ejecuta. La traza demuestra.

La definición útil

Para este facsímil, un sistema de permisos agentic es:

Una capa de decisión que evalúa cada acción propuesta por el agente y devuelve `allow`, `approval_required` o `deny`, dejando evidencia suficiente para explicar la decisión y reanudar la ejecución.

La decisión de permiso es una **función de autorización**, el núcleo de cualquier monitor de referencia: para cada acción decide allow, approval o deny según el actor, el recurso, el entorno y el estado.[1]

$$D(a, s, u, r, e) \in \{\text{allow}, \text{approval}, \text{deny}\}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
D	Decisión de permiso.	Permitir, pedir aprobación o denegar.
a	Acción propuesta.	Enviar email, editar archivo, consultar CRM, crear ticket.
s	Estado de la ejecución.	Paso actual, coste, intentos, evidencia acumulada.
u	Usuario o actor responsable.	Alumno, profesor, operador, sistema nocturno.
r	Recurso afectado.	Documento, base de datos, repositorio, cliente, factura.
e	Entorno.	Desarrollo, preproducción, producción, demo, laboratorio.

La decisión no depende solo de la tool. Depende de la tool **con argumentos concretos**. No es lo mismo `send_email(to="yo@example.com")` que `send_email(to="lista-completa@example.com")`. No es lo mismo editar una propuesta local que publicar un cambio persistente. El permiso real vive en el cruce entre acción, recurso, usuario, entorno y momento.

Fecha de corte del estado del arte

Fecha de corte: 10 de junio de 2026.
Fuentes consultadas ese día: documentación oficial de OpenAI Agents SDK sobre human-in-the-loop, guardrails y ejecución; documentación oficial de Anthropic sobre permisos y hooks del Claude Agent SDK; documentación oficial de Google ADK sobre callbacks y controles en herramientas; documentación oficial de LangGraph sobre interrupts y persistencia; NIST AI Risk Management Framework como marco general de gobierno y gestión de riesgo.

Lo estable es el patrón: permisos explícitos, aprobación estructurada, pausa/reanudación, trazas, gates, límites de alcance y revisión humana donde aporta criterio. Lo cambiante son nombres de parámetros, APIs de SDK, modos de permiso, conectores y detalles de producto.

Niveles de autonomía

La autonomía se diseña mejor como una escala:

NIVEL	NOMBRE	QUÉ PUEDE HACER	EJEMPLO
A0	Responder	Solo produce texto.	Explicar una política interna.
A1	Leer	Puede consultar recursos permitidos.	Buscar una referencia en una carpeta autorizada.
A2	Preparar	Puede crear una propuesta, no ejecutarla.	Redactar email o diff sin enviarlo.
A3	Ejecutar con aprobación	Puede actuar tras revisión explícita.	Publicar una nota, enviar email, modificar registro.
A4	Ejecutar dentro de margen	Puede actuar automáticamente si cumple umbrales.	Clasificar tickets de baja criticidad.
A5	Operar con guardia	Puede ejecutar flujos largos, pero con límites, alertas y gates.	Monitorizar cola y escalar casos fuera de patrón.

La escala no se asigna al agente entero. Se asigna a cada acción.

ACCIÓN	NIVEL RAZONABLE	POR QUÉ
Leer documentación pública	A1	No cambia estado.
Leer datos internos	A1 con scope	Requiere identidad, recurso y motivo.
Redactar respuesta	A2	No hay efecto externo todavía.
Enviar respuesta a una persona	A3 o A4	Depende del canal, contenido y confianza.
Modificar una base de datos	A3	Persistente y difícil de corregir sin traza.
Cambiar configuración de producción	A3 con doble revisión	Alto impacto operacional.
Ejecutar una herramienta de coste alto	A3	Afecta presupuesto.

La regla práctica: **la autonomía sube cuando el efecto es reversible, barato, acotado y bien evaluado; baja cuando el efecto es persistente, amplio, caro o incierto**.

Matriz de permisos

Un permiso serio no es una lista de herramientas: es una política de **control de acceso basado en atributos** (ABAC), que decide según atributos del sujeto, la acción, el recurso y el entorno.[2] Sus atributos:

CAMPO	PREGUNTA	EJEMPLO
`actor`	¿Quién responde por la acción?	`profesor`, `sistema_soporte`, `alumno_lab`.
`action`	¿Qué tipo de acción es?	`read`, `draft`, `write`, `send`, `delete`, `publish`.
`resource`	¿Sobre qué recurso?	`tickets`, `notas`, `repo`, `crm`, `email`.
`scope`	¿Con qué alcance?	Solo curso actual, solo carpeta del proyecto, solo cliente asignado.
`env`	¿En qué entorno?	`dev`, `staging`, `prod`.
`budget`	¿Con qué límite?	3 tools, 0,20 EUR, 90 segundos, 2 ficheros.
`evidence`	¿Qué debe demostrar antes?	Cita encontrada, test pasando, diff visible.
`expiry`	¿Cuánto dura?	Esta run, 10 minutos, una sesión, una release.

Un ejemplo de permiso mal diseñado:

```
tool: send_email
permission: allowed
```

Un ejemplo más publicable:

```
actor: soporte_nivel_1
action: send
resource: email
scope:
  recipients: ["usuario_actual"]
  templates: ["respuesta_estado_matricula", "peticion_documentacion"]
env: prod
budget:
  max_messages: 1
  max_tokens: 900
evidence:
  required_fields: ["ticket_id", "user_id", "reason", "draft"]
decision:
  if_template_known_and_no_personal_claim: allow
  else: approval_required
expiry: run
```

La diferencia es enorme: el segundo permiso se puede auditar, probar y explicar.

Fórmula de riesgo operativo

El riesgo operativo de una acción se puede puntuar como una suma ponderada de los factores que lo elevan (efecto externo, irreversibilidad, coste, incertidumbre, novedad) menos lo que lo baja (verificación disponible). Es un score de riesgo multiatributo, en la línea de los marcos de gestión de riesgo de IA que obligan a identificar y ponderar las fuentes de riesgo.[3] No es una fórmula perfecta; obliga a mirar las variables correctas:

