EDICIÓN #016

Agent Skills: Qué son y cómo configurarlos en GitHub Copilot con SKILLS.md

14 de Enero de 2026

Agent Skills: Qué son y cómo configurarlos en GitHub Copilot con SKILLS.md

Descubre cómo funcionan los archivos SKILL.md, en qué se diferencian de los MCP y cómo configurar instrucciones modulares que se activan bajo demanda para ahorrar tokens.

14 January 2026

¿Alguna vez te has sentido como Bill Murray en El día de la marmota copiando y pegando el mismo prompt de trescientas líneas cada vez que empiezas una tarea nueva con **Claude** o **Copilot**? Ya sabes a lo que me refiero.

Tienes un proyecto con unas reglas de estilo muy concretas, usas Tailwind de una forma específica o necesitas que los tests sigan una estructura determinada. Y cada vez que abres un chat nuevo, te toca alimentar al modelo con todo ese contexto para que no te devuelva código genérico que no te sirve para nada. Y la verdad, da una pereza enorme...

Durante el último año nos hemos obsesionado con el *Context Window*. Pero la realidad es que **meter todo tu conocimiento en el prompt del sistema o en una ventana de chat gigante es ineficiente**, caro y, sinceramente, una mala arquitectura. El modelo se lía, pierde el foco y empieza a alucinar.

Aquí es donde entran los **Agent Skills**, un cambio fundamental en cómo generamos la arquitectura de relaciones entre nuestro conocimiento y la IA. Es pasar de pedirle cosas al modelo a "instalarle" capacidades modulares que solo usa cuando las necesita.

Hoy quiero explicarte **qué son exactamente estos Skills**, por qué GitHub y Anthropic están apostando todo a este estándar y, lo más importante, cómo puedes configurarlos hoy mismo para dejar de repetir las mismas instrucciones una y otra vez.

Resumen ejecutivo (TL;DR)

Los Agent Skills son paquetes de instrucciones y recursos que permiten estandarizar cómo una IA ejecuta tareas complejas sin saturar su contexto.

- **Son modulares:** Funcionan como carpetas con un archivo `SKILL.md` que el modelo carga solo cuando detecta que lo necesita, basándose en el principio de *Progressive Disclosure*.
- **Diferencia clave:** A diferencia de las Custom Instructions que están siempre activas (y pueden confundir al modelo), los Skills se activan bajo demanda para tareas específicas.
- **No son MCPs:** Mientras que el *Model Context Protocol* (MCP) conecta la IA con datos externos (bases de datos, APIs), los Skills definen *cómo* procesar esa información o generar outputs (estándares de código, guías de estilo). Para más información sobre sus diferencias consulta este post.
- **Portabilidad:** Se pueden compartir entre equipos mediante repositorios, permitiendo que todos los desarrolladores generen código o documentación siguiendo las mismas reglas corporativas.

Cómo funcionan los Agent Skills: Arquitectura de carpetas y carga dinámica

Para entender los Skills, primero tenemos que entender el problema que resuelven. Hasta ahora, teníamos dos formas de dar contexto a un modelo. O bien lo escribíamos todo en el prompt cada vez o bien usábamos las Custom Instructions o System Prompts de GitHub.

El problema de **los Custom Instructions es que son globales**. Si le dices a tu IA que "siempre actúe como un experto en SQL", cuando le pidas una imagen Docker ese sesgo va a estar ahí molestando y además, **ocupando Tokens de Contexto**. No puedes meter ahí la guía de estilos de tu empresa, la documentación de tu API interna y las reglas de seguridad de tus servidores.

Los Agent Skills solucionan esto aplicando un concepto de diseño de software llamado **Progressive Disclosure** o divulgación progresiva.

Imagina un manual de instrucciones gigante. En lugar de obligar al agente de IA a leerse las mil páginas antes de empezar a hablar contigo (lo que llenaría su contexto), le das solo el índice. El modelo sabe que existen capítulos sobre "Cómo crear componentes en React", "Cómo hacer migraciones de base de datos" o "Cómo redactar informes de incidentes".

