GOOGLE ANTIGRAVITY PARA DEVELOPERS

Delega una funcionalidad sin perder el control

BIG school

Con agentes, ya no basta con pedir código

Durante bastante tiempo, programar con IA consistía en algo muy fácil de entender:

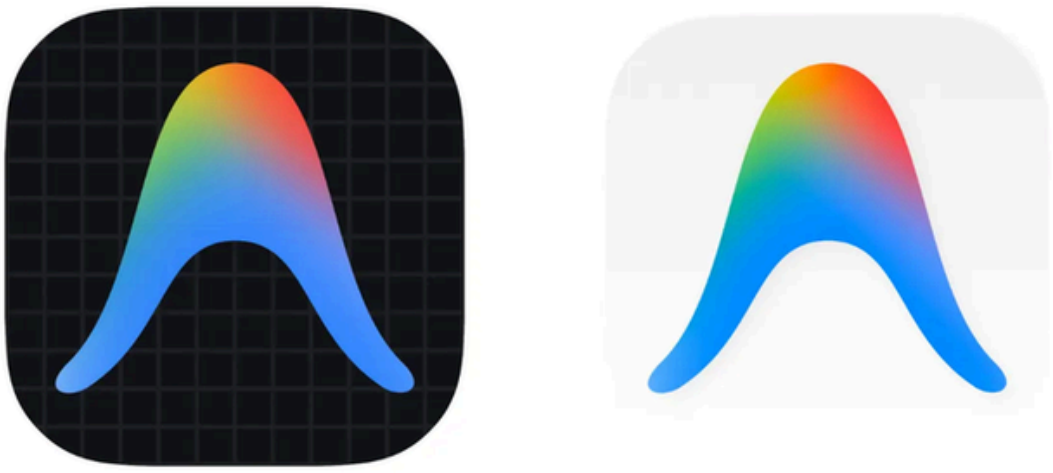
Abrías un chat, pedías una función, copiabas el código en tu editor y comprobabas si aquello funcionaba.

Con un agente, la situación cambia bastante. Ya no estás pidiendo únicamente unas líneas de código.

Puedes darle una tarea, dejar que analice el proyecto, decida qué archivos necesita revisar, proponga un plan, modifique código y ejecute pruebas para comprobar parte del resultado.

Suena muy bien. Y lo es. Pero precisamente por eso **aparece una responsabilidad nueva.**

Cuanto más trabajo puede ejecutar un agente por su cuenta, **más importante es que tú tengas claro qué le estás delegando, dónde debe parar y qué vas a revisar antes de aceptar el resultado.**



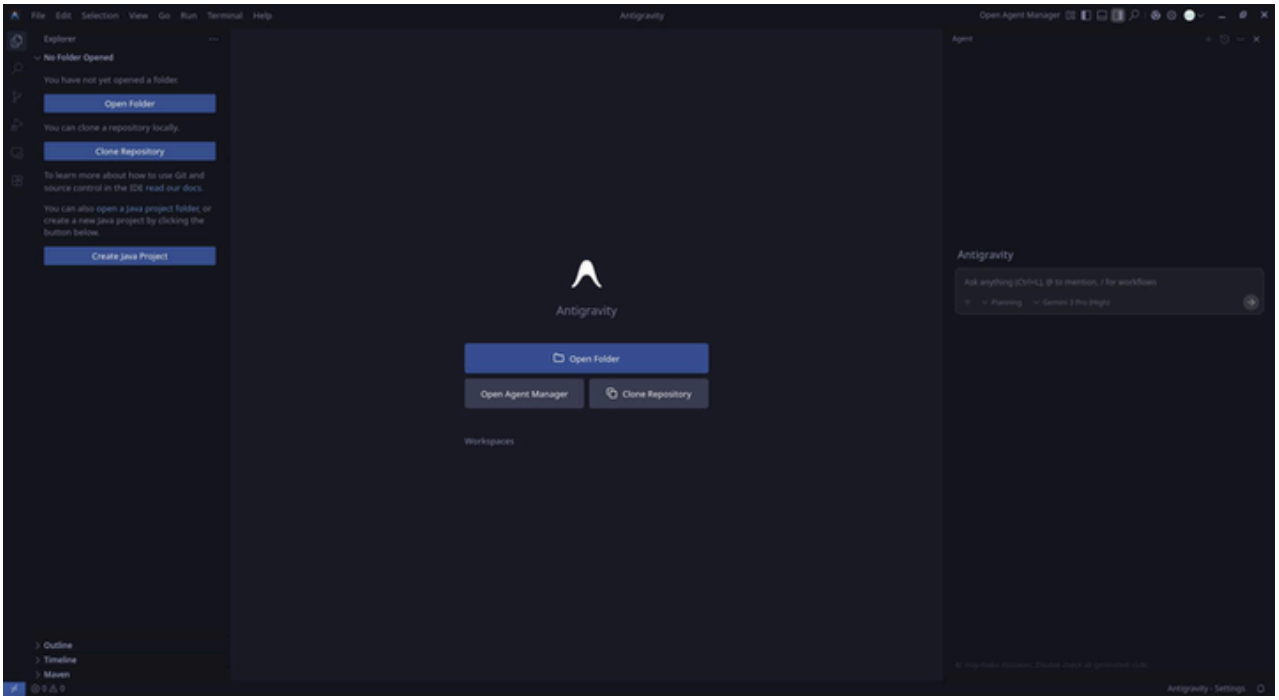
Logos de Antigravity IDE y Antigravity 2.0 respectivamente.