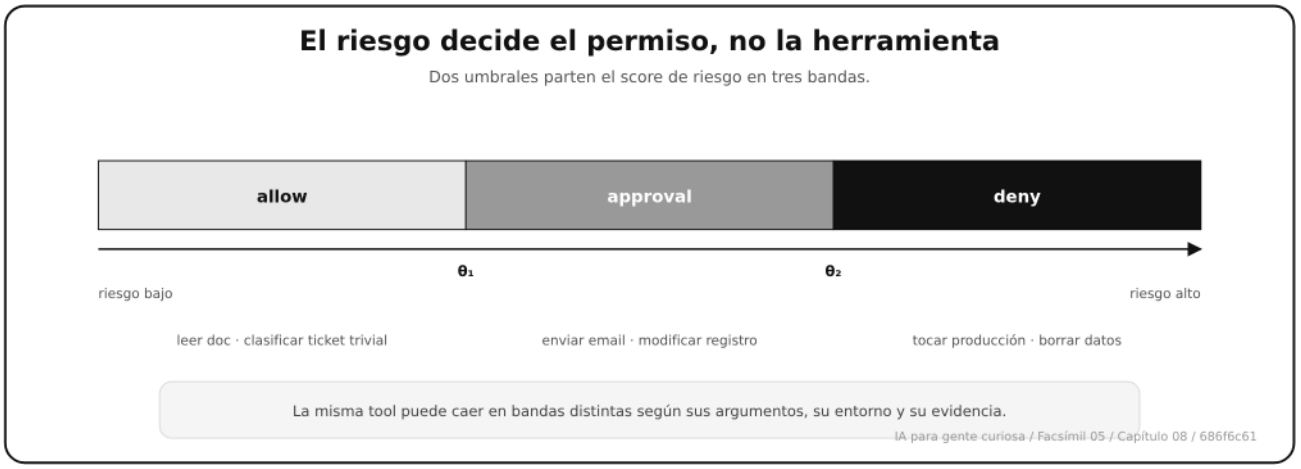
$$\rho(a) = w_E E(a) + w_R R(a) + w_C C(a) + w_U U(a) + w_N N(a) - w_V V(a)$$

TÉRMINO	QUÉ MIDE	EJEMPLO
$E(a)$	Efecto externo.	Enviar, publicar, cobrar, modificar.
$R(a)$	Reversibilidad.	Se puede deshacer fácil o no.
$C(a)$	Coste.	Tokens, API externa, GPU, tiempo humano.
$U(a)$	Incertidumbre.	Falta evidencia o confianza.
$N(a)$	Novedad.	Acción poco probada o fuera de patrón.
$V(a)$	Verificación disponible.	Tests, schema, cita, diff, regla determinista.
w	Pesos del dominio.	No pesan igual en educación que en facturación.

Y decidimos por bandas de riesgo, como en una matriz de riesgo clásica:

$$D(a) = \begin{cases} \text{allow} & \rho(a) < \theta_1 \\ \text{approval} & \theta_1 \le \rho(a) < \theta_2 \\ \text{deny} & \rho(a) \ge \theta_2 \end{cases}$$

Esto no sustituye al criterio. Lo documenta. Si una acción queda en `approval`, la persona no revisa “todo el agente”; revisa una acción concreta con evidencia concreta.



Para las acciones de la banda más alta, la aprobación de una sola persona a veces no basta. La seguridad clásica llama a esto **separación de privilegio**: ninguna acción crítica debería depender de una sola llave.[4] En un agente se traduce en la regla de dos personas para desplegar a producción, borrar datos o mover dinero: el agente prepara, una persona revisa y otra confirma. No es burocracia; es la diferencia entre un error recuperable y uno irreversible.

Lo que dicen los SDKs actuales

OpenAI Agents SDK documenta un flujo human-in-the-loop donde la ejecución puede pausar hasta que una persona aprueba o rechaza llamadas a tools; las interrupciones pueden aparecer en tools normales, MCP hospedado y agentes usados como tools.[5] También distingue guardrails de entrada, salida y herramientas, y recomienda tool guardrails cuando hay managers, handoffs o especialistas delegados.[6]

Anthropic Claude Agent SDK permite controlar tools mediante modos de permiso, allow/deny lists, callbacks y hooks; además, sus hooks permiten intervenir antes o después de tool use, parada, notificaciones o subagentes.[7][8]

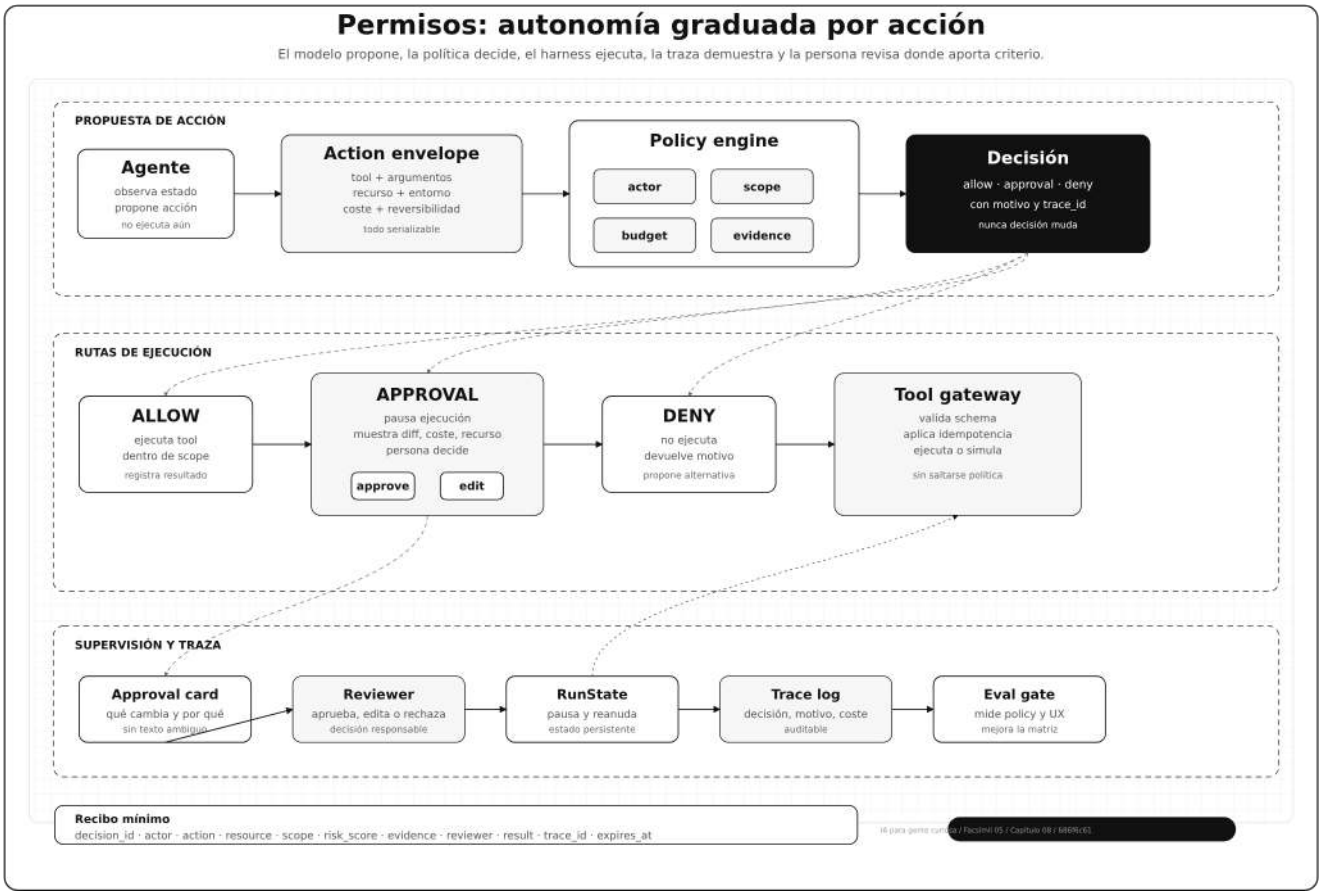
Google ADK coloca los callbacks como puntos de observación y control antes/después de agente, modelo y tools; los callbacks de tool permiten intervenir justo antes o después de que una herramienta se ejecute.[9] Su documentación de controles en agentes insiste en diseñar

herramientas defensivamente y usar callbacks como capas de validación y control.[10]

LangGraph usa `interrupt()` para pausar una ejecución, guardar estado con checkpointer y reanudar con `Command`; esto es importante porque la aprobación humana no debería perder el estado de la ejecución.[11]

La coincidencia entre ecosistemas es clara: las aprobaciones no son un modal decorativo. Son una primitiva de ejecución: pausar, mostrar evidencia, decidir, reanudar y dejar traza.

