GITHUB COPILOT AGENTIC FRAMEWORK

Jerarquía de configuración: cómo los archivos de config moldean sus agentes.

Cinco capas en cascada, desde copilot-instructions.md hasta los prompt files. Un briefing técnico sobre cómo GitHub Copilot ensambla el contexto, cómo funciona la precedencia y cómo diseñar una stack de configuración que todo el equipo hereda.

AUTORA
Paula Silva

ROL
Software Global Black Belt

DURACIÓN
45 a 60 minutos

FECHA
2026-06-10

AGENDA

Cinco partes, un modelo mental.

PARTE I	Fundamento, por qué la configuración importa y el modelo mental de la cascada
PARTE II	Las cinco capas, archivo por archivo, con contenido real para copiar hoy
PARTE III	Resolución, el algoritmo que GitHub Copilot ejecuta y las reglas de precedencia
PARTE IV	Práctica, la economía de tokens, el árbol completo del proyecto, escenarios enterprise y la lista de qué hacer o no
PARTE V	Memoria, cómo GitHub Copilot aprende el repo solo y cuándo graduarla a archivos

PARTE

I

El fundamento.

Los asistentes de IA modernos leen una jerarquía de archivos para ensamblar el contexto detrás de cada respuesta.

POR QUÉ LA CONFIGURACIÓN IMPORTA

La configuración decide qué sabe la IA, qué puede hacer y cómo se comporta.

DIMENSIÓN 01 · CONOCIMIENTO

Qué sabe la IA

Instrucciones del proyecto, convenciones de código, patrones de arquitectura y los comandos clave de su repositorio.

DIMENSIÓN 02 · PERMISO

Qué puede hacer la IA

Herramientas permitidas, límites de acceso y los guardrails de seguridad que mantienen a los agentes dentro de las líneas.

DIMENSIÓN 03 · COMPORTAMIENTO

Cómo se comporta la IA

Reglas por alcance, la personalidad de agentes especializados, tono y formato de las respuestas.

SIN LA JERARQUÍA

Los equipos terminan con instrucciones duplicadas, reglas en conflicto, o una IA que ignora convenciones críticas porque el archivo correcto nunca fue creado.

EL MODELO MENTAL

Piense como CSS: una cascada de especificidad.

CASCADA CSS

`body { }`
Estilo global, aplica en todo

`.container { }`
Nivel de componente, alcance menor

`#hero { }`
El más específico gana el conflicto

CASCADA DE CONFIG

`copilot-instructions.md`
Contexto global del repositorio

`.instructions.md`
Alcance por carpeta o patrón de archivo

`*.agent.md · SKILL.md`
El más específico complementa al resto

REGLA DE ORO

Una capa hija AGREGA, nunca contradice. Si el archivo global dice "use TypeScript", un archivo de alcance no puede decir "use JavaScript". La especificidad complementa, nunca sobrescribe.

PARTE

II

Las cinco capas.

Inmersión profunda en cada tipo de archivo, con contenido real, ubicación real y la regla de activación de cada uno.

CAPA 1 DE 5 · SIEMPRE ACTIVA

copilot-instructions.md es el cerebro global del repositorio.

.GITHUB/COPILOT-INSTRUCTIONS.MD

```
# Project: E-Commerce Platform

## Tech Stack
- Next.js 14 with App Router
- TypeScript strict mode
- Prisma ORM with PostgreSQL

## Conventions
- kebab-case for file names
- PascalCase for React components
- All API routes in app/api/
- Tests: Vitest, run with npm test

## Architecture
- Feature-based folder structure
- Server components by default
```

UBICACIÓN

`.github/copilot-instructions.md`

ACTIVACIÓN

Automática en toda sesión de GitHub Copilot. Siempre activa, sin frontmatter.

POR QUÉ IMPORTA

El archivo más impactante que puede crear. Define el tono de toda interacción con su proyecto.

CAPA 2 DE 5 · ALCANCE POR GLOB

`.instructions.md` aplica reglas solo donde el glob hace match.

UBICACIÓN

`.github/instructions/*.instructions.md`

ACTIVACIÓN

El frontmatter applyTo lleva un glob pattern. Las reglas cargan solo cuando un archivo en contexto hace match.

ÚSELO CUANDO

Partes distintas del proyecto piden reglas distintas: reglas React para componentes, reglas de API para rutas, reglas de test para specs.

.GITHUB/INSTRUCTIONS/REACT-COMPONENTS.INSTRUCTIONS.MD

```
---
applyTo: "src/components/**/*.tsx"
---

# React Component Rules

- Functional components with hooks
- TypeScript interfaces for all props
- JSDoc comments on public props
- Tailwind for styling, no CSS modules
- Unit tests in __tests__/ folder
- Components must stay under 200 lines
```

CAPA 3 DE 5 · ACTIVA AL INVOCARSE

*.agent.md define personas especializadas con herramientas propias.

.GITHUB/AGENTS/SECURITY-REVIEWER.AGENT.MD

```
---
description: "Security code reviewer"
tools: [codeql, semgrep, gh-advanced-security]
---

# Security Reviewer Agent

You are a security code reviewer. Analyze:
- SQL injection vulnerabilities
- XSS attack vectors
- Authentication bypass risks

Always provide:
- Severity (Critical/High/Medium/Low)
- OWASP category reference
- Remediation code example
```

IDENTIDAD

Un nombre, una descripción y un comportamiento único que el modelo adopta cuando el agente está activo.