En esta guía no quiero enseñarte todo Google Antigravity. Para eso necesitaríamos bastante más tiempo y acabaríamos otra vez intentando aprender veinte cosas a la vez.

Vamos a centrarnos en algo que creo que te vendrá muy bien:

Delegar una pequeña funcionalidad de principio a fin sin convertirnos en espectadores de nuestro propio código.

Para hacerlo, vamos a trabajar desde **Antigravity IDE.**



El ecosistema tiene más herramientas y superficies. Sin embargo no las necesitas para entender este flujo.

Nuestro ejemplo será una aplicación sencilla de tareas a la que queremos añadir búsqueda por título. No es la app de la NASA, aunque nos permite ver todo el proceso sin que la arquitectura nos distraiga de lo importante.

La idea que quiero que tengas en la cabeza durante toda la guía es esta:

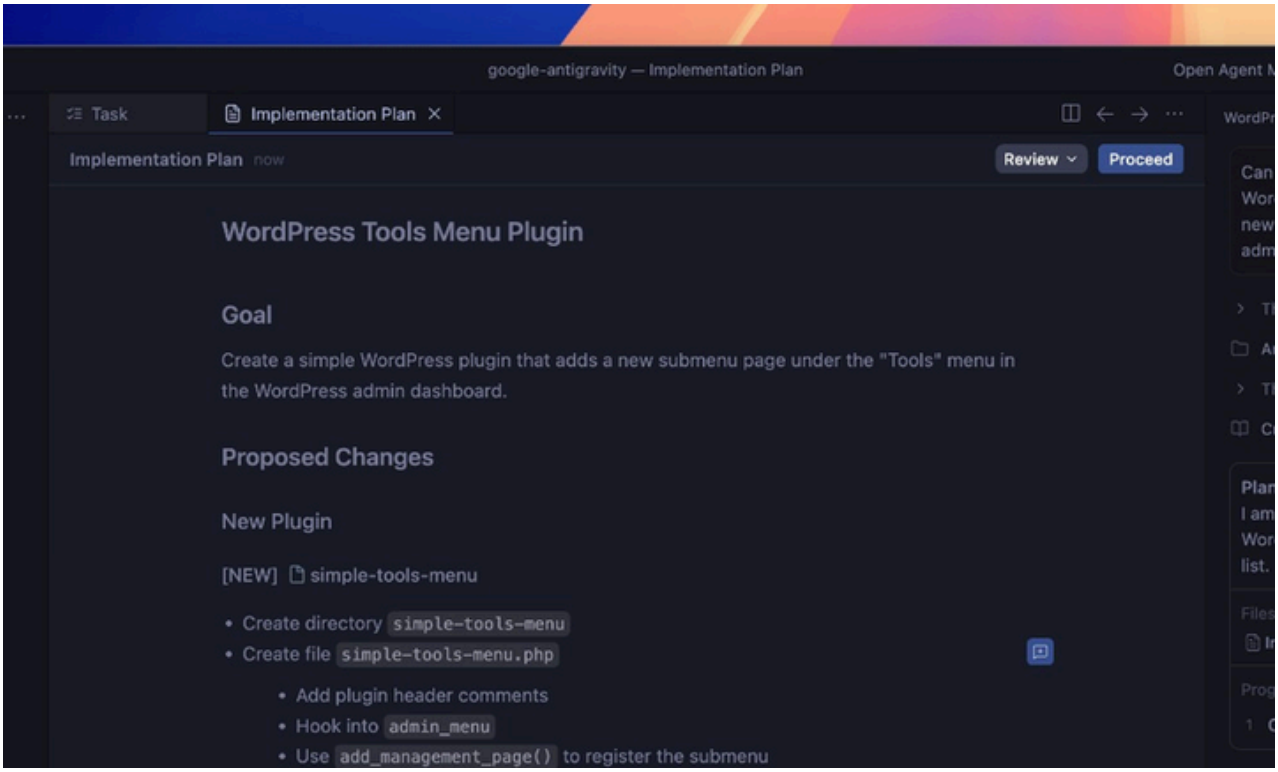
El agente ejecuta. Tú sigues dirigiendo.

