EcomLabs

STEP-BY-STEP GUIDE

Claude Code permissions in settings.json

The settings.json explained rule by rule for a Node or Next.js project, the local and user files, a Node script that checks for the common mistakes and how to confirm the rules apply. With a checklist.

FOR	Node or Next.js projects you work on with Claude Code
TESTED WITH	Node 22.22.3 · files validated with Ajv against the published settings schema · check-permissions.mjs on correct and broken files
UPDATED	November 19, 2026
DOCS	code.claude.com/docs/en/permissions
ARTICLE	ecomlabs.dev/g/permisos-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	How rules decide
Step 01	The project settings.json, explained
Step 02	The local file and the user file
Step 03	The check script
Step 04	Confirm the rules apply
Checklist	Before you commit

BEFORE YOU START

How rules decide

- Every rule is `Tool` or `Tool(specifier)` and goes in `allow`, `ask` or `deny`. They are checked like this: deny first, then ask, then allow. The first match decides, even if another rule is more specific.
- Lists from every file add up, and a deny in any of them blocks. The files, strongest first: managed, `claude --settings`, `.claude/settings.local.json`, `.claude/settings.json`, `~/.claude/settings.json`.
- If no rule matches, the permission mode decides. Deny blocks in every mode; ask prompts even in `auto` and `bypassPermissions`.

STEP 01

The project settings.json, explained

Save it as `.claude/settings.json` at the project root and commit it. It is strict JSON, with no comments or trailing commas, so the comments live in the table. Validated with Ajv against the published schema (`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 *)"
    ]
  }
}
```

RULE	WHAT IT DOES AND WHEN TO CHANGE IT
`Bash(npm run lint)` and the other three exact rules	Run without a prompt and without extra arguments. Use the names in your `package.json`.
`Bash(npm run test *)`	`npm run test` with arguments. It does not cover `npm run test:e2e`: the space is part of the rule.
`Bash(git commit *)`, `Bash(git push *)`	Prompt before every commit and push, in auto mode too.
`Bash(npm install *)`, `Bash(npm i *)`	Two rules because Bash rules compare text and know nothing about aliases.
`Bash(rm *)`, `Bash(curl *)`	Prompt instead of blocking: deleting `.next` or hitting `localhost` is normal.
`Read(**/.env)`, `Read(**/.env.*)`, `Read(!.env.example)`	Block reading, editing and creating `.env` and its variants at any depth; the `!` lets Claude read the template.
`Read(secrets/**)`, `Read(**/*.pem)`	Any `secrets` folder and any key or certificate. Add your `*.key` files.
`Bash(git push --force *)`, `Bash(npm publish *)`	Deny comes before ask. `git push origin main --force` does not match and falls into ask. A published version cannot be replaced.

With pnpm or yarn, write the command the way your team types it: `Bash(pnpm lint)` and `Bash(pnpm run lint)` are two different rules. In a monorepo, `Read(**/.env)` already covers `apps/web/.env`.

STEP 02

The local file and the user file

`.claude/settings.local.json` holds rules that only make sense on your machine, such as an MCP server registered just for you:

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

- `npx` runs whatever you pass it: name the inner command too. `Bash(npx *)` would allow any package.
- `mcp__` rules take no parentheses. If you create the file by hand, add it to `.gitignore`.
- An allow here does not beat an ask from the project.

`~/.claude/settings.json` protects what lives outside your projects. Use `~/`: with a single `/`, the path would point at `~/.claude`. It is the only one of your files where `"auto"` works as `defaultMode`:

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

STEP 03

The check script

`check-permissions.mjs` has no dependencies and does not evaluate rules: it flags the traps the documentation describes and the schema does not catch, such as `Write(.env)`, `Bash(git:* push)` or `defaultMode` set to `auto` inside the project.

```
// 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;
```

Run it from the project root: `node check-permissions.mjs .claude/settings.json .claude/settings.local.json`. It exits with code 1 if it flags anything.

STEP 04

Confirm the rules apply

1. Open Claude Code at the project root and accept the trust dialog: without it, the project's allow rules do not apply.
2. `/status`: the *Setting sources* line lists the loaded files. `/permissions`: every rule with its file. `claude doctor`: rejected entries.
3. Ask "read .env". The read should be refused.
4. Ask "run npm run lint". It should run without a prompt.
5. Ask "run git push --dry-run". It should prompt you, in auto mode too.

A Bash rule compares the command text: it does not stop `/usr/bin/curl` or `sh -c`. A `Read` deny does not reach a script that opens the file itself. That is what the sandbox and `PreToolUse` hooks are for.

CHECKLIST

Before you commit

- [ ] `.claude/settings.json` has `$schema`, all three lists and the script names from your `package.json`.
- [ ] No allow rule uses a bare `Bash` or an asterisk before the subcommand.
- [ ] `ask` covers pushes, commits, dependencies, deletions and network calls; `deny` covers `.env`, keys and `npm publish`.
- [ ] `.claude/settings.local.json` stays out of git.
- [ ] `defaultMode` set to `auto`, if you want it, only in `~/.claude/settings.json`.
- [ ] `check-permissions.mjs` reports OK and the Step 04 tests give the expected result.