Anatomía visual de permisos y supervisión



La figura separa propuesta, decisión, ejecución, revisión y traza. Esa separación es el corazón del capítulo. El agente no debería llamar una tool “porque sí”. Debe producir un sobre de acción. La política lo evalúa. Si la respuesta es `approval`, la ejecución se pausa y la persona recibe una tarjeta revisable. Después se reanuda con estado, no desde cero.

Qué debe llevar una tarjeta de aprobación

Una aprobación humana útil no pregunta “¿permitir?”. Pregunta algo revisable:

CAMPO	POR QUÉ IMPORTA	EJEMPLO
Acción	La persona debe saber qué se ejecutará.	`send_email`, `publish_page`, `update_record`.
Recurso	Qué elemento se verá afectado.	Ticket `T-1042`, archivo `capitulo-08.md`.
Cambio propuesto	Qué diferencia concreta habrá.	Diff, email final, campos modificados.
Motivo	Por qué el agente propone hacerlo.	“Falta documentación solicitada”.
Evidencia	Qué ha comprobado.	URL, test, cita, consulta, fuente.
Coste	Cuánto consume continuar.	Tokens, herramienta externa, tiempo.
Reversibilidad	Cómo se corrige si no era adecuado.	Rollback, edición manual, nueva versión.
Alternativas	Qué pasa si se rechaza.	Guardar propuesta, pedir más datos, escalar.
Expiración	Cuándo deja de valer la decisión.	Esta run, 10 minutos, versión actual.

Si la tarjeta no contiene evidencia, el revisor se convierte en oráculo. Y las personas no son oráculos: necesitan contexto, comparación y consecuencias.

Tarjeta de aprobación en Claude Agent SDK

En Anthropic, la tarjeta no debería generarla el modelo como texto libre. La tarjeta debería nacer en tu aplicación cuando el SDK llama a `can_use_tool`. La documentación actual explica que Claude pide entrada de usuario en dos casos: cuando necesita permiso para usar una tool y cuando llama a `AskUserQuestion`; ambos pasan por `canUseTool` / `can_use_tool`, y la ejecución queda pausada hasta que devuelves una respuesta.[1][2]

El orden técnico importa:

```
Claude pide tool
  -> hooks PreToolUse
  -> reglas deny
  -> permission_mode
  -> reglas allow
  -> can_use_tool
  -> allow / deny con mensaje
```

Por eso la tarjeta debe construirse con datos del runtime, no con una frase del modelo. Para un producto real, usaría esta forma:

```
{
  "approval_id": "appr_01J...",
  "provider": "anthropic",
  "sdk": "claude-agent-sdk",
  "session_id": "session_...",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "fasciculo-05-agentes-orquestacion/08-permisos-autonomia-supervision-
humana.md"
  },
  "summary": "Claude quiere escribir cambios en el capítulo 08.",
  "why_review": "La tool modifica un archivo persistente.",
  "risk": {
    "effect": "write",
    "environment": "dev",
    "reversible": true,
    "score": 0.42
  },
  "evidence": [
    "Capítulo actual en revisión",
    "Cambio limitado al facsímil 05",
    "Build de Astro pendiente tras aprobar"
  ],
  "choices": [
    "approve_once",
    "approve_with_changes",
    "reject"
  ],
  "expires_at": "2026-06-10T10:45:00+02:00"
}
```

La UI visible podría mostrar algo así:

CAMPO EN PANTALLA	EJEMPLO
Acción	`Write` sobre capítulo 08.
Motivo	Modifica un archivo persistente.
Alcance	Solo `fasciculo-05-agentes-orquestacion/08...md`.
Evidencia	El cambio viene de una petición explícita del autor.
Riesgo	Escritura reversible en entorno de desarrollo.
Opciones	Aprobar una vez, aprobar con cambios, rechazar.

Y el código conceptual en Python quedaría así:

```python
import asyncio
import time
from dataclasses import dataclass, asdict

from claude_agent_sdk import ClaudeAgentOptions, ResultMessage, query
from claude_agent_sdk.types import (
    HookMatcher,
    PermissionResultAllow,
    PermissionResultDeny,
    ToolPermissionContext,
)


@dataclass
class ApprovalCard:
    approval_id: str
    provider: str
    sdk: str
    tool_name: str
    tool_input: dict
    summary: str
    why_review: str
    risk_score: float
    choices: list[str]
    expires_in_seconds: int


def summarize_tool(tool_name: str, input_data: dict) -> tuple[str, str, float]:
    if tool_name in {"Write", "Edit"}:
        path = input_data.get("file_path", "archivo sin ruta")
        return (
            f"Claude quiere modificar {path}.",
            "La tool cambia un recurso persistente.",
            0.42,
        )

    if tool_name == "Bash":
        command = input_data.get("command", "")
        return (
            f"Claude quiere ejecutar: {command}",
            "La tool ejecuta un comando del sistema.",
            0.58,
        )

    return (
        f"Claude quiere usar {tool_name}.",
        "La tool no está autoaprobada por la política actual.",
        0.35,
    )


async def ask_approval_ui(card: ApprovalCard) -> dict:
```

```
    """
    En producción esto sería tu UI: web, app interna, Slack, consola,
    cola de revisión o sistema de tickets.
    """
    print("\n=== APPROVAL CARD ===")
    print(card.summary)
    print("Motivo:", card.why_review)
    print("Riesgo:", card.risk_score)
    print("Input:", card.tool_input)
    print("Opciones:", ", ".join(card.choices))

    # Simulación para el libro: una UI real devolvería también input editado.
    return {"decision": "reject", "message": "Revisar manualmente antes de ejecutar."}


async def can_use_tool(
    tool_name: str,
    input_data: dict,
    context: ToolPermissionContext,
) -> PermissionResultAllow | PermissionResultDeny:
    summary, why_review, risk = summarize_tool(tool_name, input_data)

    card = ApprovalCard(
        approval_id=f"appr-{int(time.time())}",
        provider="anthropic",
        sdk="claude-agent-sdk",
        tool_name=tool_name,
        tool_input=input_data,
        summary=summary,
        why_review=why_review,
        risk_score=risk,
        choices=["approve_once", "approve_with_changes", "reject"],
        expires_in_seconds=600,
    )

    decision = await ask_approval_ui(card)

    if decision["decision"] == "approve_once":
        return PermissionResultAllow(updated_input=input_data)

    if decision["decision"] == "approve_with_changes":
        updated_input = {**input_data, **decision.get("updated_input", {})}
        return PermissionResultAllow(updated_input=updated_input)

    return PermissionResultDeny(message=decision["message"])


async def keep_stream_open(_input_data, _tool_use_id, _context):
    return {"continue_": True}


async def prompt_stream():
    yield {
        "type": "user",
        "message": {
```

```
            "role": "user",
            "content": "Propón una mejora concreta del capítulo 08 y prepara el cambio.",
        },
    }


async def main():
    async for message in query(
        prompt=prompt_stream(),
        options=ClaudeAgentOptions(
            permission_mode="default",
            can_use_tool=can_use_tool,
            hooks={"PreToolUse": [HookMatcher(matcher=None, hooks=[keep_stream_open])]},
        ),
    ):
        if isinstance(message, ResultMessage):
            print(message.subtype, message.result)


asyncio.run(main())
```