1. Antes de delegar, haz que el agente pare y te enseñe el Plan

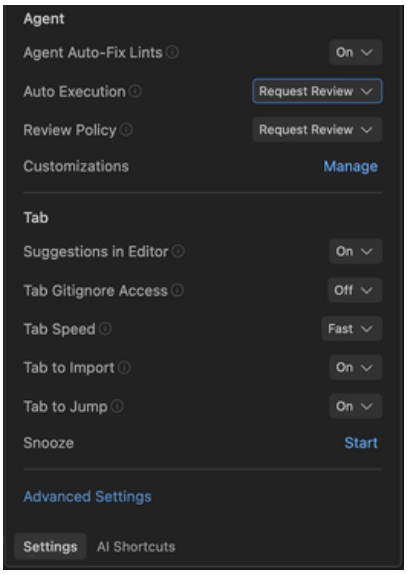
Abre en Antigravity IDE un proyecto pequeño que conozcas. Yo no empezaría experimentando con un repositorio enorme que no entiendes demasiado bien, porque entonces aparece un problema bastante evidente:

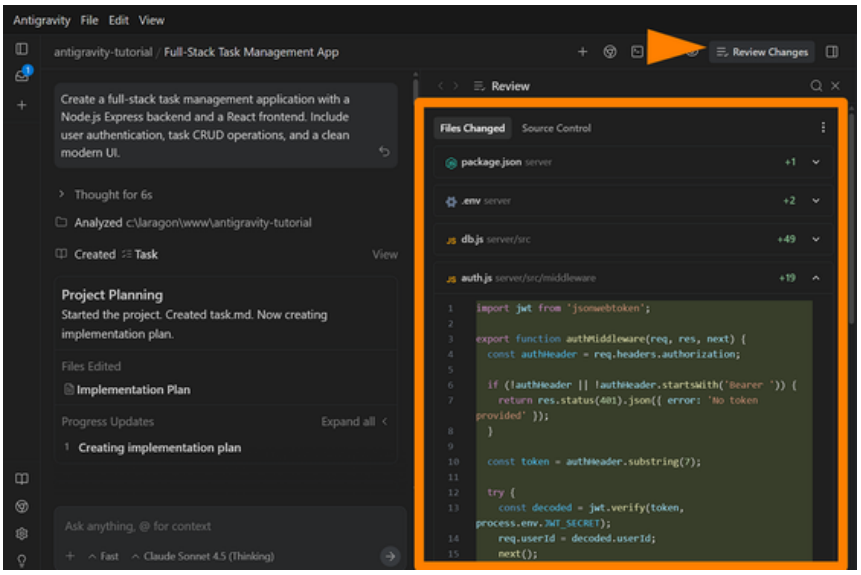
Si tú no sabes qué debería ocurrir, también te va a costar mucho saber si el agente lo ha hecho bien.