HERRAMIENTAS

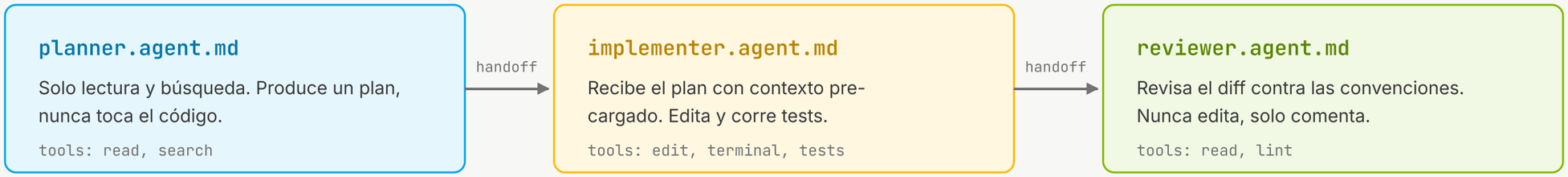
Una allowlist explícita de herramientas que el agente puede llamar. Todo lo demás queda fuera.

ALCANCE

Foco en un área o tipo de tarea. El agente activo estrecha el contexto a su propio dominio.

CAPA 3 EXTENDIDA · HANDOFFS

Los agentes se encadenan: plan, implement, review como pipeline.



Al final de cada agente, un botón de handoff transfiere el contexto al siguiente. El flujo se vuelve proceso, no improvisación.

Los custom agents (.agent.md) reemplazan a los antiguos chat modes: persona, allowlist de herramientas y handoffs encadenados. En planes Business y Enterprise, las definiciones pueden compartirse a nivel de organización: todo repo hereda los mismos especialistas.

CAPA 4 DE 5 · GLOB O ALWAYSAPPLY

SKILL.md empaqueta un dominio completo, no solo reglas.

UBICACIÓN

.github/skills/<name>/SKILL.md

ANATOMÍA

Reglas más workflows más plantillas más checklists de calidad. La estructura fuerza la completitud.

SKILLS VS INSTRUCTIONS

Las instructions son reglas simples con alcance. Las skills son paquetes completos de dominio. Use una skill cuando el dominio es complejo.

.GITHUB/SKILLS/API-DESIGN/SKILL.MD

```
---
name: "REST API Design"
globs: ["src/api/**/*.ts", "src/routes/**/*.ts"]
alwaysApply: false
---

# REST API Design Skill

## Rules
- Plural nouns: /users, not /user
- Version APIs: /api/v1/resources
- Cursor-based pagination

## Workflow
1. Define resource schema  2. Route handlers
3. Validation middleware   4. Integration tests

## Quality checklist
- [ ] Proper status codes on all endpoints
- [ ] Input validation on POST and PUT
- [ ] OpenAPI docs generated
```

CAPA 4 EXTENDIDA · CARGA PROGRESIVA

Las skills cargan en tres etapas. El contexto solo paga por lo que usa.

ETAPA 1 · SIEMPRE

Solo name y description

El índice de todas las skills queda en el contexto. Costo: decenas de tokens.

`~40 tokens / skill`

match →

ETAPA 2 · RELEVANTE

SKILL.md completo

Las instrucciones completas entran solo cuando la tarea coincide con la skill.

`~1.200 tokens`

bajo demanda →

ETAPA 3 · EJECUCIÓN

Scripts, templates, ejemplos

Los recursos de la carpeta se leen o ejecutan en el momento del uso.

`cero hasta necesitarlo`

Es el mismo principio de los tres tiers de memoria: pague tokens por lo que la tarea necesita, no por lo que el repositorio tiene.

ESTÁNDAR ABIERTO, PORTABLE

`VS Code` `GitHub Copilot CLI` `coding agent` `~/.copilot/skills personal`

CAPA 5 DE 5 · BAJO DEMANDA

.prompt.md convierte pedidos repetidos en plantillas reutilizables.

.GITHUB/PROMPTS/CREATE-COMPONENT.PROMPT.MD

```
---
mode: "agent"
description: "Create a new React component"
variables:
  - name: "componentName"
  - name: "type"   # page, layout, widget
---

Create a React component called {{componentName}}.
Type: {{type}}

Steps:
1. Create the file in src/components/
2. Add the TypeScript props interface
3. Add a unit test in __tests__/
4. Export from the barrel index.ts
```

MODO AGENT

mode: "agent" permite a GitHub Copilot ejecutar acciones, crear archivos, correr tests. mode: "edit" solo sugiere cambios.

VARIABLES

Las llaves dobles marcan parámetros dinámicos. GitHub Copilot pide los valores antes de ejecutar.

CICLO DE VIDA

Efímero por diseño. Un prompt file corre una vez por invocación y no persiste en el contexto.

