GUÍA PASO A PASO

Permisos de Claude Code en settings.json

El settings.json comentado regla por regla para un proyecto Node o Next.js, el archivo local y el de usuario, un script en Node que revisa los errores comunes y cómo comprobar que las reglas aplican. Con la checklist.

PARA	Proyectos Node o Next.js en los que trabajas con Claude Code
PROBADO CON	Node 22.22.3 · archivos validados con Ajv contra el schema publicado de settings · check-permissions.mjs con archivos correctos y con errores
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/permissions
ARTÍCULO	ecomlabs.dev/g/permisos
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Cómo deciden las reglas
Paso 01	El settings.json del proyecto, comentado
Paso 02	El archivo local y el de usuario
Paso 03	El script de revisión
Paso 04	Comprobar que las reglas aplican
Checklist	Antes de hacer commit

ANTES DE EMPEZAR

Cómo deciden las reglas

- Cada regla es `Herramienta` o `Herramienta(especificador)` y va en `allow`, `ask` o `deny`. Se revisan así: primero deny, luego ask, al final allow. La primera que coincide decide, aunque otra sea más específica.
- Las listas de todos los archivos se suman y un deny en cualquiera bloquea. El orden de los archivos, de más a menos fuerza: administrado, `claude --settings`, `.claude/settings.local.json`, `.claude/settings.json`, `~/.claude/settings.json`.
- Si ninguna regla coincide, decide el modo de permisos. Deny bloquea en todos los modos; ask pregunta incluso en `auto` y `bypassPermissions`.

PASO 01

El settings.json del proyecto, comentado

Guárdalo como `.claude/settings.json` en la raíz del proyecto y súbelo a git. Es JSON estricto, sin comentarios ni comas finales, así que los comentarios van en la tabla. Validado con Ajv contra el schema publicado (`json.schemastore.org/claude-code-settings.json`):

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run typecheck)",
      "Bash(npm run build)",
      "Bash(npm test)",
      "Bash(npm run test *)"
    ],
    "ask": [
      "Bash(git commit *)",
      "Bash(git push *)",
      "Bash(npm install *)",
      "Bash(npm i *)",
      "Bash(rm *)",
      "Bash(curl *)"
    ],
    "deny": [
      "Read(**/.env)",
      "Read(**/.env.*)",
      "Read(!.env.example)",
      "Read(secrets/**)",
      "Read(**/*.pem)",
      "Bash(git push --force *)",
      "Bash(npm publish *)"
    ]
```

```
  }
}
```

REGLA	QUÉ HACE Y CUÁNDO CAMBIARLA
`Bash(npm run lint)` y las otras tres exactas	Corren sin preguntar y sin argumentos extra. Usa los nombres de tu `package.json`.
`Bash(npm run test *)`	`npm run test` con argumentos. No cubre `npm run test:e2e`: el espacio es parte de la regla.
`Bash(git commit *)`, `Bash(git push *)`	Te preguntan antes de cada commit y push, también en modo auto.
`Bash(npm install *)`, `Bash(npm i *)`	Dos reglas porque Bash compara texto y no conoce alias.
`Bash(rm *)`, `Bash(curl *)`	Preguntan en vez de bloquear: borrar `.next` o probar `localhost` es normal.
`Read(**/.env)`, `Read(**/.env.*)`, `Read(!.env.example)`	Bloquean leer, editar y crear `.env` y sus variantes a cualquier profundidad; el `!` deja leer la plantilla.
`Read(secrets/**)`, `Read(**/*.pem)`	Cualquier carpeta `secrets` y cualquier llave o certificado. Agrega tus `*.key`.
`Bash(git push --force *)`, `Bash(npm publish *)`	Deny va antes que ask. `git push origin main --force` no coincide y cae en ask. Una versión publicada no se reemplaza.

Con pnpm o yarn, escribe el comando como lo escribe tu equipo: `Bash(pnpm lint)` y `Bash(pnpm run lint)` son dos reglas distintas. En un monorepo, `Read(**/.env)` ya cubre `apps/web/.env`.

PASO 02

El archivo local y el de usuario

`.claude/settings.local.json` guarda reglas que solo valen en tu máquina, como un servidor MCP registrado solo para ti:

```
{
  "permissions": {
    "allow": ["Bash(npx playwright test *)", "mcp__postgres-ro__*"],
    "additionalDirectories": ["../design-tokens/"]
  }
}
```

- `npx` ejecuta lo que le pases: nombra también el comando de dentro. `Bash(npx *)` permitiría cualquier paquete.
- Las reglas `mcp__` van sin paréntesis. Si creas el archivo a mano, agrégalo a `.gitignore`.
- Un allow aquí no le gana a un ask del proyecto.

`~/.claude/settings.json` protege lo que vive fuera de los proyectos. Usa `~/`: con una sola `/`, la ruta apuntaría a `~/.claude`. Es el único de tus archivos donde `"auto"` funciona como `defaultMode`:

```
{
  "permissions": {
    "defaultMode": "acceptEdits",
    "deny": ["Read(~/.ssh/**)", "Read(~/.aws/**)", "Read(~/.npmrc)"]
  }
}
```

PASO 03

El script de revisión

`check-permissions.mjs` no tiene dependencias y no evalúa reglas: marca las trampas que describe la documentación y que el schema no detecta, como `Write(.env)`, `Bash(git:* push)` o `defaultMode` en `auto` dentro del proyecto.