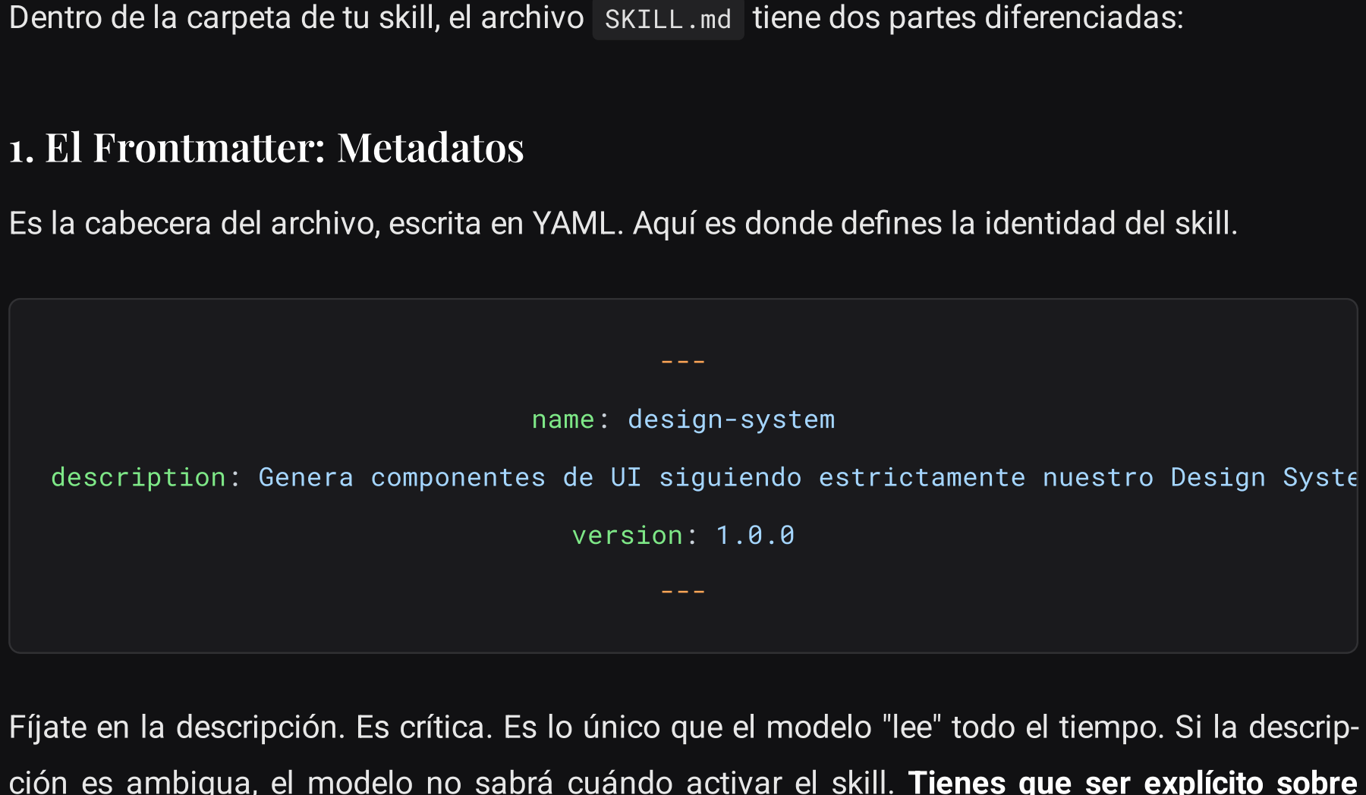
Cuando tú le pides "Crea un componente de botón", el modelo piensa: *"Vale, para esto necesito consultar el capítulo de componentes"*. Y es en ese momento, y solo en ese momento, **cuando carga el contenido de ese Skill en su contexto**.

Los agents skills son una tecnología desarrollada por Antrophic y de la que ya hablamos en este post hace unos meses. La adopción fue tan grande que otros proveedores de IA lo están adoptando como estándar junto con MCP [Guía de MCP desde cero].

Técnicamente, un Agent Skill no es más que una **carpeta**. Dentro de esa carpeta vive un archivo obligatorio llamado `SKILL.md` y, opcionalmente, otros recursos como **scripts**, **plantillas** o **ejemplos** de código. Es una forma de empaquetar conocimiento congelado que la IA puede descongelar a voluntad.

