BUILD WITH CLAUDE CODE

Get started with Claude Code hooks

Copy page

Learn how to customize and extend Claude Code's behavior by registering shell commands

Claude Code hooks are user-defined shell commands that execute at various points in Claude Code's lifecycle. Hooks provide deterministic control over Claude Code's behavior, ensuring certain actions always happen rather than relying on the LLM to choose to run them.

For reference documentation on hooks, see **Hooks reference**.

Example use cases for hooks include:

- **Notifications**: Customize how you get notified when Claude Code is awaiting your input or permission to run something.
- **Automatic formatting**: Run `prettier` on .ts files, `gofmt` on .go files, etc. after every file edit.
- **Logging**: Track and count all executed commands for compliance or debugging.
- **Feedback**: Provide automated feedback when Claude Code produces code that does not follow your codebase conventions.
- **Custom permissions**: Block modifications to production files or sensitive directories.

By encoding these rules as hooks rather than prompting instructions, you turn suggestions into app-level code that executes every time it is expected to run.

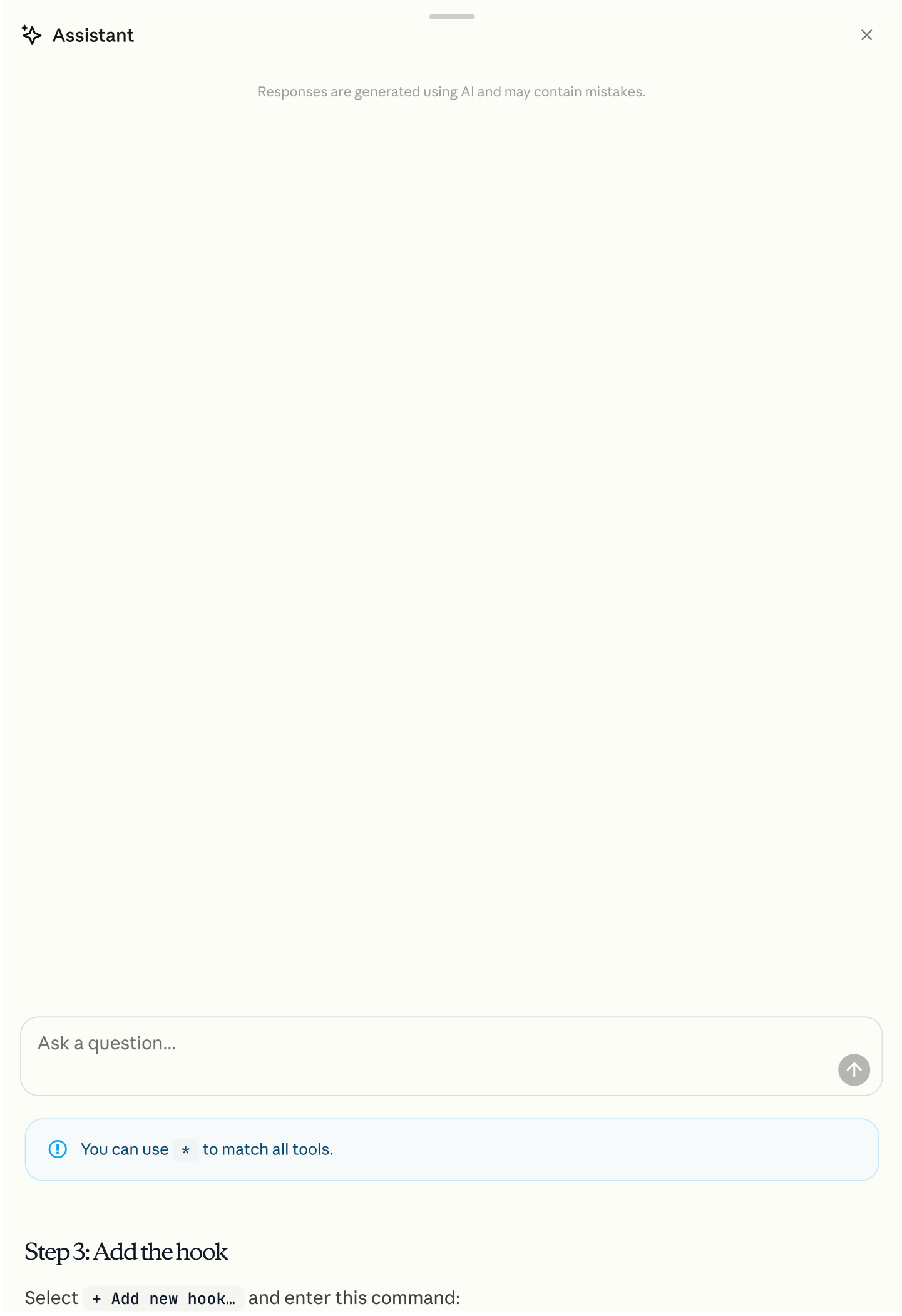
You must consider the security implication of hooks as you add them, because hooks run automatically during the agent loop with your current environment's credentials. For example, malicious hooks code can exfiltrate your data. Always review your hooks implementation before registering them.

For full security best practices, see **Security Considerations** in the hooks reference documentation.

Hook Events Overview

Claude Code provides several hook events that run at different points in the workflow:

- **PreToolUse**: Runs before tool calls (can block them)



Step 3: Add the hook

Select `+ Add new hook…` and enter this command:

Assistant

Responses are generated using AI and may contain mistakes.

You should see entries like:

```
ls - Lists files and directories
```

Assistant

Responses are generated using AI and may contain mistakes.

Assistant

Responses are generated using AI and may contain mistakes.

Assistant

Responses are generated using AI and may contain mistakes.

```
    return 'text'

def format_markdown(content):
    """Format markdown content with language detection."""
    # Fix unlabeled code fences
    def add_lang_to_fence(match):
```

Assistant

Responses are generated using AI and may contain mistakes.

```
chmod +x .claude/hooks/markdown_formatter.py
```

This hook automatically:

- Detects programming languages in unlabeled code blocks

Assistant

Responses are generated using AI and may contain mistakes.

Assistant

Responses are generated using AI and may contain mistakes.

Assistant

Responses are generated using AI and may contain mistakes.