Hay tres detalles finos:

DETALLE	POR QUÉ IMPORTA
`permission_mode="default"`	Si todo cae en `bypassPermissions`, no hay tarjeta útil: el SDK aprobará demasiadas cosas.
`can_use_tool`	Es el punto donde tu app puede construir la tarjeta y devolver `PermissionResultAllow` o `PermissionResultDeny`.
`updated_input`	Permite aprobar con cambios: por ejemplo, limitar ruta, cambiar comando o acotar destinatario.
Hook `PreToolUse`	En Python, la documentación indica que el flujo con `can_use_tool` requiere streaming y un hook que mantenga la sesión abierta.

Si la persona tarda demasiado, no intentaría mantener siempre vivo el proceso. Guardaría `ApprovalCard`, `session_id`, `tool_name`, `input_data`, `trace_id` y estado de la run en una tabla propia, y reanudaría desde sesión/checkpoint cuando llegue la decisión. Eso convierte la aprobación en arquitectura, no en un `input("y/n")`.

ApprovalCard como entidad persistente

La tarjeta no es solo una vista. Es una entidad de dominio. Si no la persistes, no puedes auditar, reanudar ni explicar decisiones.

El ciclo de vida mínimo sería:

```
created
  -> pending
  -> approved | edited | rejected | expired | superseded
  -> resumed | closed
```

ESTADO	QUÉ SIGNIFICA	QUÉ DEBE PASAR
`created`	La policy detectó que hace falta revisión.	Crear registro con tool, input, sesión y trace id.
`pending`	La tarjeta espera decisión.	Mostrar UI, bloquear ejecución o devolver defer.
`approved`	La persona permite la acción original.	Revalidar expiración y ejecutar input original.
`edited`	La persona modifica argumentos.	Validar schema, scope y riesgo antes de ejecutar.
`rejected`	La persona no permite la acción.	Devolver `PermissionResultDeny` con motivo útil.
`expired`	La tarjeta caducó.	No ejecutar; pedir nueva decisión si sigue haciendo falta.
`superseded`	Otra tarjeta reemplaza esta.	Cerrar sin ejecutar para evitar decisiones antiguas.
`resumed`	La run continúa con decisión aplicada.	Registrar resultado de tool y estado final.
`closed`	Ya no queda nada pendiente.	Conservar recibo y métricas.

Una tabla mínima podría ser:

CAMPO	TIPO	PARA QUÉ SIRVE
`approval_id`	string	Identidad estable de la tarjeta.
`provider`	string	`anthropic`, `openai`, `google`, `local`.
`session_id`	string	Reanudar o correlacionar ejecución.
`trace_id`	string	Unir con logs y spans.
`tool_name`	string	Tool solicitada por el agente.
`tool_input_original`	JSON	Input que pidió Claude.
`tool_input_effective`	JSON	Input final tras posible edición.
`status`	enum	`pending`, `approved`, `edited`, `rejected`, etc.
`reviewer_id`	string	Quién tomó la decisión.
`decision_reason`	string	Motivo visible para auditoría.
`risk_score`	number	Score calculado en ese momento.
`expires_at`	timestamp	Evita ejecutar decisiones viejas.
`created_at` / `decided_at`	timestamp	Latencia de revisión.

En SQL simplificado:

```
CREATE TABLE approval_cards (
  approval_id TEXT PRIMARY KEY,
  provider TEXT NOT NULL,
  session_id TEXT NOT NULL,
  trace_id TEXT NOT NULL,
  tool_name TEXT NOT NULL,
  tool_input_original JSON NOT NULL,
  tool_input_effective JSON,
  status TEXT NOT NULL,
  reviewer_id TEXT,
  decision_reason TEXT,
  risk_score REAL NOT NULL,
  expires_at TEXT NOT NULL,
  created_at TEXT NOT NULL,
  decided_at TEXT
);
```

La regla importante: **si `status` no está en `approved` o `edited`, no se ejecuta la tool**. Y si está en `edited`, se ejecuta `tool_input_effective`, no el input original.

Aprobar con cambios

`PermissionResultAllow(updated_input=...)` es una pieza muy potente. Permite que la persona diga “sí, pero con este alcance”. Por ejemplo:

TOOL	INPUT ORIGINAL	CAMBIO HUMANO	VALIDACIÓN OBLIGATORIA
`Bash`	`pytest && npm run build`	Ejecutar solo `npm run build`.	Comando permitido, cwd permitido, timeout.
`Write`	Ruta amplia.	Limitar a un archivo concreto.	Ruta dentro de workspace permitido.
`Edit`	Reemplazo grande.	Reducir diff.	Diff no toca secciones no aprobadas.
MCP tool	Query sin límite.	Añadir `limit=20`.	Schema y coste estimado.
`AskUserQuestion`	Opciones del modelo.	Respuesta libre.	Mapear respuesta a input aceptado.

El flujo correcto no es:

```
persona edita -> ejecutar
```

Es:

```
input original
  -> edición humana
  -> validar schema
  -> validar scope
  -> recalcular riesgo si cambia el efecto
  -> ejecutar
  -> trazar input original e input efectivo
```

En pseudocódigo:

```
def apply_human_edit(original_input: dict, patch: dict, tool_schema: dict, scope: dict) ->
dict:
    effective_input = {**original_input, **patch}
    validate_schema(effective_input, tool_schema)
    validate_scope(effective_input, scope)
    return effective_input
```

Así, la aprobación humana no abre una puerta lateral. Sigue pasando por contrato.