Arquitectura de un Agent Skill: Aprende a crear tu primer SKILL.md

Lo brillante de este sistema es que no necesitas aprender un lenguaje de programación nuevo ni una sintaxis compleja. Si sabes escribir **Markdown** y **YAML**, sabes crear un Skill.



```
agent-skill-ejemplo/
|– SKILL.md          # Instrucciones y metadatos*
|– scripts/          # Código ejecutable
|– references/       # Documentación
|– assets/           # Plantillas y recursos

* El fichero SKILL.md es obligatorio mientras que el resto de
carpetas son opcionales.

                                              lvrpiz.com
```

Diagrama de la estructura de archivos de un Agent Skill.

La estructura de carpetas estándar que utilizan tanto GitHub Copilot como Claude es muy sencilla. Generalmente, los encontrarás en `.github/skills/` (para skills específicos de un proyecto) o en directorios globales como `~/.copilot/skills`.

Dentro de la carpeta de tu skill, el archivo `SKILL.md` tiene dos partes diferenciadas:

1. El Frontmatter: Metadatos

Es la cabecera del archivo, escrita en YAML. Aquí es donde defines la identidad del skill.

```
---
name: design-system
description: Genera componentes de UI siguiendo estrictamente nuestro Design Syste
version: 1.0.0
---
```

Fíjate en la descripción. Es crítica. Es lo único que el modelo "lee" todo el tiempo. Si la descripción es ambigua, el modelo no sabrá cuándo activar el skill. **Tienes que ser explícito sobre *cuándo* debe usarse**. Cuantos más Skills tengan en tu proyecto más descriptivo debes ser para evitar confundir al modelo.

Antrophic recomienda que los metadatos no superen los 100 tokens. Si quieres conocer todos los campos disponibles lee la especificación completa.

2. El Cuerpo: Instrucciones

Aquí va el Markdown y donde **vuelcas tu conocimiento o cómo quieres que se comporte el modelo**. Pero a diferencia de un prompt normal, aquí puedes estructurarlo como documentación técnica.