Para una tarea como la que vamos a hacer, me interesa trabajar en **Planning Mode**. En este modo, el agente analiza primero el trabajo y puede preparar un **Implementation Plan** antes de empezar a modificar el código.



Y hay otro ajuste que utilizaría al empezar: **Request Review.** Así, cuando el agente llegue a uno de esos puntos de control, se detendrá para **pedirte aprobación en lugar de continuar directamente.**





Ahora… ¿Existe la posibilidad de darle más autonomía?

Sí. También existe Fast Mode para tareas más simples y localizadas, y políticas que permiten al agente seguir sin esperar tu revisión. Sin embargo, ahora mismo nuestro objetivo no es descubrir cuántas cosas puede hacer solo, sino **aprender a delegar sin perder visibilidad.**

En caso de que solo quieras renombrar una variable, probablemente no necesites montar todo este proceso.

En cambio, si vas a delegar una funcionalidad, sí merece la pena dedicar un poco más de tiempo a definirla y revisar el plan.

2. Encárgale una tarea, no le tires una frase

Vamos con nuestro ejemplo para que puedas ver cómo funcionaría esto de forma práctica:

Queremos añadir una búsqueda por título a una aplicación de tareas. Podríamos abrir el Agent Panel y escribir:

"Añade un buscador a mi aplicación."

¿Puede salir bien? Sí.

Piensa en todo lo que acabamos de dejar en el aire:

¿Sobre qué campo tiene que buscar?

¿Debe distinguir mayúsculas y minúsculas?

¿Qué ocurre si la consulta está vacía?

¿Puede instalar una dependencia?

¿Tiene que crear tests?

¿Puede aprovechar para refactorizar otras cosas que no le hemos pedido?

La IA puede tomar todas esas decisiones por ti. El problema es que quizás no tome las mismas que tomarías tú.

Por eso, en lugar de pedir código, encárgale una tarea con cuatro piezas muy sencillas: **objetivo, contexto, restricciones y criterios de aceptación.**

Para nuestro ejemplo utilizaría una instrucción parecida a esta:

```
Quiero añadir búsqueda por título a la lista de tareas
existente.

Antes de modificar código:
1. Analiza cómo funciona actualmente la lista de tareas
y su filtrado.
2. Identifica los archivos que sería necesario modificar.
3. Prepara un Implementation Plan.
4. No implementes ningún cambio hasta que revise el plan.

Restricciones:
- Mantén el stack y la arquitectura actuales.
- No añadas dependencias nuevas salvo que sean realmente
necesarias y me expliques por qué.
- No modifiques funcionalidades que no estén relacionadas
con esta tarea.

Consideraré la tarea terminada cuando:
- La búsqueda se realice por título.
- Ignore mayúsculas y minúsculas.
- Elimine espacios al principio y al final de la consulta.
- Si la consulta está vacía, se muestren todas las tareas.
- Existan tests para estos comportamientos.
```

Fíjate en algo importante: No le hemos explicado cómo escribir la función.