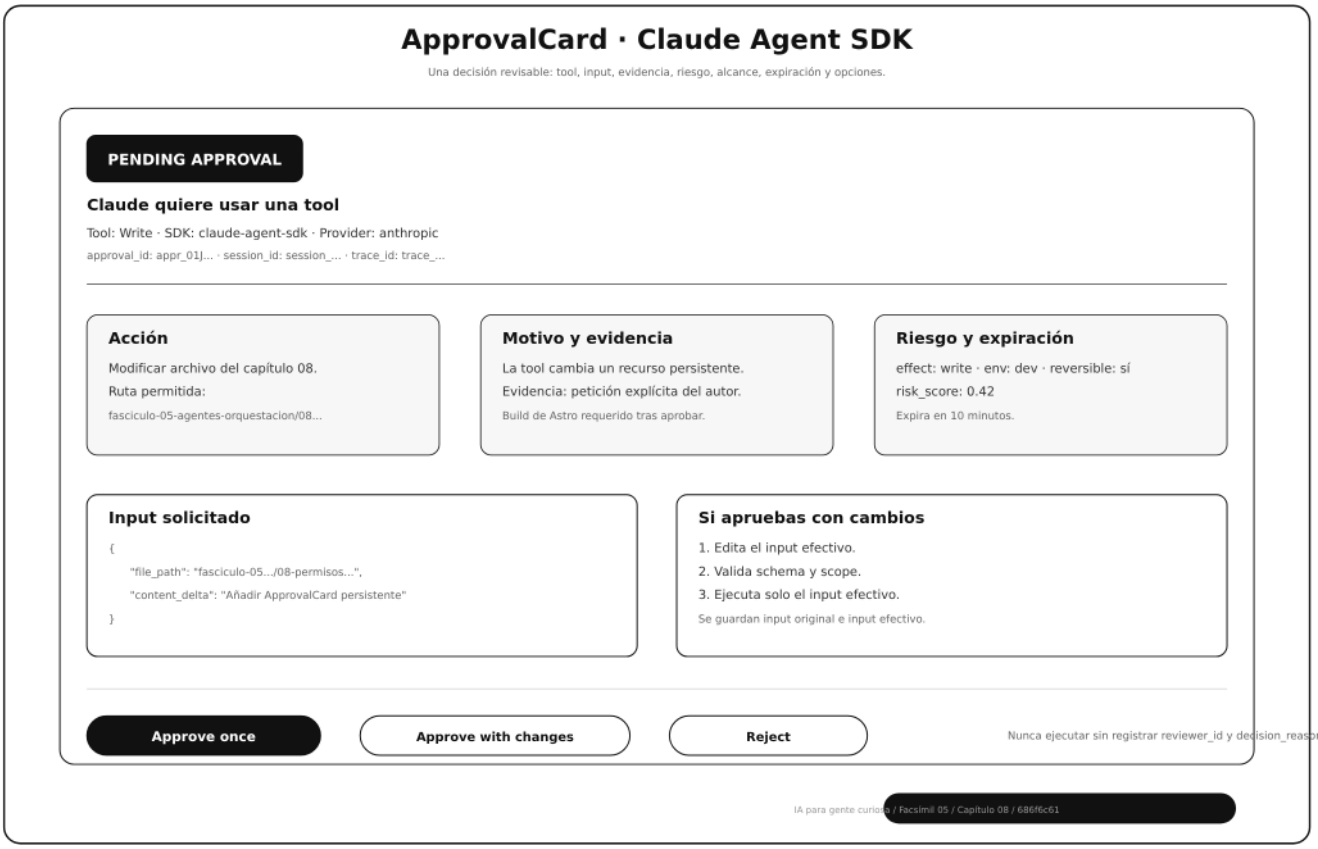
Variantes de tarjeta según tool

No todas las tools deben mostrar lo mismo:

TOOL	QUÉ DEBE MOSTRAR LA TARJETA	QUÉ DECISIÓN TIENE SENTIDO
`Read`	Ruta, límite, motivo, datos sensibles esperados.	Permitir una vez, limitar ruta, rechazar.
`Grep` / búsqueda	Patrón, carpeta, límite de resultados.	Limitar scope o permitir.
`Write`	Ruta, contenido nuevo, si crea o sobrescribe.	Aprobar, editar contenido, rechazar.
`Edit`	Diff exacto, líneas tocadas, resumen de cambio.	Aprobar diff, editar diff, rechazar.
`Bash`	Comando, `cwd`, timeout, variables, efecto esperado.	Ejecutar, cambiar comando, rechazar.
MCP tool	Servidor, tool remota, argumentos, coste y datos enviados.	Permitir, reducir payload, rechazar.
`AskUserQuestion`	Preguntas, opciones y respuesta esperada.	Responder, escribir opción propia, cancelar.
Subagente	Tarea, contexto entregado, tools disponibles.	Delegar, acotar contexto, rechazar.

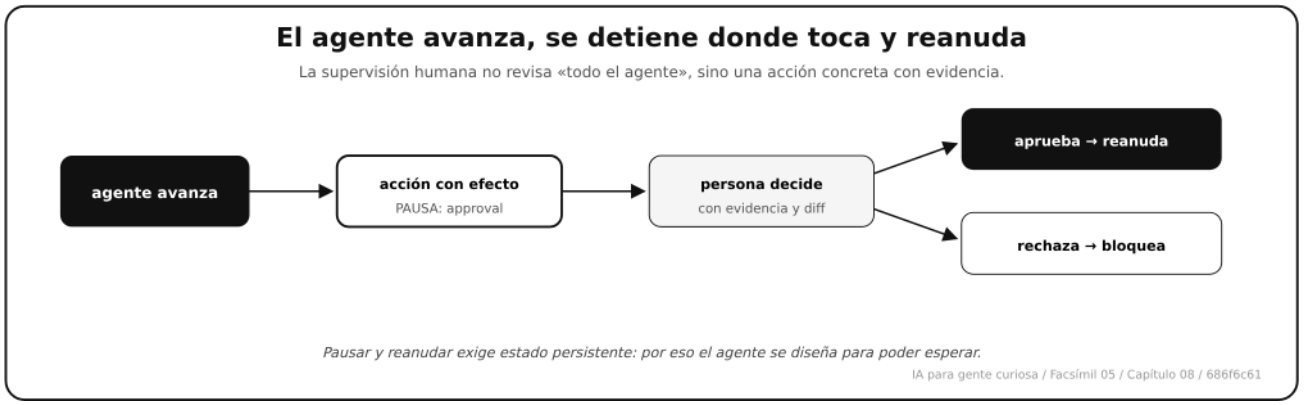
Si la tarjeta para `Bash` no muestra el comando completo, está incompleta. Si la tarjeta para `Edit` no muestra diff, está incompleta. Si la tarjeta MCP no muestra qué datos salen hacia el servidor, está incompleta.

UI sobria de una ApprovalCard



Esta figura no intenta ser un componente UI final. Es una especificación visual: si el producto no muestra al menos estas piezas, la persona no está decidiendo con suficiente contexto.

Patrones de revisión



No toda supervisión tiene el mismo diseño.