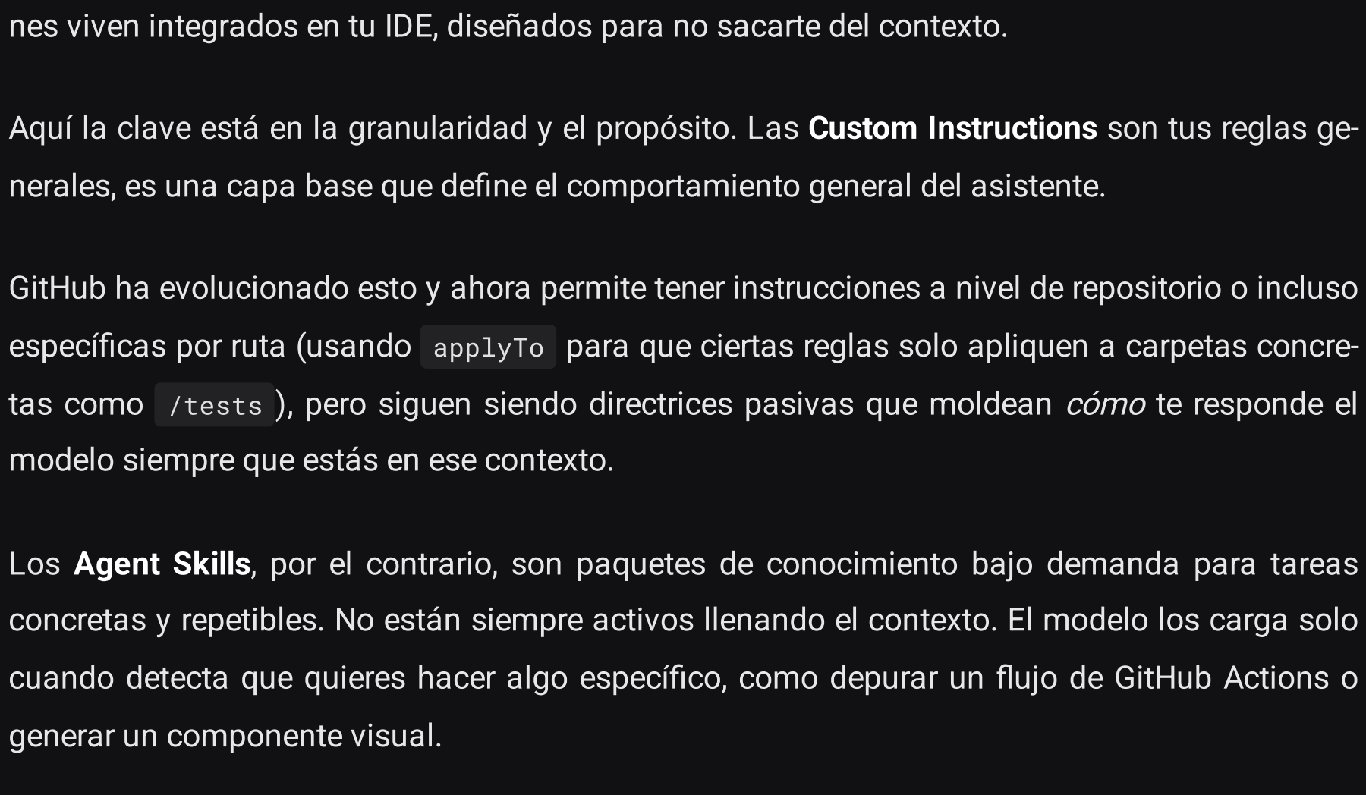
```
# Reglas de Implementación de UI (ejemplo)

## Colores Permitidos
No uses valores hex arbitrarios. Usa siempre las variables de CSS:
- Primario: var(--color-brand-primary)
- Secundario: var(--color-brand-secondary)

## Accesibilidad
Todo componente interactivo debe:
1. Tener un `aria-label` si no tiene texto visible.
2. Soportar navegación por teclado.
3. Pasar la validación de contraste WCAG AA.

## Patrón de Código (React)
Usa siempre componentes funcionales y TypeScript.
No uses `default export`, usa exportaciones nombradas.
```

Cuando activas este skill, Claude o Copilot no solo sabrán cómo programar en React, sino que lo harán como tú quieres que se haga en tu empresa o tu proyecto. En este caso Antrophic recomienda que las Skills no superen las 500 líneas o 5000 Tokens. Lee la especificación completa aquí.

TIP: Puedes validar que tu Skill está bien construida usando skills-ref de Antrophic. Esto comprueba que el SKILL.md sea válido y siga todas las convenciones de nomenclatura.

Agent Skills vs Custom Instructions vs MCP vs GPTs: Diferencias clave y cuándo usar cada uno

Sabiendo qué es un Agent Skill y antes de enseñarte cómo crearlos te estarás preguntando en qué se diferencia de las Custom Instructions, o de un GPT o incluso de MCP...



Custom Instructions	Agent Skills	MCP	GPTs
Reglas globales que definen personalidad y formato siempre activos para personalizar cada respuesta del asistente de inteligencia artificial.	Carpetas con instrucciones y scripts especializados que la IA carga automáticamente solo cuando la tarea requiere ese conocimiento específico.	Protocolo estándar que conecta modelos de IA con fuentes de datos externas, herramientas locales y APIs de forma universal.	Asistentes personalizados que combinan instrucciones, conocimientos y capacidades básicas para realizar tareas específicas dentro de un entorno cerrado.

lvrpiz.com

Infografía comparativa de herramientas de IA: Custom Instructions, Agent Skills, MCP (Model Context Protocol) y GPTs, explicando sus funciones de personalización, especialización y conexión de datos.

Piensa en los **GPTs** como islas separadas: son herramientas potentes pero aisladas a las que tienes que ir expresamente, rompiendo tu flujo en el editor. En cambio, los Skills y las instrucciones viven integrados en tu IDE, diseñados para no sacarte del contexto.

Aquí la clave está en la granularidad y el propósito. Las **Custom Instructions** son tus reglas generales, es una capa base que define el comportamiento general del asistente.