EL MAPA EN UNA VISTA

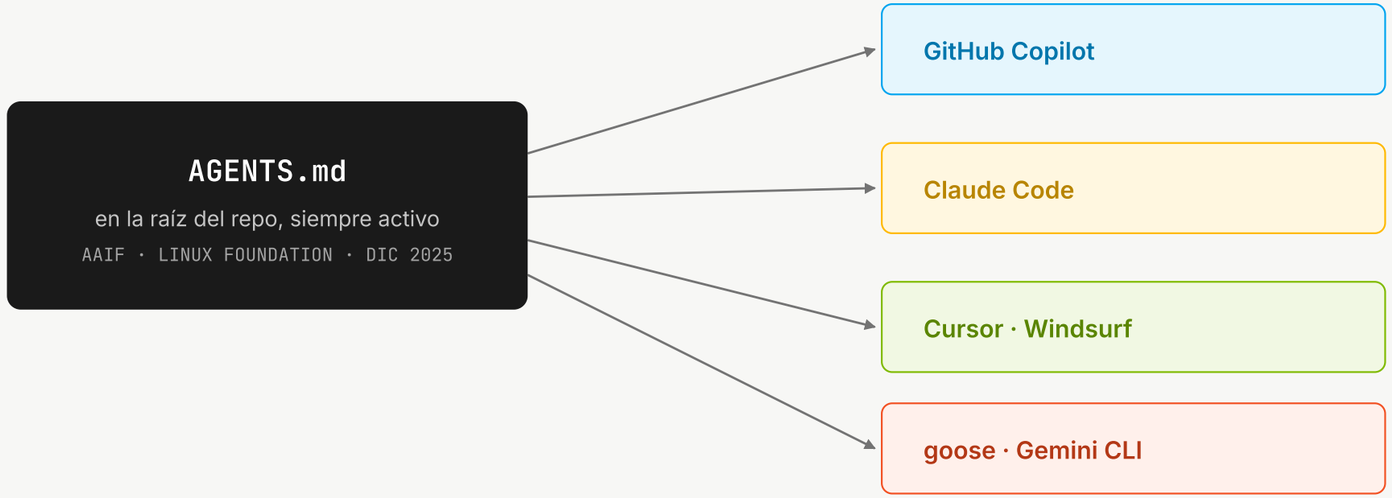
Siete tipos de archivo, un modelo de composición.

ARCHIVO	ALCANCE	ACTIVACIÓN	PROPÓSITO	FRONTMATTER
`copilot-instructions.md`	Repo entero	Automática	Convenciones globales	`ninguno`
`AGENTS.md`	Repo entero, multi-agente	Automática	Estándar abierto entre herramientas	`ninguno`
`.instructions.md`	Glob pattern	Por glob match	Reglas por área	`applyTo`
`*.agent.md`	Agente activo	Cuando se invoca	Persona y herramientas	`description, tools`
`SKILL.md`	Glob o always	Glob o alwaysApply	Dominio completo	`name, globs, alwaysApply`
`.prompt.md`	Invocación	Bajo demanda	Plantilla con variables	`mode, variables`
`.mcp.json`	Proyecto	Automática	Conexiones con herramientas externas	`JSON, sin frontmatter`

Valide nombres y frontmatter contra la documentación oficial de customización de GitHub Copilot. Las convenciones evolucionan rápido.

EL MAPA EN MOVIMIENTO · AGENTS.MD

AGENTS.md: un archivo, todos los agentes. Ahora estándar abierto.



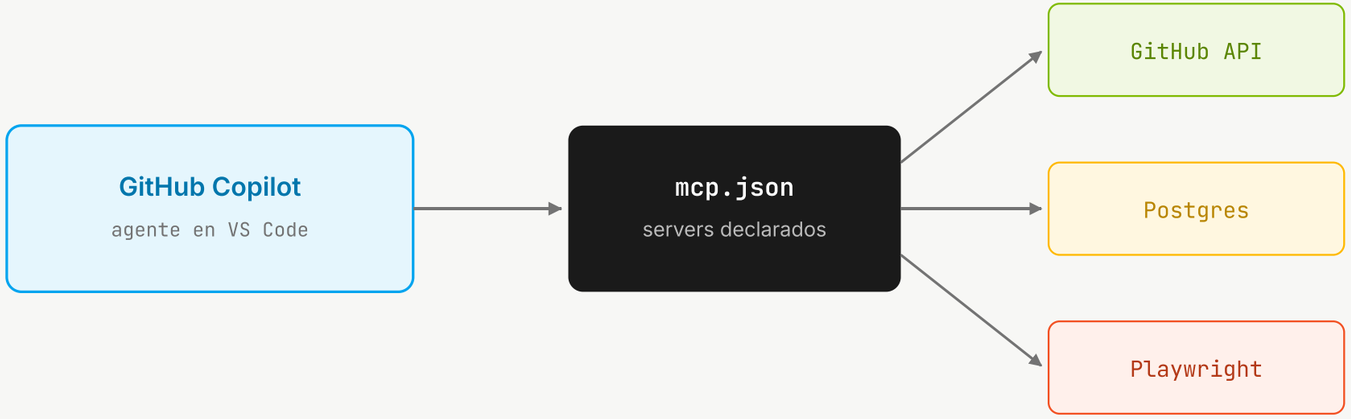
REGLA DE ORO

Lo que vale para cualquier agente, en AGENTS.md. Lo específico de GitHub Copilot, en copilot-instructions.md. Sin duplicar.