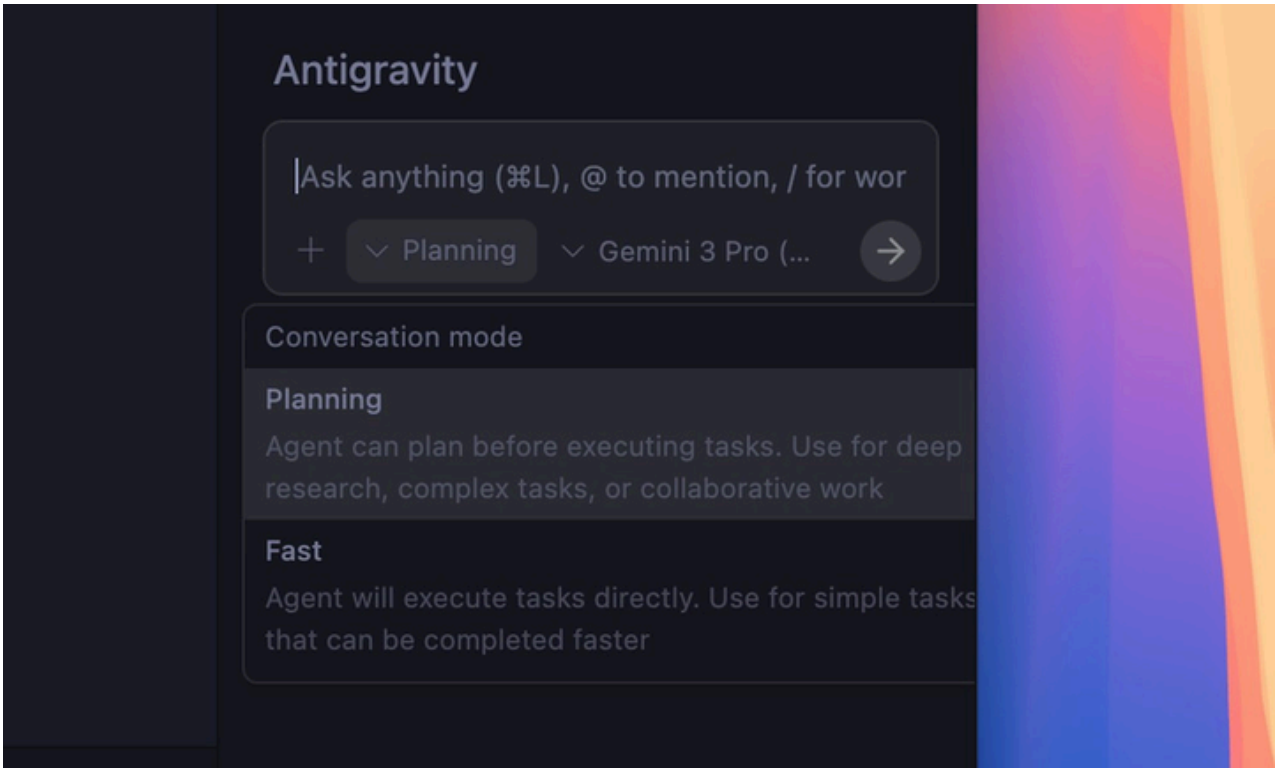
Tampoco le hemos dado una solución cerrada. **Le hemos explicado qué problema tiene que resolver, qué límites debe respetar y qué significa para nosotros que el trabajo esté terminado.**

Asegurarte de que el agente entienda eso es mucho más interesante que intentar escribir el “prompt perfecto”.

3. Antes de pulsar Proceed, lee lo que piensa hacer

En **Planning Mode**, Antigravity puede convertir la tarea en un **Implementation Plan.**

Básicamente, es la **propuesta de trabajo del agente antes de empezar a tocar el proyecto.**



Y este es uno de los momentos más importantes de todo el flujo.

Así que, no pases el plan como quien acepta las cookies de una web. Léelo, estúdialo, y si es necesario, discútelo con el agente.

Es normal que la IA cometa errores; tu trabajo es auditarlo con criterio. Por eso, antes de pulsar Proceed, yo comprobaría al menos estas cinco cosas:

1. ¿Ha entendido realmente el objetivo de la tarea?

2. ¿Piensa modificar únicamente los archivos necesarios?

3. ¿La solución encaja con la arquitectura y las convenciones actuales del proyecto?

4. ¿Quiere añadir alguna dependencia o complejidad que no necesitamos?