GitHub ha evolucionado esto y ahora permite tener instrucciones a nivel de repositorio o incluso específicas por ruta (usando `applyTo` para que ciertas reglas solo apliquen a carpetas concretas como `/tests`), pero siguen siendo directrices pasivas que moldean *cómo* te responde el modelo siempre que estás en ese contexto.

Los **Agent Skills**, por el contrario, son paquetes de conocimiento bajo demanda para tareas concretas y repetibles. No están siempre activos llenando el contexto. El modelo los carga solo cuando detecta que quieres hacer algo específico, como depurar un flujo de GitHub Actions o generar un componente visual.

Y para cerrar el círculo tenemos el **MCP** (Model Context Protocol), que a menudo se confunde con los anteriores. Mientras que un Skill es el manual que le dice a la IA cómo hacer un trabajo, el MCP es el conector que le da las herramientas para hacerlo.

Imagina un caso real: podrías tener un Skill llamado `debug-workflow` que contenga la lógica de pasos para solucionar un error, y que para ejecutarse utilice herramientas de un servidor MCP (como `get_job_logs`) para leer los datos reales del fallo.

En **resumen:**

- **Custom Instructions** para el estilo y reglas fijas a nivel general.
- **Agent Skills** para flujos de trabajo complejos bajo demanda.
- **MCP** para conectar con datos y herramientas externas.
- **GPTs** para cuando necesites una herramienta aislada fuera de tu código.

Tutorial: Crea un Agent Skill para automatizar la revisión de código

Vamos a bajar al barro. Uno de los mejores casos de uso para empezar es un **skill de revisión de código**. Todos sabemos que los *code reviews* son necesarios, pero a veces se nos pasan cosas o nos da pereza ser exhaustivos con el *linting*.

Vamos a crear un skill que enseñe a Copilot a revisar PRs con tus criterios.

Paso 1: Crear el directorio. En la raíz de tu proyecto, crea `.github/skills/code-reviewer/`.

Paso 2: Crear el archivo de definición. Crea `.github/skills/code-reviewer/SKILL.md`.

Paso 3: Definir el contenido

```
---
name: strict-code-reviewer
description: Realiza revisiones de código enfocadas en seguridad, rendimiento y ma
---

# Guía de Revisión de Código

Actúa como un Tech Lead senior y antipático pero justo. Tu objetivo es encontrar p

## Checklist de Seguridad
- ¿Hay inputs de usuario sin sanitizar?
- ¿Se están exponiendo secretos o keys en el código?
- ¿Hay llamadas a API sin gestión de errores (try/catch)?

## Checklist de Rendimiento
- Busca bucles O(n^2) innecesarios.
- Identifica re-renderizados en componentes de React que no estén memorizados.
- Detecta importaciones de librerías gigantes cuando solo se usa una función.

## Formato de Salida
No me des el código corregido directamente. Primero lista los problemas encontrado
```

Ahora, cuando estés en VS Code y le digas a Copilot que revise un archivo, si tiene acceso a este skill, aplicará esta lógica. No te dirá simplemente "está bien", sino que buscará activamente lo que le has pedido en la sección de seguridad y rendimiento.

Lo potente de esto es que este archivo vive en tu repositorio. Si mañana entra un junior al equipo, al hacer `git pull`, ya tiene por defecto este Skill en su asistente.

Skills avanzados: Ejecución de código y validación automática

Hasta ahora hemos hablado de Skills que son básicamente texto. Pero la cosa se pone seria cuando **le damos a los Skills la capacidad de ejecutar código**. Esto es lo que se conoce como "**Code-Enabled Skills**".

Imagina el ejemplo del Design System que comentábamos antes. Está muy bien decirle a la IA que se asegure de que el contraste sea suficiente. Pero los LLMs son malos calculando matemáticas de colores. A veces te dicen que un gris sobre blanco se lee bien cuando no es cierto.

En un Skill avanzado, **puedes incluir un script de Python o JavaScript** en la misma carpeta.

```
design-system/
├── SKILL.md          # Archivo principal
└── validate-a11y.js  # Scripts ejecutables
```

Por ejemplo, podrías tener `design-system/validate-a11y.js`, un script pequeño que toma dos códigos hexadecimales y calcula matemáticamente el ratio de contraste según el estándar WCAG.