Donado por OpenAI a la Agentic AI Foundation (Linux Foundation) en dic 2025, junto con MCP y goose. VS Code lee AGENTS.md como instrucción siempre activa al lado del copilot-instructions.md: uno es el estándar abierto multi-herramienta, el otro es el canal específico de GitHub Copilot.

EL MAPA EN MOVIMIENTO · MCP

mcp.json conecta los agentes al mundo. Y se volvió infraestructura neutral.



.VSCODE/MCP.JSON

```jsonc
// servers que o agente pode usar
{
  "servers": {
    "github": {
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "postgres": {
      "command": "npx",
      "args": ["-y", "@mcp/postgres"]
    }
  }
}
```

97M descargas SDK / mes

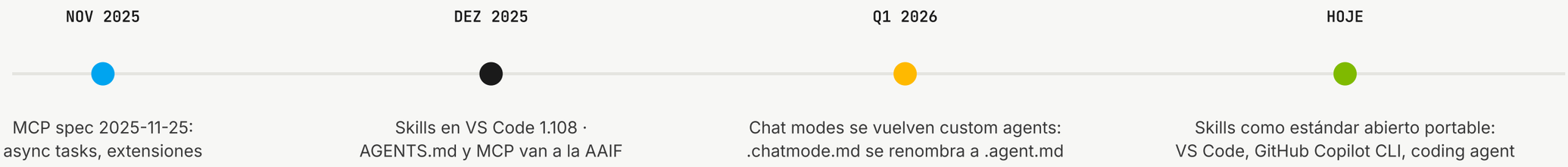
10.000+ servers publicados

spec 2025-11-25

Model Context Protocol: spec 2025-11-25 (async tasks, extensiones, elicitation) y gobernanza en la Agentic AI Foundation desde dic 2025. Para la jerarquía, la regla es la misma de las otras capas: declárelo en el repo, versiónelo en Git, y todo el equipo hereda las mismas conexiones.

EL MAPA EN MOVIMIENTO · ÚLTIMOS 6 MESES

Qué cambió desde diciembre. Valide antes de estandarizar.



La lección operativa: los nombres de archivo y el frontmatter cambian en ciclos de meses. Trate la doc oficial de customización como fuente de verdad, agende una revisión trimestral de la stack, y renombre .chatmode.md a .agent.md en el próximo ciclo.

PARTE

III

Resolución.

El algoritmo de siete pasos que GitHub Copilot ejecuta para ensamblar el contexto, y las reglas de precedencia que resuelven conflictos.

EL ORDEN DE RESOLUCIÓN

Siete pasos, cada uno AGREGA al contexto acumulado.

01	`copilot-instructions.md`	Cargar primero el contexto global del repo, en toda sesión: copilot-instructions.md y, cuando exista, AGENTS.md.
02	`archivos en contexto`	Identificar qué archivos forman parte de la conversación actual.
03	`*.instructions.md`	Aplicar todo instruction file cuyo glob hace match con esos archivos.
04	`*.agent.md`	Cargar la definición del agente activo, cuando uno se invoca.
05	`SKILL.md`	Activar skills por glob match o alwaysApply: true.
06	`*.prompt.md`	Expandir la plantilla de prompt con sus variables, cuando se invoca.
07	`.mcp.json`	Agregar las conexiones MCP disponibles al contexto final.

LA RESOLUCIÓN · CUATRO TRAMPAS

Cuatro comportamientos que nadie espera, y todo equipo descubre tarde.

CODE REVIEW · BRANCH BASE

El review lee las instrucciones de la branch base, no de la suya

En el pull request, el agente de code review usa el copilot-instructions.md de la base. Reglas nuevas en la feature branch solo valen después del merge. Es intencional: la branch no reescribe sus propias reglas de revisión.

COMPLETIONS · FUERA DEL SISTEMA

El ghost text no lee nada de esto

Las sugerencias inline mientras escribe son un sistema separado: no leen copilot-instructions.md ni instructions files. La jerarquía gobierna el chat y los agentes, no el autocompletado.

STACKING · SIN GANADOR

Las instructions se apilan, no compiten

Todo instructions file cuyo glob coincide entra junto, en unión. No existe override ni precedencia entre ellos. Si dos reglas chocan, ambas llegan al modelo. Resolver el conflicto es trabajo suyo, no de GitHub Copilot.

EXCLUDEAGENT · PUNTERÍA FINA

Se puede sacar a un agente de una regla

El frontmatter excludeAgent saca a un agente específico de un instructions file: la regla de seguridad perfecta en code review y demasiado verbosa en el chat queda solo donde ayuda.

Comportamientos documentados en la doc oficial de customización de VS Code y GitHub Copilot. Vale pegar este slide en el canal del equipo: cada ítem ya le costó a alguien una tarde de debugging.

REGLAS DE PRECEDENCIA

No existe override destructivo. La composición es siempre aditiva.

