Buenas prácticas para crear Skills de agentes

Skills - Como escrirlos correctamente

Esta guía explica cómo escribir skills de nivel profesional para agentes, validarlas con LLMs y mantener una ventana de contexto ligera.

Se trata de un conjunto concentrado de buenas prácticas para crear skills de agentes. Si buscas la documentación completa, consulta la documentación de Claude.

Estructura de una skill

Toda skill debe seguir esta estructura de directorios:

Plaintext

```
nombre-de-la-skill/
├── SKILL.md              # Obligatorio: Metadatos e instrucciones principales (<500 líneas)
├── scripts/              # Código ejecutable (Python/Bash) diseñado como mini CLIs
├── references/           # Contexto complementario (esquemas, hojas de referencia)
└── assets/               # Plantillas o archivos estáticos para la salida
```

- **SKILL.md:** Actúa como el "cerebro". Úsalo para navegación y procedimientos de alto nivel.
- **References:** Enlázalos directamente desde SKILL.md. Mantén **un solo nivel de profundidad**.
- **Scripts:** Úsalos para operaciones frágiles o repetitivas donde la variación es un bug. **No incluyas código de biblioteca aquí**.

Optimiza el frontmatter para que sea fácil de encontrar

Los campos `name` y `description` en el frontmatter de tu `SKILL.md` son lo único que el agente ve antes de activar una skill. Si no están optimizados para ser detectables y lo bastante específicos, tu skill será invisible.

- **Nombres estrictos:** El campo name debe tener entre 1 y 64 caracteres, contener solo letras minúsculas, números y guiones (sin guiones consecutivos), y **debe coincidir exactamente con el nombre del directorio padre** (por ejemplo, name: `angular-testing` debe estar en `angular-testing/SKILL.md`).
- **Descripciones optimizadas para activación:** (Máximo 1024 caracteres). Son los únicos metadatos que el agente ve para decidir el enrutamiento. Describe la funcionalidad en tercera persona e incluye "disparadores negativos".
  - **Mal:** "Skills de React." (Demasiado vago).
  - **Bien:** "Crea y compila componentes React con Tailwind CSS. Se usa cuando el usuario quiere actualizar estilos de componentes o lógica de interfaz. No usar para proyectos con Vue, Svelte o CSS puro."

Revelación progresiva y gestión de recursos

Mantén la ventana de contexto limpia cargando información solo cuando haga falta. **SKILL.md** es el "cerebro" para la lógica de alto nivel; delega los detalles a subdirectorios.

- **Mantén SKILL.md ligero:** Limita el archivo principal a **menos de 500 líneas**. Úsalo para navegación y procedimientos principales.
- **Usa subdirectorios planos:** Mueve el contexto voluminoso a carpetas estándar. Mantén los archivos a **un solo nivel de profundidad** (por ejemplo, `references/schema.md`, no `references/db/v1/schema.md`).
  - `references/`: Documentación de APIs, hojas de referencia, lógica de dominio.
  - `scripts/`: Código ejecutable para tareas deterministas.
  - `assets/`: Plantillas de salida, esquemas JSON, imágenes.
- **Carga justo a tiempo (JiT):** Indica explícitamente al agente cuándo debe leer un archivo. No verá estos recursos hasta que se lo indiques (por ejemplo, "*Consulta `references/auth-flow.md` para los códigos de error específicos*").
- **Rutas explícitas:** Usa siempre **rutas relativas** con barras inclinadas (`/`), independientemente del sistema operativo.

Las skills son para agentes, no para humanos. Para mantener la ventana de contexto ligera y evitar un consumo innecesario de tokens, **no crees:**

- **Archivos de documentación:** `README.md`, `CHANGELOG.md` o `INSTALLATION_GUIDE.md`.
- **Lógica redundante:** Si el agente ya resuelve una tarea de forma fiable sin ayuda, elimina la instrucción.
- **Código de biblioteca:** Las skills deben referenciar herramientas existentes o contener scripts pequeños de propósito único. El código de biblioteca de larga vida pertenece a los directorios CLI estándar del repositorio.

Usa instrucciones procedimentales específicas en lugar de prosa

Escribe instrucciones para LLMs, no para humanos.

- **Numera los pasos:** Define el flujo de trabajo como una secuencia cronológica estricta. Si hay un árbol de decisiones, mapéalo con claridad (por ejemplo, "*Paso 2: Si necesitas source maps, ejecuta `ng build --source-map`. De lo contrario, pasa al Paso 3.*").
- **Proporciona plantillas concretas:** Los agentes son excelentes reconociendo patrones. En lugar de gastar párrafos describiendo cómo debe lucir una salida JSON, coloca una plantilla en la carpeta assets/ e indica al agente que copie su estructura.
- **Escribe en tercera persona imperativa:** Formula las instrucciones como órdenes directas al agente (por ejemplo, "*Extraer el texto...*" en lugar de "*Voy a extraer...*" o "*Deberías extraer...*").

Sé específico y consistente en la forma en que referencias conceptos en tus archivos de skill.

- **Usa terminología idéntica:** Elige un solo término para referirte a un concepto específico.
- **Especificidad:** Usa la terminología más específica y nativa del dominio que describes. Por ejemplo, en Angular usa el concepto "template" en lugar de "html", "markup" o "view".

Incluye scripts deterministas para operaciones repetitivas

No le pidas al LLM que escriba lógica compleja de parsing o código boilerplate desde cero cada vez que ejecuta una skill.

- **Delega las tareas frágiles o repetitivas:** Si el agente necesita parsear un dataset complejo o consultar una base de datos específica, proporciónale un script probado en Python, Bash o Node dentro del directorio scripts/.