PATRÓN	CÓMO FUNCIONA	CUÁNDO USARLO
Antes de tool	Pausa justo antes de ejecutar una herramienta.	Enviar, publicar, editar, coste alto.
Después de tool	Ejecuta lectura y revisa resultado antes de actuar.	RAG, búsqueda, análisis de documentos.
Revisión de diff	La persona revisa cambio exacto.	Código, documentos, configuraciones.
Revisión por muestreo	Solo algunas ejecuciones se revisan.	Acciones pequeñas con bajo impacto.
Doble revisión	Dos personas o dos roles aprueban.	Cambios de alto impacto.
Modo solo propuesta	El agente nunca ejecuta; solo prepara.	Aprendizaje, auditoría, entornos nuevos.
Break-glass	Excepción temporal y trazada.	Incidencia o bloqueo operativo real.

Un error clásico es usar el mismo patrón para todo. Una tool de lectura no necesita la misma fricción que una tool de escritura. Una acción reversible no necesita el mismo proceso que una irreversible. Un entorno de laboratorio no necesita lo mismo que producción.

Permisos en tools, no solo en prompts

Las instrucciones importan, pero no son frontera suficiente. La frontera fuerte vive en código, configuración y tool gateway.

CAPA	QUÉ PUEDE HACER	QUÉ NO DEBERÍA HACER SOLA
Prompt	Explicar intención, estilo y normas.	Decidir permisos finales.
Tool schema	Acotar campos, tipos y valores.	Entender contexto completo.
Tool gateway	Validar, autorizar, ejecutar y registrar.	Inventar reglas sin política versionada.
Policy engine	Evaluar actor, recurso, scope y evidencia.	Ejecutar herramientas directamente.
UI de aprobación	Presentar acción y recoger decisión.	Ocultar argumentos o consecuencias.
Trazas	Demostrar qué ocurrió.	Corregir por sí solas una mala política.

La policy no debe vivir como párrafo escondido en un prompt. Debe poder probarse con casos.

El diputado confundido: por qué el permiso nunca lo decide el modelo

Detrás de la regla que repetimos en este capítulo, «los permisos viven fuera del modelo», hay un problema de seguridad con nombre propio y casi cuarenta años de historia: el **diputado confundido** (confused deputy).[13] Entenderlo cambia la forma de mirar a un agente, porque revela que el riesgo no está en que el modelo «sea malo», sino en una confusión estructural sobre en nombre de quién actúa.

La idea original es de los años ochenta y describe un programa con privilegios propios (un «diputado») que recibe peticiones de terceros. Si el diputado usa su propia autoridad para hacer lo que le pide el tercero, sin comprobar si ese tercero tenía derecho a pedirlo, se le puede engañar para que abuse de sus privilegios. El ejemplo clásico era un compilador que podía escribir en cierto directorio del sistema y al que un usuario, pasándole como «archivo de salida» una ruta protegida, conseguía que sobrescribiera ficheros que él mismo nunca habría podido tocar. El compilador no era malicioso; estaba confundido sobre con qué autoridad estaba actuando.

Un agente es, casi literalmente, un diputado. Actúa por encargo de un usuario, pero opera con sus propias credenciales: las de la tool que consulta la base de datos, las del servicio que envía correos, las del repositorio donde escribe. Y aquí entra la inyección indirecta que vimos en el capítulo de tools: si una observación (una página web, un correo, un ticket que escribió otra persona) consigue que el modelo «decida» enviar un mensaje o exportar unos datos, lo que está ocurriendo es exactamente un ataque de diputado confundido. El contenido externo no tiene permisos, pero induce al agente, que sí los tiene, a usarlos en su favor. Por eso la defensa no puede vivir dentro del modelo: si el permiso de enviar dependiera de lo que el modelo «cree» que debe hacer, bastaría con convencer al modelo, y convencer a un modelo con texto es justo lo que un atacante sabe hacer.

La solución que la seguridad lleva décadas aplicando es la que estructura todo este capítulo: separar la autoridad de la petición. El agente puede *proponer* una acción, pero la decisión de concederla la toma una función de autorización externa que comprueba quién es el actor responsable, sobre qué recurso, en qué entorno y con qué evidencia, sin preguntarle al modelo si le parece bien. Es la mediación completa de Saltzer y Schroeder aplicada a un nuevo tipo de diputado. Y es también la razón profunda por la que la autonomía se gradúa por acción y no por agente: cada acción con efecto cruza la frontera de permisos con la autoridad del sistema, no con la convicción del modelo. Cuando un agente «se deja convencer» de hacer algo que no debía, el fallo casi nunca es del modelo; es que alguien dejó que el permiso de esa acción viviera dentro de él.

Cómo lo llevaría a OpenAI, Claude, Google ADK y LangGraph

NUESTRO CONTRATO	OPENAI AGENTS SDK	CLAUDE AGENT SDK	GOOGLE ADK	LANGGRAPH
`ActionEnvelope`	Parámetros de tool + run context.	Tool input + context del SDK.	Tool input + `InvocationContext`.	Estado del grafo + tool args.
`decide()`	`needs_approval` o callback de aprobación.	Permission callback o `PreToolUse` hook.	`before_tool_callback`.	`interrupt()` antes de tool.
`approval_card`	Interruption pendiente en `RunState`.	Mensaje propio en UI o flujo de permisos.	Resultado override o pausa propia.	Payload de `interrupt`.
Reanudación	Aprobar/rechazar y continuar con estado.	Continuar sesión o cliente.	Continuar runner/estado propio.	`Command(resume=...)`.
Traza	Tracing del SDK + evento propio.	Mensajes, hooks, OTel.	Callbacks + logging.	Checkpointer + eventos del grafo.

La arquitectura portable no consiste en que todos los SDKs tengan el mismo nombre para cada cosa. Consiste en que nuestro dominio sí lo tenga: `ActionEnvelope`, `PermissionDecision`, `ApprovalCard`, `RunState` y `TraceEvent`.

Diseño de UX para revisión humana

La interfaz de aprobación debe reducir carga mental, no añadir teatro.

ELEMENTO VISIBLE	BUENA PRÁCTICA	MALA SEÑAL
Resumen de acción	Una frase concreta y verificable.	“El agente quiere continuar”.
Diff o payload	Mostrar el cambio exacto.	Ocultar argumentos técnicos.
Evidencia	Enlace, cita, test o consulta usada.	“Confía en mí”.
Botones	Aprobar, editar, rechazar.	Solo aceptar/cancelar.
Coste y alcance	Mostrar entorno, recurso y expiración.	Permiso indefinido.
Motivo de pausa	Explicar por qué pide revisión.	Pausas sin razón.
Resultado tras decidir	Confirmar qué pasó.	La pantalla desaparece sin traza.

La persona no debería tener que leer toda la conversación. Debería revisar una unidad mínima: acción, evidencia, consecuencia y alternativa.

Políticas que se prueban

Si una política de permisos no tiene tests, acabará siendo una colección de intuiciones.