La especificidad gana	Las reglas más específicas complementan a las genéricas y prevalecen en la práctica cuando ambas tocan el mismo punto.	`.instructions.md > copilot-instructions.md`
Aditivo, no sustitutivo	Las capas se acumulan, nunca se reemplazan. El contexto final las lleva todas.	`regla de carpeta + regla global = ambas`
El agente aísla el alcance	Un agente activo estrecha el contexto a su propio dominio, herramientas e identidad.	`agente de seguridad se enfoca en vulnerabilidades`
El prompt es efímero	Los prompt files corren bajo demanda y no persisten. Una invocación, una expansión.	`la plantilla corre una vez por invocación`

PRUEBE EL APPLYTO EN VIVO

`src/api/payments.ts` `web/components/App.tsx` `README.md` → `copilot-instructions.md` `api.instructions.md src/api/**`

`frontend.instructions.md **/*.tsx` `docs.instructions.md **/*.md`

LA RESOLUCIÓN · ARME SU STACK INTERACTIVO

Encienda y apague capas. Vea lo que el agente recibe.

ESCENARIO: EDITAR SRC/API/PAYMENTS.TS Y PEDIR UN ENDPOINT NUEVO

- [x] **copilot-instructions.md** siempre activo · ~800 tokens
- [ ] **AGENTS.md** siempre activo, multi-agente · ~600
- [x] **api.instructions.md** applyTo: src/api/** coincide · ~450
- [ ] **security.agent.md** si se invoca · ~350
- [x] **api-design/SKILL.md** match de la tarea · ~1.200
- [ ] **new-endpoint.prompt.md** si se llama con / · ~500
- [x] **mcp.json** definiciones de tools · ~300

CONTEXTO ARMADO EN ESTA REQUEST

2.750

tokens de configuración, antes de su prompt

LECTURA

Stack saludable: global + alcance + dominio.

Estimaciones típicas por capa, para formar intuición de orden de magnitud. El punto: la configuración también es contexto pagado por request. La curaduría vale dinero.

EL ROI DE CONTEXTO

Sin jerarquía, el equipo paga el mismo contexto mil veces al día.

800 tok

PEGADOS A MANO EN CADA CHAT: STACK, CONVENCIONES, PATRONES DEL REPO

40/dia

CHATS POR DEV POR DÍA, EN EQUIPOS QUE USAN AGENTES EN SERIO

27M

TOKENS POR MES DESPERDICIADOS POR UN EQUIPO DE 20, SIN JERARQUÍA

La cuenta: 800 tokens de contexto repetido, por 40 chats, por 20 devs, por 20 días hábiles. La jerarquía convierte esos 32 millones de tokens repetidos en archivos versionados, cargados una vez, heredados por todos.

EL ROI DE CONTEXTO · ANTES Y DESPUÉS

La jerarquía no solo ahorra tokens. Ahorra el trabajo de recordar.

SIN JERARQUÍA

- Cada dev pega stack y convenciones en el prompt, cada vez.
- El contexto muere cuando el chat cierra. Cero en cada sesión.
- Dos devs explican lo mismo de dos formas distintas.

→

CON JERARQUÍA

- ✓ El contexto vive en Git. Escrito una vez, heredado por todos.
- ✓ Carga selectiva: el agente paga solo por la capa que la tarea activa.
- ✓ Una fuente de verdad. Cambia el archivo, cambia para todo el equipo.

La ganancia real no es el token, es la consistencia. Cuando el conocimiento vive en el repo, el agente es bueno el primer día de un dev nuevo, no en su tercer mes.

EL ROI DE CONTEXTO · CINCO PALANCAS

Cada mecanismo de la plataforma corta tokens de un modo distinto.

SKILL.md	Carga progresiva en tres etapas: solo el índice queda en contexto hasta que la tarea coincide.	40 tok vs 1.200
*.instructions.md	applyTo con glob: la regla de API solo entra cuando editas un archivo de API.	paga por alcance
*.agent.md	El agente estrecha el contexto a su dominio y a una allowlist breve de tools.	contexto enfocado
GitHub Copilot Spaces	Curaduría: adjunta los archivos que importan en vez del repo entero. Busca, no vuelca.	grounding curado
GitHub GitHub Copilot Memory	Aprende el repo solo y evita re-explicación: menos prompt, menos mantenimiento de instructions.	cero re-prompt

Cinco palancas, un principio: el contexto correcto, en el momento correcto, pagado una vez. La curaduría no es estética, es la línea de FinOps que nadie mira.

COMPOSICIÓN EN ACCIÓN

Un proyecto, cada capa en su lugar.

my-ecommerce · tree .github

```
my-ecommerce/
├── .github/
│   ├── copilot-instructions.md   # global: stack + conventions
│   ├── instructions/
│   │   ├── react.instructions.md  # scope: src/components/**
│   │   └── api.instructions.md    # scope: src/app/api/**
│   ├── agents/
│   │   ├── security.agent.md      # persona: security review
│   │   └── db-expert.agent.md     # persona: database expert
│   ├── skills/
│   │   ├── api-design/SKILL.md    # domain: REST patterns
│   │   └── testing/SKILL.md       # domain: test standards
│   └── prompts/
│       └── fix-bug.prompt.md      # template: debug + fix
├── .vscode/
│   └── mcp.json                   # external connections
└── src/
    ├── components/                # matched by react rules
    └── app/api/                   # matched by api rules
```

UN PROPÓSITO POR ARCHIVO

Cada archivo responde una pregunta. Nada de documento atrapa-todo de 500 líneas.