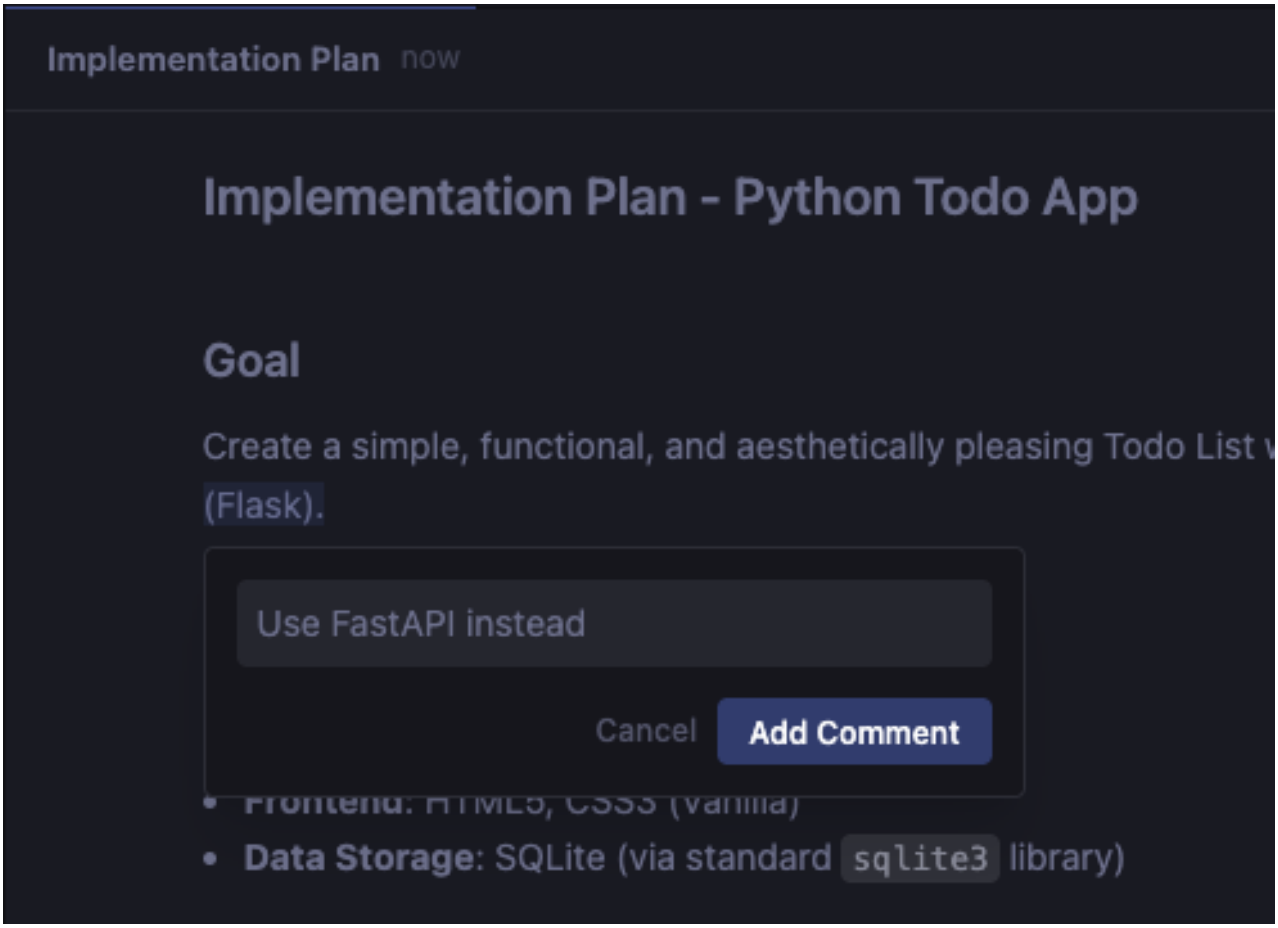
5. ¿Cómo piensa comprobar que la implementación funciona?

Ahora, volvamos a nuestro buscador.

Imagina que, para filtrar unas tareas por título, el agente propone instalar una librería de búsqueda, crear un nuevo servicio y modificar media arquitectura.

Igual funciona, sí. Sin embargo, seguramente acabamos de complicar un problema que era bastante más sencillo.

Aquí tienes un gran ejemplo de cuándo **podrías comentar directamente el plan** y pedirle que reduzca el scope, cambie un enfoque o corrija algo que haya entendido mal antes de que modifique el código.



Da igual cuál sea tu proyecto. Quédate con esto:

Un agente rápido ejecutando un mal plan sigue ejecutando un mal plan (y al final te hace perder más tiempo).