```
// Usage: node check-permissions.mjs .claude/settings.json [.claude/settings.local.json ...]
// Flags the traps the Claude Code permissions docs describe. It does not
// evaluate rules: Claude Code does that. Exit code 1 if anything was flagged.
import { readFileSync } from "node:fs";
import { homedir } from "node:os";
import { join, resolve } from "node:path";

// ~/.claude/settings.json has the same name as a project file but is yours.
const USER_FILE = join(homedir(), ".claude", "settings.json");
let flagged = 0;

for (const file of process.argv.slice(2)) {
  const out = [];
  const flag = (msg) => out.push(msg);
  let settings;
  try {
    settings = JSON.parse(readFileSync(file, "utf8"));
  } catch (e) {
    flag(`not valid JSON (no comments, no trailing commas): ${e.message}`);
  }
  const p = settings?.permissions ?? {};
  const user = resolve(file) === USER_FILE;
  const inRepo = !user && /\.claude[\\/]settings(\.local)?\.json$/.test(file);
  const shared = !user && /\.claude[\\/]settings\.json$/.test(file);

  for (const list of ["allow", "ask", "deny"]) {
    for (const rule of p[list] ?? []) {
      if (/^(Write|NotebookEdit|MultiEdit|Glob|Grep)\((?!\*\))/.test(rule))
        flag(`${list}: ${rule} never matches a path; write Edit(...) or Read(...)`);
      if (/:\*(?!\)$)/.test(rule))
        flag(`${list}: ${rule} has ":*" before the end, where the colon is literal`);
      if (/^mcp__[^(]*\(/.test(rule))
        flag(`${list}: ${rule} is skipped: mcp__ rules take no parentheses in a settings file`);
      if (list === "allow" && /^(Bash|PowerShell)(\(\*.*\))?$/.test(rule))
```

```
        flag(`allow: ${rule} approves any program`);
    }
  }
  if (inRepo && ["auto", "bypassPermissions"].includes(p.defaultMode))
    flag(`defaultMode "${p.defaultMode}" has no effect in a project or local file`);
  if (shared && settings && !(p.deny ?? []).some((r) => /^Read\(.*\.env/.test(r)))
    flag("no Read deny rule for .env files");

  console.log(`${out.length ? "FAIL" : "OK  "} ${file}`);
  for (const msg of out) console.log(`     - ${msg}`);
  flagged += out.length;
}

process.exitCode = flagged ? 1 : 0;
```

Ejecútalo desde la raíz del proyecto: `node check-permissions.mjs .claude/settings.json .claude/settings.local.json`. Sale con código 1 si marca algo.

PASO 04

Comprobar que las reglas aplican

1. Abre Claude Code en la raíz del proyecto y acepta el diálogo de confianza: sin él, las reglas allow del proyecto no aplican.
2. `/status`: la línea *Setting sources* lista los archivos cargados. `/permissions`: cada regla con su archivo. `claude doctor`: las entradas rechazadas.
3. Pide «lee el .env». La lectura debe quedar rechazada.
4. Pide «ejecuta npm run lint». Debe correr sin preguntarte.
5. Pide «ejecuta git push --dry-run». Debe preguntarte, también en modo auto.

Una regla de Bash compara el texto del comando: no detiene `/usr/bin/curl` ni `sh -c`. Un deny de `Read` no alcanza a un script que abre el archivo por su cuenta. Para eso están el sandbox y los hooks `PreToolUse`.

CHECKLIST

Antes de hacer commit

- [ ] `.claude/settings.json` tiene `$schema`, las tres listas y los nombres de tu `package.json`.
- [ ] Ningún allow usa `Bash` sola ni un asterisco antes del subcomando.
- [ ] `ask` cubre push, commits, dependencias, borrados y red; `deny` cubre `.env`, llaves y `npm publish`.
- [ ] `.claude/settings.local.json` está fuera de git.
- [ ] `defaultMode` en `auto`, si lo quieres, solo en `~/.claude/settings.json`.
- [ ] `check-permissions.mjs` da OK y las pruebas del Paso 04 salen como esperas.