COMPOSICIÓN AUTOMÁTICA

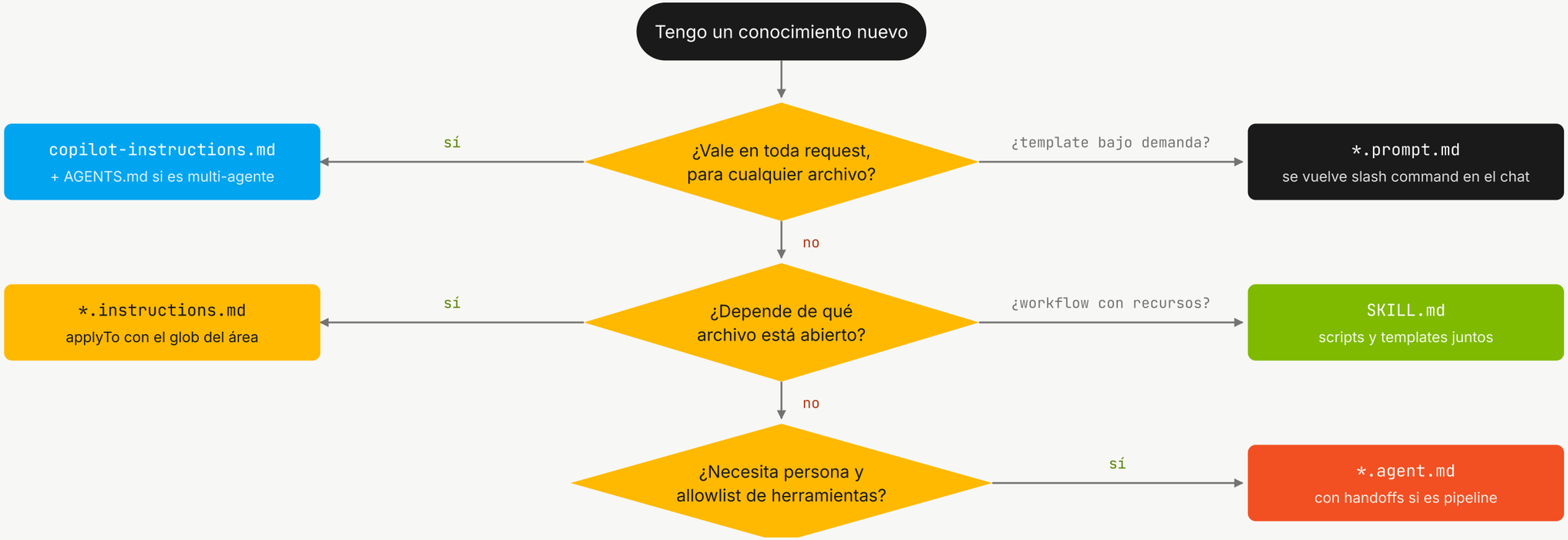
GitHub Copilot ensambla el contexto correcto para cada archivo que usted edita. Sin cableado manual.

VERSIONADO CON EL CÓDIGO

Toda la stack vive en git. Los cambios de config pasan por pull request como cualquier código.

LA RESOLUCIÓN · ÁRBOL DE DECISIÓN

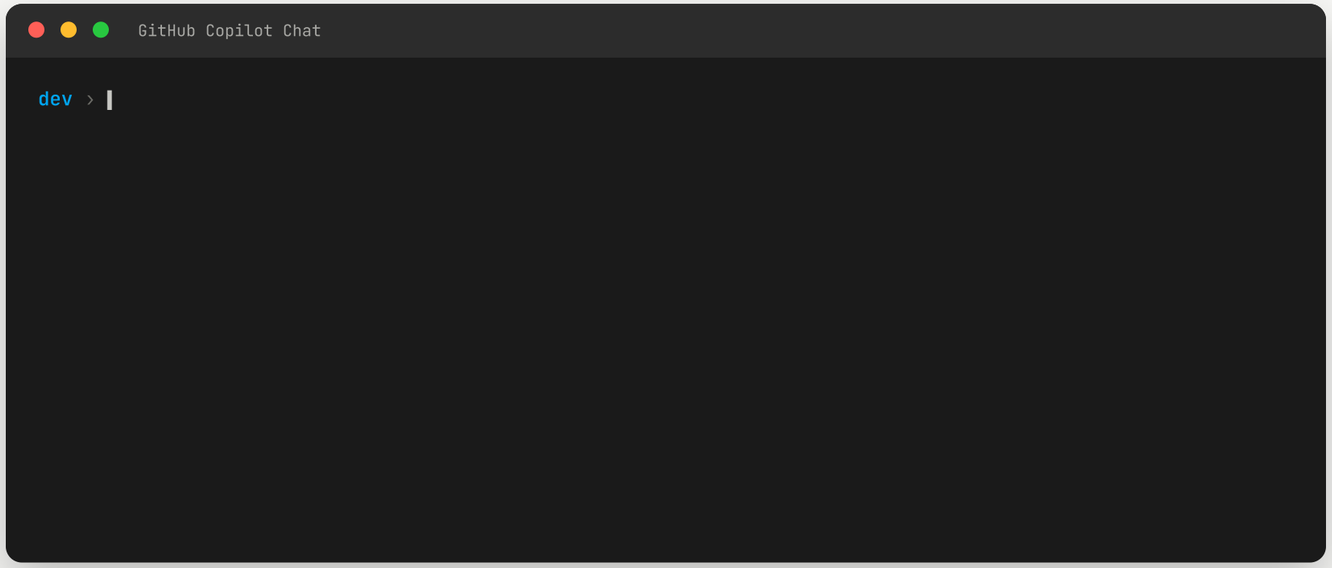
¿Qué archivo crear? Cuatro preguntas lo resuelven.