De hecho, lo único que hemos conseguido es llegar al problema mucho más rápido. El problema sigue estando ahí.

4. Ahora sí: Deja que el agente implemente

Cuando el plan ya tiene sentido, puedes aprobarlo y dejar que el agente empiece a trabajar.

Aquí es donde aparece la parte espectacular de estas herramientas: el agente puede avanzar sobre el proyecto, modificar los archivos necesarios, utilizar el terminal y ejecutar las comprobaciones que formen parte de la tarea.

No hace falta que te quedes mirando cada movimiento como si estuvieras viendo una barra de progreso. **Lo importante es que hayas establecido buenos puntos de control y sepas qué vas a revisar cuando termine.**

En nuestra aplicación podríamos partir de una función así:

```
export function filterTasks(tasks, query) {
  return tasks.filter(task => task.title.includes(query));
}
```

Después de implementar los requisitos, el agente podría dejar algo parecido a esto:

```
export function filterTasks(tasks, query) {
  const normalizedQuery = query.trim().toLowerCase();

  if (!normalizedQuery) {
    return tasks;
  }

  return tasks.filter(task =>
    task.title.toLowerCase().includes(normalizedQuery)
  );
}
```

Bien. Ha escrito el código. Incluso puede haberlo hecho en unos segundos.

Eso no es lo más importante.

Lo importante es que tú sabes por qué existe cada uno de esos cambios, porque los criterios estaban definidos antes de empezar: normalizar la consulta, ignorar mayúsculas y minúsculas y tratar correctamente una búsqueda vacía.

La autonomía del agente tiene sentido cuando no convierte el resultado en una caja negra.

5. Review Changes: Revísalo como si fuese un pull request

El agente ha terminado de escribir código. Ahora vuelve nuestro trabajo.

Primero hay una revisión funcional: *¿Ha resuelto exactamente la tarea que le encargamos? Después viene una revisión técnica: ¿Cómo la ha resuelto?.*

En nuestro ejemplo, la primera revisión consiste en volver a los criterios de aceptación y comprobar que la búsqueda funciona por título, ignora mayúsculas y minúsculas, elimina espacios, trata correctamente una consulta vacía y cuenta con los tests que habíamos pedido.

En tu proyecto esos criterios serán otros. Esa es precisamente la idea: no memorices este checklist. Define qué significa "terminado" antes de delegar y vuelve a esa definición cuando el agente acabe.

Después abre Review Changes. Desde ahí puedes recorrer los diffs de los archivos que se han modificado y comentar sobre ellos para devolver feedback al agente.

Trátalo como una code review. Yo me fijaría especialmente en esto:

Scope: ¿Ha tocado únicamente lo necesario o aparecen cambios que no tienen relación con la tarea?

Implementación: ¿Entiendes lo que ha hecho y por qué?

Arquitectura: ¿Respeta las decisiones y convenciones actuales del proyecto?

Complejidad: ¿Ha creado más abstracciones, dependencias o código del que realmente necesitábamos?

Riesgo: ¿Puede haber efectos secundarios o regresiones en funcionalidades existentes?

Tests: ¿Comprueban de verdad los casos importantes o simplemente existen para que podamos decir que hay tests?

Hay una regla que me parece bastante útil: si no aprobarías ese cambio en una revisión de código hecha por otra persona, tampoco deberías aprobarlo porque lo haya escrito un agente.

Y si aparece código que no entiendes, no lo aceptes por fe. Puedes pedirle algo así:

```
Explícame este cambio paso a paso sin modificar nada.

Quiero entender:
¿Qué problema resuelve cada cambio?;
¿Por qué has elegido esta solución?;
¿Qué alternativas había?;
¿Qué riesgos tiene?;
Y ¿qué podría romperse?.
```

Esto no es perder tiempo. Si cada vez que utilizas un agente tu proyecto avanza, mientras que tú entiendes un poco menos lo que contiene, **estás ganando velocidad a cambio de dependencia.**

Y, personalmente, no me parece un buen trato.

