Claude Code no solo sirve para pedirle cambios puntuales en un repositorio. También podemos usarlo para dejar tareas en marcha dentro de una sesión, ya sea repitiendo una comprobación cada cierto tiempo o trabajando hasta que se cumpla una condición concreta. Ahí entran dos comandos especialmente útiles: **/loop** y **/goal**. Se parecen porque ambos permiten que Claude Code siga actuando sin que tengamos que escribir un prompt nuevo a cada paso, pero resuelven problemas distintos.

La diferencia rápida es esta:

- **/loop** sirve para repetir una tarea con el paso del tiempo.
- **/goal** sirve para seguir trabajando hasta cumplir un objetivo.

/loop

Usamos **/loop** cuando queremos que Claude Code repita un prompt dentro de la sesión actual. Es útil para:

- Tareas de vigilancia.
- Espera o seguimiento.
- Comprobar si ha terminado un despliegue.
- Revisar si CI ya ha pasado.
- Mirar si hay nuevos comentarios en una pull request.
- Comprobar de vez en cuando si una build larga ha terminado.
- Etc

Para usarlo necesitamos Claude Code v2.1.72 o superior. Podemos comprobar la versión instalada con:

```
claude --version
```

Un ejemplo típico sería:

```
/loop 5m Comprueba si el deployment ha terminadoy dime que ha pasado
```

En este caso, Claude Code ejecutará ese prompt cada cinco minutos. Es decir, no está intentando arreglar activamente el despliegue sino que está comprobando periódicamente su estado. También podemos dejar que Claude elija el intervalo:

```
/loop Comprueba si CI pasó y encárgate de manejar cualquier comentario de review
```

Cuando no indicamos un intervalo, Claude Code decide cada cuánto repetir la tarea según lo que observe. Puede esperar menos si parece que algo está a punto de terminar, o más si no hay actividad relevante. Incluso podemos lanzar **/loop** sin argumentos:

```
/loop
```

En ese caso se ejecuta el comportamiento de mantenimiento por defecto. Claude intentará continuar trabajo pendiente, revisar el estado de la sesión, atender comentarios, comprobar conflictos o hacer tareas razonables dentro del contexto ya existente.

Si queremos personalizar ese comportamiento por defecto, podemos crear un archivo llamado **loop.md** dentro del proyecto:

```
.claude/loop.md
```

O también a nivel de usuario:

```
~/.claude/loop.md
```

Ese archivo sustituye el prompt de mantenimiento por defecto cuando ejecutamos **/loop** sin argumentos.

/goal

Usamos **/goal** cuando no queremos que Claude Code repita una comprobación por tiempo, sino que trabaje hasta alcanzar una condición concreta. Para usarlo necesitamos Claude Code v2.1.139 o superior. El patrón básico es:

```
/goal Todas las pruebas en test/auth pasan y el paso de lint está limpio.
```

Al ejecutar ese comando, Claude Code empieza a trabajar inmediatamente. No hace falta escribir otro prompt después. El propio objetivo se convierte en la instrucción principal. Después de cada turno, Claude Code usa un evaluador rápido para comprobar si la condición ya se ha cumplido. Si todavía no se cumple, continúa con otro turno. Si se cumple, el objetivo se limpia automáticamente y la sesión deja de insistir.

Un ejemplo más realista en un repositorio Node.js podría ser:

```
/goal npm test termina con código 0 y git status está limpio.
```

Ese objetivo está bien planteado porque tiene una condición verificable: los tests tienen que terminar correctamente y el estado de Git debe quedar limpio. También podemos usarlo para tareas de refactorización:

```
/goal Migra el módulo de autenticación a la nueva API hasta que todas las pruebas pasen.
```

O para arreglar una integración continua fallida:

```
/goal
Arregla el trabajo de CI que está fallando, demuéstralo haciendo que la prueba fallida pase y detente después de 20
 turnos si no se resuelve.
```

La parte final es importante. Si la tarea puede atascarse, conviene poner un límite: número máximo de turnos, tiempo máximo o una condición alternativa de parada. Así evitamos dejar a Claude trabajando demasiado tiempo en algo que quizá necesita intervención humana.

Buenos objetivos con /goal

Un mal objetivo sería algo ambiguo como:

```
/goal mejora el proyecto
```

Eso no tiene una condición clara. Claude puede interpretar muchas cosas distintas: refactorizar, cambiar tests, mejorar nombres, tocar documentación o modificar estructura.

Un buen objetivo debería incluir tres cosas: una condición medible, una forma de comprobarla y restricciones si hacen falta. Por ejemplo:

```
/goal pytest termina con código 0, ruff check pasa y no se modifica ningún archivo fuera de src/auth y tests/auth.
```

Ahí estamos diciendo exactamente qué debe conseguirse, cómo debe probarse y qué no queremos que toque.

Otro ejemplo útil:

```
/goal
Todos los archivos de src/controllers tienen menos de 300 líneas, npm test termina con código 0 y la API pública pe
rmanece sin cambios.
```

Esto nos sirve para una refactorización más controlada. No le estamos diciendo simplemente “ordena el código”; le estamos dando una condición de finalización concreta.

Diferencias prácticas entre /loop y /goal

/loop trabaja por tiempo. Lo usamos cuando hay que comprobar algo periódicamente.

```
/loop 10m Comprueba si el despliegue en producción ha terminado y resume el resultado.<br />
```

/goal trabaja por condición. Lo usamos cuando hay que resolver algo hasta que quede hecho.

```
/goal
El despliegue en producción está arreglado, las comprobaciones de salud están en verde y la prueba de humo que fall
aba pasa.
```

La diferencia es importante. Si solo queremos esperar a que CI termine, usamos **/loop**. Si queremos que Claude intente arreglar CI hasta dejarlo verde, usamos **/goal**.

Otra forma de verlo:

- **/loop**: “comprueba esto cada cierto tiempo”.
- **/goal**: “trabaja hasta que esto esté conseguido”.

Limitaciones que conviene tener claras

/loop está ligado a la sesión. No es lo mismo que una automatización persistente en la nube. Si cerramos la sesión o Claude Code deja de estar disponible, las tareas de ese loop dejan de ejecutarse. Para automatizaciones más persistentes tendríamos que mirar otras opciones como tareas programadas de Desktop, rutinas o GitHub Actions.

/goal tampoco es magia. El evaluador no entra por su cuenta al sistema de archivos ni ejecuta comandos de forma independiente. Juzga lo que Claude ha mostrado en la conversación. Por eso es importante pedirle que enseñe pruebas: salida de tests, estado de Git, resultado de build o cualquier otra evidencia verificable.

También debemos tener cuidado con los permisos. Si Claude necesita editar archivos, ejecutar comandos o usar herramientas, la configuración de permisos de la sesión sigue importando. **/goal** evita que tengamos que iniciar cada turno manualmente, pero no convierte una configuración restrictiva en una sesión completamente autónoma.