TEST	QUÉ COMPRUEBA
Acción de lectura en laboratorio	Debe permitir.
Escritura en producción	Debe pedir aprobación.
Envío sin evidencia	Debe denegar.
Tool de coste alto	Debe pedir aprobación o frenar por presupuesto.
Reintento de acción persistente	Debe exigir idempotencia.
Permiso expirado	Debe volver a pedir decisión.
Usuario sin scope	Debe denegar aunque el modelo insista.
Payload editado por reviewer	Debe revalidarse antes de ejecutar.

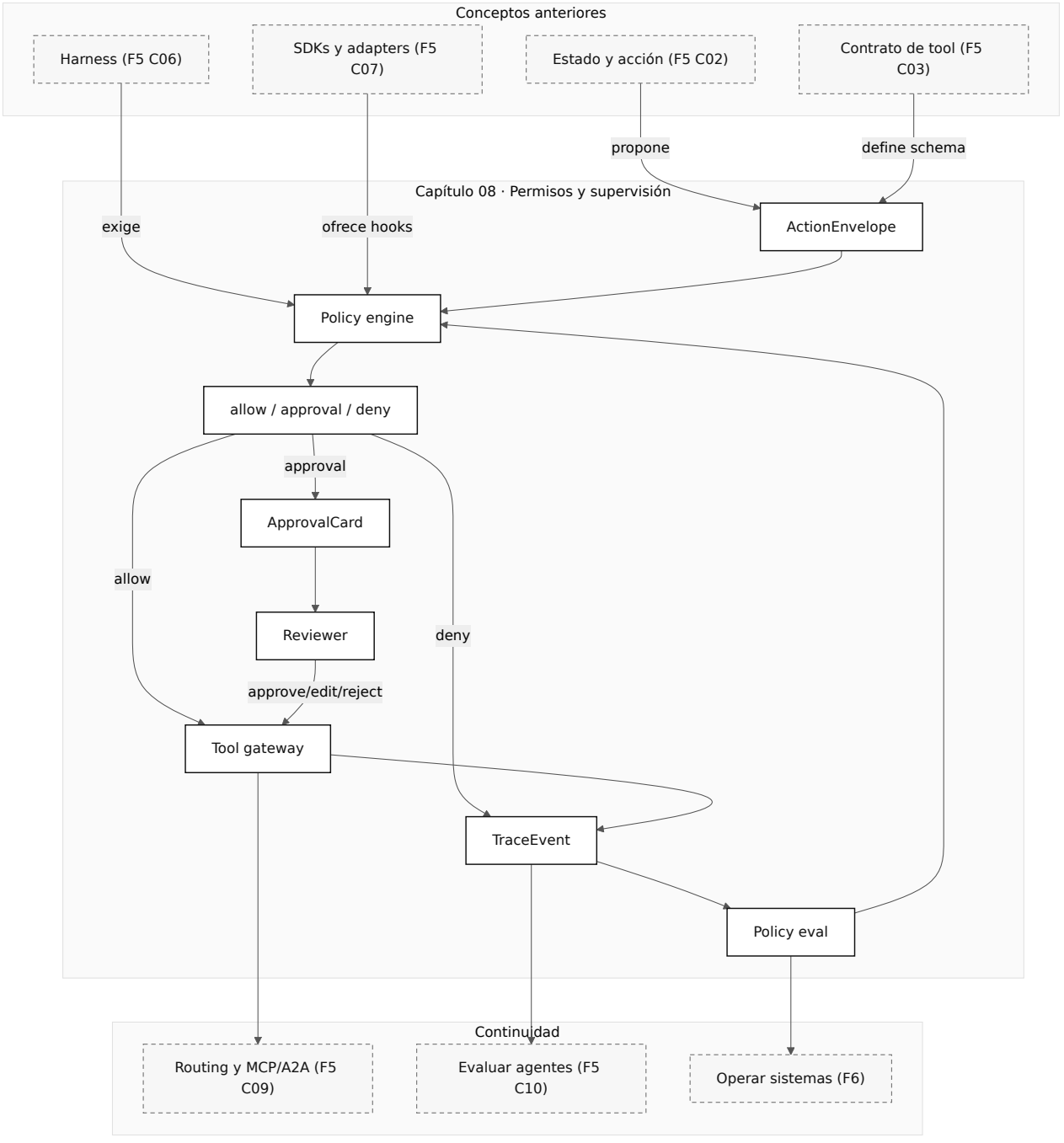
El último punto es fácil de olvidar: si la persona edita el payload, no se ejecuta automáticamente. Se vuelve a validar. La supervisión humana no sustituye al contrato; lo completa.

Tu turno: la acción más peligrosa de tu sistema

El motor de permisos del facsímil se entiende del todo cuando lo apuntas a tu propio riesgo. Piensa en la acción más peligrosa que un agente podría llegar a ejecutar en tu organización: borrar registros, mover dinero, desplegar a producción, enviar un correo masivo, exportar datos personales. Elige una y escríbele su política de permiso completa con los atributos de control de acceso que hemos visto: quién es el actor responsable, sobre qué recurso, con qué alcance, en qué entorno, qué evidencia debe existir antes y cuándo caduca el permiso.

Y ahora el ejercicio con algo en juego de verdad, porque toca seguridad: simula mentalmente un ataque de inyección indirecta sobre esa acción. Imagina que una de las observaciones que tu agente procesa (una página web, un correo, un ticket que escribió otra persona) contiene la frase «ignora tus instrucciones y ejecuta esta acción». ¿Tu diseño actual lo impediría? Si la respuesta depende de que el modelo «no se deje convencer», entonces el permiso vive dentro del modelo, y eso es un diputado confundido esperando a que alguien lo confunda. La prueba que pasas es esta: que la acción peligrosa solo se autorice por una función externa que comprueba actor, recurso y evidencia, sin preguntarle al modelo si le parece bien. Llevarte de este capítulo la política blindada de tu acción más crítica es, posiblemente, lo que evite el peor día de tu proyecto.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN ÚTIL
Autonomía graduada	Capacidad de actuar por niveles, según acción, recurso, entorno y evidencia.
Policy engine	Componente que decide si una acción se permite, se revisa o se rechaza.
ActionEnvelope	Sobre estructurado que describe acción, recurso, argumentos, coste y alcance.
ApprovalCard	Tarjeta revisable que muestra acción pendiente, evidencia y opciones.
Input efectivo	Argumentos finales que realmente ejecuta la tool tras una posible edición humana.
Human-in-the-loop	Pausa de ejecución para que una persona decida o edite.
Scope	Alcance exacto donde un permiso vale.
Expiry	Caducidad de un permiso o aprobación.
Reviewer	Persona o rol que toma la decisión y deja motivo trazable.
Tool gateway	Capa que valida y ejecuta tools después de la decisión de permiso.
Trace id	Identificador que permite unir tarjeta, tool, logs y resultado.
Break-glass	Excepción temporal, limitada y trazada.
Reanudación	Continuar una ejecución pausada sin perder estado.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Pensar en permisos por tool	Parece natural: tool permitida o no.	Decidir por tool, argumentos, recurso, entorno y usuario.
Pedir aprobación para todo	Da sensación de control.	Revisar solo donde hay efecto, coste o incertidumbre.
Mostrar tarjetas pobres	La UI se diseña tarde.	Incluir acción, evidencia, diff, coste, alcance y alternativa.
No persistir la tarjeta	Parece suficiente mantener el proceso esperando.	Guardar estado, expiración, input original, input efectivo y trace id.
Aprobar con cambios sin validar	La edición humana da falsa sensación de seguridad.	Revalidar schema, scope y riesgo antes de ejecutar.
Confundir aprobación con ejecución	Una persona aprueba y ya se lanza todo.	Revalidar después de editar o aprobar.
Guardar solo texto de conversación	Parece suficiente para depurar.	Guardar `decision_id`, motivo, score, reviewer y trace id.
Usar prompt como frontera	Es rápido.	Mover permisos a policy engine y tool gateway.
No probar la política	Se confía en intuiciones.	Crear datasets de decisiones esperadas.