- **Maneja los casos límite con elegancia:** Un agente depende de la salida estándar (stdout/stderr) para saber si un script fue exitoso. Escribe scripts que devuelvan mensajes de error descriptivos y legibles para que el agente sepa exactamente cómo autocorregirse sin necesidad de intervención del usuario.

Guía de validación

Dado que los LLMs van a usar tus skills, la mejor forma que he identificado para asegurar su utilidad es colaborando con LLMs.

Es fundamental contar con evaluaciones para tus skills que confirmen que los cambios que haces tienen un impacto positivo y no provocan regresiones. Un benchmark popular para skills es SkillsBench, que puede servir de inspiración.

Una vez que tengas el borrador inicial de tus skills, puedes validar tu trabajo siguiendo estos pasos:

Validación de detección

Los agentes cargan skills basándose exclusivamente en el frontmatter YAML. Prueba cómo un LLM interpreta tu descripción de forma aislada para evitar activaciones falsas (como dispararse para una app React cuando está pensada para Angular).

Pega exactamente el siguiente texto en un chat nuevo con un LLM:

> Estoy creando una Skill de Agente basada en la especificación de agentskills.io. Los agentes decidirán si cargar esta skill basándose enteramente en los metadatos YAML de abajo.
>
> ```
> name: angular-vite-migrator
> description: Migrates Angular CLI projects from Webpack to Vite and esbuild.
> Use when the user wants to update builder configurations, replace webpack
> plugins with rollup equivalents, or speed up Angular compilation.
> ```
>
> Basándote estrictamente en esta descripción:
>
> 1. Genera 3 prompts realistas de usuario que estés 100% seguro de que deberían activar esta skill.
> 2. Genera 3 prompts de usuario que suenen parecido pero que NO deberían activarla (por ejemplo, migrar una app React a Vite, o simplemente actualizar versiones de Angular).

> 3. Critica la descripción: ¿Es demasiado amplia? Sugiere una reescritura optimizada.

Además, envía a los agentes tareas que esperas que activen la lectura de una skill e inspecciona el proceso de razonamiento. Ve y viene con el agente para identificar por qué eligió (o no) determinadas skills.

Validación de lógica

Asegúrate de que tus instrucciones paso a paso sean deterministas y no obliguen al agente a alucinar pasos que faltan.

Pásale al LLM tu `SKILL.md` completo y la estructura de directorios:

> Aquí está el borrador completo de mi SKILL.md y el árbol de directorios con sus archivos de soporte.
>
> ```
> ├── SKILL.md
> ├── scripts/esbuild-optimizer.mjs
> └── assets/vite.config.template.ts
> ```
>
> [Pega aquí el contenido de tu SKILL.md] Actúa como un agente autónomo que acaba de activar esta skill. Simula tu ejecución paso a paso a partir de una solicitud para "Migrar mi app Angular v17 a Vite".
>
> Para cada paso, escribe tu monólogo interno:
>
> 1. ¿Qué estás haciendo exactamente?
> 2. ¿Qué archivo o script específico estás leyendo o ejecutando?
> 3. Señala cualquier bloqueo de ejecución: indica la línea exacta donde te ves obligado a adivinar o alucinar porque mis instrucciones son ambiguas (por ejemplo, cómo mapear los archivos de entorno de Angular a `import.meta.env` de Vite).

Pruebas de casos límite

Obliga al LLM a buscar vulnerabilidades, configuraciones no soportadas y estados de fallo inherentes a las herramientas web.

Pídele al LLM que ataque tu lógica:

>

> Ahora cambia de rol. Actúa como un tester QA implacable. Tu objetivo es romper esta skill. Hazme de 3 a 5 preguntas muy específicas y desafiantes sobre casos límite, estados de fallo o fallbacks que faltan en el SKILL.md. Enfócate en:
>
> - ¿Qué pasa si `scripts/esbuild-optimizer.mjs` falla por una dependencia legacy en CommonJS?
> - ¿Qué pasa si el `angular.json` del usuario tiene builders de Webpack muy personalizados (`@angular-builders/custom-webpack`) que Vite no soporta?
> - ¿Hay suposiciones implícitas que hice sobre el entorno de Node del usuario?
>
> No corrijas estos problemas todavía. Solo hazme las preguntas numeradas y espera a que las responda.

Refinamiento de arquitectura

Los LLMs suelen intentar meter archivos de configuración grandes directamente en el prompt principal. Usa este paso para aplicar la revelación progresiva y reducir el consumo de tokens.

Haz que el LLM aplique tus correcciones y reestructure la skill:

> Basándote en mis respuestas a tus preguntas sobre casos límite, reescribe el archivo SKILL.md aplicando estrictamente el patrón de diseño de revelación progresiva:
>
> 1. Mantén el `SKILL.md` principal estrictamente como un conjunto de pasos de alto nivel usando comandos imperativos en tercera persona (por ejemplo, "Ejecutar el script de esbuild", "Leer la plantilla de configuración de Vite").
> 2. Si hay reglas densas, plantillas extensas de `vite.config.ts` o esquemas complejos de `angular.json` actualmente en el archivo, elimínalos. Indícame que cree un nuevo archivo en `references/` o `assets/`, y reemplaza el texto en `SKILL.md` con una instrucción estricta para leer ese archivo específico solo cuando sea necesario.
> 3. Añade una sección dedicada de manejo de errores al final que incorpore mis respuestas sobre los fallbacks de Webpack y la resolución de CommonJS.

- Gracias a Best Practices for Creating Agent Skills

Revision #1
Created 2026-02-26 07:30:43 UTC by Abkrim

Updated 2026-02-26 07:33:39 UTC by Abkrim