En tu SKILL.md, le dirías al modelo:

```
Cuando generes una combinación de colores, no adivines el contraste. Ejecuta el sc
```

Aquí es donde la frontera entre generador de texto y agente se empieza a difuminar ya que el modelo no solo escribe, sino que utiliza herramientas que tú le has proporcionado para validar su propio trabajo antes de entregártelo.

Esto es vital para tareas donde la precisión es no negociable, como cálculos financieros, conversiones de unidades o validaciones de seguridad.

El futuro de la documentación: Documentación para Máquinas

Llevo tiempo dándole vueltas a una idea: estamos entrando en una era donde la documentación técnica ya no se escribe para humanos, sino para IAs.

Nadie se lee los manuales de 500 páginas. Lo que hacemos es preguntarle a ChatGPT. Por tanto, si eres el responsable técnico de una empresa o mantienes una librería Open Source, tu prioridad debería ser que la IA entienda tu producto.

Los Agent Skills son el primer paso hacia esta **documentación ejecutable**.

En lugar de tener una wiki en Confluence que nadie actualiza y que explica "Cómo desplegar a producción", deberíamos tener un Skill `deployment-guide` en el repositorio.

Esto tiene un **efecto secundario maravilloso**: el Bus Factor.

Si el único desarrollador que sabe cómo funciona el sistema de facturación legacy se va de la empresa, es un drama. Pero si **ese desarrollador ha volcado su conocimiento en un conjunto de Skills bien definidos, el siguiente desarrollador (apoyado por la IA) podrá operar el sistema con mucha más seguridad**.

Estamos empaquetando el conocimiento organizacional en un formato que es legible tanto por humanos (es Markdown al fin y al cabo) como por máquinas.

Cómo empezar a usar esto hoy mismo

Actualmente, el soporte de Skills está llegando de forma nativa a muchas herramientas.

1. **En GitHub Copilot:** Si tienes acceso a las versiones *preview* o *insiders* de VS Code, puedes empezar a crear la carpeta `.github/skills` en tus repositorios hoy mismo. Copilot empezará a indexarlos. Más información de cómo usarlo en VSCode.
2. **En Claude:** Puedes usar proyectos y simular skills subiendo archivos de texto, pero la integración real viene con herramientas de terceros y con el uso de la API para construir tus propios agentes.
3. **Community Skills:** Ya hay repositorios como `github/awesome-copilot` o `anthropics/skills` donde la gente está subiendo skills prefabricados. Es una forma genial de aprender viendo cómo otros estructuran sus instrucciones. Revisa las referencias técnicas al final del post para acceder a ellas.

Mi recomendación es que empieces por lo que más te duela. **Identifica esa tarea repetitiva que haces todos los días**, esa en la que siempre tienes que corregir a la IA ("no, usa la librería X, no la Y"). Crea un Skill para eso.

Hay una **diferencia muy grande y que cada vez es más evidente** entre los programadores que usan estas soluciones como Agent Skills, MCP, Custom Instructions y los de... "Hazme el componente de Login". Si quieres que la IA trabaje como si fueras tú mismo (o mejor) debes enseñarla.

No se trata de que la IA trabaje más, sino de que trabaje mejor. Y para eso, primero tenemos que aprender a enseñarle.

Referencias técnicas

1. Documentación Oficial de GitHub Copilot Agent Skills - Guía completa sobre implementación y estructura.
2. Anthropic Skills Repository - Colección de skills de ejemplo y mejores prácticas mantenida por Anthropic.
3. AgentSkills.io - Directorio y recursos sobre la creación de skills para agentes.
4. Awesome Copilot - Lista curada de recursos, incluyendo ejemplos de skills comunitarios.
5. VS Code Custom Instructions - Documentación sobre cómo VS Code gestiona el contexto y las instrucciones personalizadas.
6. Qué son Agent Skills - Gustavo Espíndola

Recurso de referencia. Disponible en los siguientes formatos:

PDF
Markdown
Explora el Blueprint con IA