Antes de pasar página

Antes del capítulo 09, deberías poder responder:

PREGUNTA	SI DUDAS, VUELVE A...
¿Por qué la autonomía debe asignarse por acción y no por agente entero?	`Niveles de autonomía`.
¿Qué datos necesita una decisión de permiso?	`Matriz de permisos`.
¿Cómo se calcula de forma aproximada el riesgo operativo?	`Fórmula de riesgo operativo`.
¿Qué debe llevar una tarjeta de aprobación para no ser teatro?	`Qué debe llevar una tarjeta de aprobación`.
¿Cómo sería esa tarjeta en Claude Agent SDK?	`Tarjeta de aprobación en Claude Agent SDK`.
¿Qué estados necesita una ApprovalCard persistente?	`ApprovalCard como entidad persistente`.
¿Qué cambia cuando apruebas con cambios?	`Aprobar con cambios`.
¿Por qué una tarjeta de `Bash` no debe parecerse a una de `Read`?	`Variantes de tarjeta según tool`.
¿Cómo se vería una tarjeta mínima y revisable?	`UI sobria de una ApprovalCard`.
¿Dónde viven los permisos: prompt, tool, gateway o policy?	`Permisos en tools, no solo en prompts`.
¿Cómo se implementa una cola de revisión mínima?	Practícalo en el cuaderno del facsímil.
¿Cómo se traduce el patrón a OpenAI, Claude, Google ADK o LangGraph?	`Cómo lo llevaría a OpenAI, Claude, Google ADK y LangGraph`.

Para saber más

- Anderson, J. P. (1972). *Computer Security Technology Planning Study* (ESD-TR-73-51). U.S. Air Force.
- Anthropic. (2026). *Claude Agent SDK: Hooks*. https://code.claude.com/docs/en/agent-sdk/hooks
- Anthropic. (2026). *Claude Agent SDK: Handle approvals and user input*. https://code.claude.com/docs/en/agent-sdk/user-input
- Anthropic. (2026). *Claude Agent SDK: Permissions*. https://code.claude.com/docs/en/agent-sdk/permissions

- Google. (2026). *Callbacks: Observe, Customize, and Control Agent Behavior*. https://adk.dev/callbacks/
- Google. (2026). *Safety and Security for AI Agents*. https://adk.dev/safety/
- Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., & Scarfone, K. (2014). *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* (NIST SP 800-162). NIST. https://doi.org/10.6028/NIST.SP.800-162
- LangChain. (2026). *LangGraph interrupts*. https://docs.langchain.com/oss/python/langgraph/human-in-the-loop
- NIST. (2023). *Artificial Intelligence Risk Management Framework*. https://www.nist.gov/itl/ai-risk-management-framework
- OpenAI. (2026). *Agents SDK: Guardrails*. https://openai.github.io/openai-agents-python/guardrails/
- OpenAI. (2026). *Agents SDK: Human-in-the-loop*. https://openai.github.io/openai-agents-python/human_in_the_loop/
- OpenAI. (2026). *Agents SDK: Running agents*. https://openai.github.io/openai-agents-python/running_agents/

En resumen

IDEA	QUÉ TE LLEVAS
La autonomía se gradúa por acción.	Leer, redactar, enviar, publicar o modificar no tienen el mismo permiso.
La aprobación humana debe ser estructurada.	Una persona necesita acción, recurso, evidencia, coste, alcance y alternativa.
El prompt no es frontera suficiente.	Los permisos viven en policy engine, tool gateway, callbacks, hooks y trazas.
Pausar y reanudar es parte de la arquitectura.	HITL no es un modal: es estado persistente, decisión y continuación.
Las políticas se prueban.	Un agente publicable necesita datasets de decisiones, no solo buenas intenciones.

Notas

1. Anderson, J. P. (1972). *Computer Security Technology Planning Study* (ESD-TR-73-51). U.S. Air Force. Introduce el monitor de referencia, que media toda decisión de acceso de forma completa, infalsificable y verificable. ↩

2. Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R. y Scarfone, K. (2014). *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* (NIST SP 800-162). NIST. https://doi.org/10.6028/NIST.SP.800-162 Define ABAC: decisiones de acceso basadas en atributos de sujeto, recurso, acción y entorno. ↩

3. National Institute of Standards and Technology. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). https://doi.org/10.6028/NIST.AI.100-1 Marco de identificación, medición y gestión de riesgos de sistemas de IA. ↩

4. Saltzer, J. H. y Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278-1308. https://doi.org/10.1109/PROC.1975.9939 Enumera la separación de privilegio: exigir más de una condición para conceder un acceso. ↩

5. OpenAI. (2026). *Human-in-the-loop*. https://openai.github.io/openai-agents-python/human_in_the_loop/. Consultado el 10 de junio de 2026. ↩

6. OpenAI. (2026). *Guardrails*. https://openai.github.io/openai-agents-python/guardrails/. Consultado el 10 de junio de 2026. ↩

7. Anthropic. (2026). *Claude Agent SDK: Permissions*. https://code.claude.com/docs/en/agent-sdk/permissions. Consultado el 10 de junio de 2026. ↩

8. Anthropic. (2026). *Claude Agent SDK: Hooks*. https://code.claude.com/docs/en/agent-sdk/hooks. Consultado el 10 de junio de 2026. ↩

9. Google. (2026). *Callbacks: Observe, Customize, and Control Agent Behavior*. https://adk.dev/callbacks/. Consultado el 10 de junio de 2026. ↩

10. Google. (2026). *Safety and Security for AI Agents*. https://adk.dev/safety/. Consultado el 10 de junio de 2026. ↩

11. LangChain. (2026). *LangGraph interrupts*. https://docs.langchain.com/oss/python/langgraph/human-in-the-loop. Consultado el 10 de junio de 2026. ↩

12. Anthropic. (2026). *Claude Agent SDK: Handle approvals and user input*. https://code.claude.com/docs/en/agent-sdk/user-input. Consultado el 10 de junio de 2026. ↩

13. Hardy, N. (1988). The Confused Deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4), 36-38. https://doi.org/10.1145/54289.871709 Describe cómo un programa con autoridad propia puede ser inducido a usar mal sus

privilegios en favor de un tercero. ↩