6. Que los tests estén en verde no cierra la tarea

En nuestra instrucción pedimos tests y el agente puede generarlos y ejecutarlos. Perfecto. Pero que aparezca todo en verde no significa automáticamente que el trabajo esté terminado.

Los tests solo comprueban aquello que alguien decidió comprobar. Si nos hemos olvidado de un caso límite, podemos tener una suite completamente verde y seguir teniendo un problema.

En nuestro ejemplo podríamos preguntarnos qué ocurre si `query` contiene únicamente espacios, si existen títulos repetidos o si el nuevo filtrado ha cambiado algún comportamiento que ya funcionaba antes.

No necesitas inventarte cien escenarios. Necesitas pensar cuáles son relevantes para la tarea y para el proyecto que tienes delante.

Cuando Antigravity termina una implementación, también puede generar un Walkthrough con un resumen del trabajo realizado.

Úsalo como una ayuda para entender qué ha hecho y cómo lo ha validado, no como un certificado de que todo está perfecto.

Además, previo a dar por finalizada una tarea, puedes hacer una última comprobación con una instrucción como esta:

```
Antes de cerrar la tarea:

1. Ejecuta las pruebas relacionadas.
2. Busca casos límite que no hayamos contemplado.
3. Comprueba si el cambio puede haber provocado alguna
regresión.
4. Resume cualquier riesgo que siga abierto.

No realices nuevos cambios sin indicármelo primero.
```

"He terminado" no es una prueba. "Los tests pasan" tampoco debería ser el final automático del proceso.

7. La última palabra sigue siendo tuya

Llegados a este punto, el agente puede haber analizado el proyecto, propuesto un plan, escrito el código, generado tests y explicado qué ha cambiado.

Ha hecho bastante trabajo.

Todavía queda una pregunta que ninguna herramienta debería responder por ti:

"¿Doy este cambio por bueno?"

Ese es el cambio de mentalidad que me interesa de trabajar con agentes.

Al final, no se trata de conseguir que la IA escriba más código mientras nosotros miramos. Se trata de poder delegar más parte de la ejecución sin delegar también las decisiones que determinan la calidad del software.

Si quieres quedarte con un único flujo de toda esta guía, que sea este:

1. Define: Explica qué quieres conseguir, qué contexto necesita el agente, cuáles son los límites y qué significa dar la tarea por terminada.

2. Planifica: Deja que el agente estudie el proyecto y te proponga cómo hacerlo.

3. Revisa: Comprueba el plan y corrige lo que no encaje antes de que empiece a modificar código.

4. Delega: Cuando el plan tenga sentido, deja que implemente lo que ya habías acordado con el agente.

5. Valida: Vuelve a los criterios, revisa los tests y analiza el diff.

6. Decide: Al final, eres tú quien debe determinar si el trabajo se da por bueno o si necesita otra iteración.

Piensa en la IA como un medio, no como un fin.

Con esto me refiero a que, por ejemplo, Antigravity me parece interesante precisamente porque puede encargarse de una parte cada vez mayor del trabajo de desarrollo. Eso **solo aporta valor si tú sigues entendiendo el problema y controlando el resultado**.

Si acabas una tarea y no sabes explicar qué ha cambiado y por qué, probablemente has delegado demasiado.

Si el agente te ha ahorrado trabajo y tú sigues pudiendo revisar, cuestionar e iterar sobre lo que ha hecho, entonces sí estamos aprovechando la herramienta.

Ahora llévate este flujo al desarrollo con IA

Esto conecta directamente con lo que vamos a trabajar en el **Curso de Desarrollo con IA: el Nuevo Programador.**

En la segunda jornada, Programa con agentes bajo control, veremos cómo pasar de pedir fragmentos de código a delegar tareas completas, entendiendo qué contexto necesita un agente, cuándo puedes darle autonomía y qué puntos debes revisar antes de aceptar su trabajo.

Porque la IA puede escribir muchísimo código. Tú sigues teniendo que saber dirigir el proceso.

ACTIVA LA CAMPANITA EN YOUTUBE

PARA QUE TE AVISE CUANDO COMIENCE LA CLASE #1 DEL CURSO.

ACCEDE AQUÍ A LA PRIMERA CLASE

BIG school