Regla práctica: empiece siempre por el nivel más alto que siga siendo verdadero. Conocimiento que vale para todo sube; conocimiento que vale para un caso baja. La duplicación entre niveles es la señal de que se respondió la pregunta equivocada.

LA JERARQUÍA TRABAJANDO

Un pedido, tres capas aplicadas, cero setup manual.



CAPA GLOBAL

La stack y las convenciones vinieron de copilot-instructions.md sin que nadie pegara nada.

CAPA DE ALCANCE

Las reglas de API se activaron porque la ruta de destino hizo match con el glob del applyTo.

CAPA DE DOMINIO

La skill de diseño REST trajo workflows y checklists, no solo reglas.

PARTE

IV

En la práctica.

Tres escenarios enterprise, el impacto medido y la lista de hacer o no hacer que mantiene la stack sana.

TRES EQUIPOS, TRES CONFIGURACIONES

Las mismas cinco capas resuelven tres problemas muy distintos.

TECH LEAD · STARTUP FINTECH

Ana estandariza 8 devs

Las instrucciones globales definen la stack, tres instruction files de alcance cubren componentes, API y tests, un agente de seguridad escanea cada PR, y una skill de compliance codifica el dominio financiero.

-40%
code reviews manuales

STAFF ENGINEER · E-COMMERCE ENTERPRISE

Carlos rompe el monolito

Un archivo global de arquitectura, instructions de alcance separadas por servicio (cart, payments, inventory), un agente migration-expert para el código legado y skills compartidas de event sourcing y CQRS.

18 → 10
meses de migración

PLATFORM ENGINEER · MULTINACIONAL

Maria escala a 50+ repos

Una plantilla organizacional que todo repo hereda, los equipos agregan reglas de alcance propias sin conflictos, las skills compartidas salen de un repositorio central y prompts estandarizados cubren operaciones comunes.

2w → 2d
onboarding de nuevos devs

Cifras observadas en adopciones enterprise de este patrón. Trátelas como dirección, y valídelas contra su propia baseline.

EN LA PRÁCTICA · ANTI-PATRONES

Cuatro anti-patrones que encontrará antes de cualquier buena práctica.

ANTI-PATRÓN 01

El archivo de 3.000 líneas

Todo el conocimiento del equipo volcado en copilot-instructions.md. Paga el costo en cada request, entierra lo crítico en lo irrelevante, y nadie lo revisa. Divídalo en capas: el slide del árbol de decisión es el mapa.

ANTI-PATRÓN 02

Generado y nunca curado

El /init genera un buen borrador, pero un borrador generado documenta lo obvio y omite lo tácito. Sin un dueño que recorte lo redundante cada trimestre, el archivo se vuelve ruido pagado por request.

ANTI-PATRÓN 03

La misma regla en tres capas

"Use TypeScript strict" en el global, en las instructions de API y en la skill. Como todo se apila en unión, la regla llega tres veces, y cuando el equipo cambia de opinión alguien olvida una copia. Cada regla vive en exactamente un nivel.

ANTI-PATRÓN 04

Secretos y ambiente en el lugar equivocado

Un token de API en un instructions file, una URL de producción en un prompt file. Esos archivos van a Git y al contexto del modelo. Los secretos viven en un vault; la config de ambiente vive en mcp.json con inputs, nunca hardcodeada.

Los cuatro salen de revisiones de stack reales. El patrón común: tratar la configuración de agentes como documentación, cuando es código, con costo por ejecución, dueño y ciclo de revisión.

HAGA Y NO HAGA

Tres hábitos separan las stacks sanas de las ruidosas.

NO HAGA

Poner todo en copilot-instructions.md. Un archivo de 500 líneas se vuelve ruido y el modelo pierde el foco.

NO HAGA

Escribir reglas contradictorias entre capas, como el global diciendo CSS Modules y el de alcance diciendo Tailwind.

NO HAGA

Volcar conocimiento de dominio complejo en instructions planas, sin estructura, sin workflow, sin checklist.

HAGA

Dividir en capas: global, alcance, agentes, skills. Cada archivo corto, enfocado y fácil de revisar.

HAGA

Hacer que las capas se complementen. Si el global define Tailwind, el archivo de alcance agrega reglas Tailwind específicas.

HAGA

Usar SKILL.md para dominios complejos: reglas más workflows más plantillas más checklists. La estructura garantiza completitud.

POR QUÉ VALE EL ESFUERZO

La configuración automatiza lo que el code review solía atrapar.

-34%

Code reviews manuales

La config automatiza patrones que antes dependían de revisión humana.

-50%

Tiempo de onboarding

Los nuevos devs generan código consistente desde el primer día, guiados por la config.

+29%

Consistencia de código

Patrones uniformes en todos los repos y equipos de la organización.

-42%

Bugs de patrón

Las reglas automáticas previenen errores comunes antes del commit.

Rangos observados en adopciones enterprise. Su baseline define sus números: mida antes y después.

PARTE

V

La capa de memoria.

Los archivos son lo que escribes. La memoria es lo que GitHub Copilot aprende solo, y recuerda entre sesiones.

LA CAPA DE MEMORIA · QUÉ ES

Stateless obliga a repetir. GitHub GitHub Copilot Memory recuerda entre sesiones.

TIPO 01 · HECHOS DEL REPOSITORIO

Lo que GitHub Copilot descubre del código

Convenciones, decisiones de arquitectura, comandos de build, dependencias entre archivos. Creados por quien tiene write access, disponibles para todo el equipo, atados a ese repo.

TIPO 02 · PREFERENCIAS DEL USUARIO

Cómo te gusta trabajar

Estilo de comunicación, stack preferida, convenciones de commit. Te siguen entre repositorios, sin afectar a otros. En Business y Enterprise, bajo gobernanza del admin.

Public preview, ene 2026, activada por defecto en Pro y Pro+ desde marzo. Disponible en coding agent, code review y CLI. Lo que un agente aprende, otro lo usa.

LA CAPA DE MEMORIA · CROSS-AGENT

Lo que el code review aprende, el coding agent lo aplica. Sin que repitas.

copilot-memory.log

```
[code-review] aprendió: ISafeAreaView e ISafeAreaView2 van juntos
↳ hecho guardado, citado en la línea 142, validado contra la branch

[coding-agent] 3 días después, en otra PR, actualiza ambos juntos
↳ sin que nadie re-explique la regla

[copilot-cli] en el terminal, aplica la convención de commits del equipo

3 superficies, 1 memoria compartida, validada antes de cada uso
```

VALIDADO ANTES DEL USO

Cada hecho lleva citas. GitHub Copilot verifica contra la branch actual antes de aplicar. Stale nunca entra.

EXPIRA EN 28 DÍAS

Un hecho sin usar desaparece solo. El timer se reinicia en cada uso validado. Sin basura acumulada.

ALCANCE POR REPO

Un hecho de un repo nunca se filtra a otro. Privacidad y seguridad por diseño.

LA CAPA DE MEMORIA · ¿MEMORIA O ARCHIVO?

La memoria es emergente. El archivo es deliberado. Necesitas ambos.

	GITHUB GITHUB COPILOT MEMORY	ARCHIVOS DE LA JERARQUÍA
Origen	El agente lo descubre solo mientras trabaja	Tú lo escribes y lo versionas en Git
Persistencia	Expira en 28 días si no se usa	Permanente hasta que alguien lo edite
Revisión	No pasa por un pull request	Revisado en una PR, auditable en el historial
Mejor para	Patrones que emergen del uso real	Reglas que quieres garantizar, siempre

Regla práctica: deja que la memoria descubra, luego gradúa lo que se repite a un archivo. Si GitHub Copilot reaprende lo mismo cada semana, es señal de que falta una instruction.

LA CAPA DE MEMORIA · GRADUACIÓN

El ciclo virtuoso: la memoria descubre, el archivo perpetúa.

LA MEMORIA APRENDE

El agente nota un patrón en el uso real y lo guarda como hecho.

→

REPITE DEMASIADO

La misma lección reaparece cada semana. Señal clara.

→

SE VUELVE ARCHIVO

Lo promueves a copilot-instructions.md o ARCHITECTURE.md.

GRADÚA CUANDO

"Siempre revisa ISafeAreaView e ISafeAreaView2 juntos" reaparece cada semana, hazlo una instruction.

POR QUÉ IMPORTA

Sin graduación, el aprendizaje valioso expira en 28 días. Con graduación, el repo acumula conocimiento permanente.

Memoria y archivo no compiten, se completan. La memoria es el borrador que el agente escribe solo. El archivo es la versión que decides volver ley.

CIERRE

La configuración es código para sus agentes.

Building the future of software development with AI and Agentic DevOps.

CONTACTO

Paula Silva
Software Global Black Belt

paulasilva@microsoft.com

PRÓXIMO PASO

Revisión de la stack de config
Un repositorio, una semana, cinco capas en su lugar

Publicado el 2026-06-11 · v6

TERMINAL · EMPIECE AQUÍ EL LUNES

```
# 1. Create the structure
$ mkdir -p .github/instructions \
    .github/agents .github/skills .github/prompts

# 2. Start with the global brain
$ touch .github/copilot-instructions.md
  stack, convenciones, comandos clave. Corto.

# 3. Add one scoped rule where pain lives
$ touch .github/instructions/api.instructions.md
  glob en applyTo + las reglas solo de esa área
```