

Custom commands in practice

Your tailor-made /command in three examples: the deploy that runs only when you type it, the parametric fix-issue, the multi-argument migration.

Updated: July 2026 · Source: official Anthropic documentation (code.claude.com/docs) · Vol. 11 (extra) of the series — skills in vol. 9.

1. Commands = skills, today

the merge you need to know

Custom commands have been **merged into skills**: `.claude/commands/deploy.md` and `.claude/skills/deploy/SKILL.md` both create `/deploy` and work the same way. Files in `.claude/commands/` still count (file name without extension = command name) and support the same frontmatter; on a name clash the skill wins. Prefer the skill form for the extra features: a directory of supporting files, control over who invokes it, automatic loading when relevant. This volume covers the "command" case: **workflows you launch yourself**, with arguments.

Frontmatter	You can invoke it	Claude can invoke it	When in context
(default)	Yes	Yes	Description always; body on invocation
<code>disable-model-invocation: true</code>	Yes	No	Nothing in context until you launch it
<code>user-invocable: false</code>	No	Yes	Description always; body on invocation

2. Example 1: /deploy — the button only you press

disable-model-invocation

```
# .claude/skills/deploy/SKILL.md
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---

Deploy $ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deploy target
4. Verify the deploy succeeded
```

\$ARGUMENTS: the text after the command

- `/deploy api-gateway` → the body reaches Claude with `$ARGUMENTS` replaced by `api-gateway`.
- If the body doesn't contain `$ARGUMENTS` but you pass arguments, Claude Code still appends them as `ARGUMENTS: <text>`: nothing gets lost.
- For a literal `$` before digits or ARGUMENTS, escape it: `\$1.00`.

Pre-approving the workflow's tools

```
---
name: commit
description: Stage and commit the changes
disable-model-invocation: true
allowed-tools: Bash(git add *)
               Bash(git commit *) Bash(git status *)
---
```

While the skill is active, the listed git commands don't ask for confirmation: the workflow flows without prompts, everything else follows the normal permissions.

Why block Claude

`disable-model-invocation: true` = only you can launch it. It's the rule for anything with **side effects** or whose timing you want to decide: `/deploy`, `/commit`, `/send-slack-message`. You don't want Claude deploying because the code "looks ready to it". Bonus: the skill also drops out of context (no description hanging around) until you invoke it.

3. Example 2: /fix-issue 123

one argument, one workflow

```
# .claude/skills/fix-issue/SKILL.md
---
name: fix-issue
description: Fixes a GitHub issue
argument-hint: [issue-number]
disable-model-invocation: true
---
```

Fix GitHub issue \$ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write the tests
5. Create a commit

The details that improve usage

- `/fix-issue 123` → Claude receives "Fix GitHub issue 123..." and starts the workflow.
- `argument-hint` shows up in the `/` menu's autocomplete: users know what to pass without opening the file.
- The body is a **numbered procedure**, not a description: for task-commands, step-by-step instructions beat prose.
- They can be **stacked**: since v2.1.199 `/code-review /fix-issue 123` loads both skills and passes `123` as \$ARGUMENTS to each (up to 6 in one message).

4. Example 3: multiple positional arguments

\$0 \$1 \$2 or telling names

```
# .claude/skills/migrate-component/SKILL.md
---
name: migrate-component
description: Migrates a component across frameworks
---
```

Migrate the component \$0 from \$1 to \$2.
Preserve existing behavior and tests.

```
# /migrate-component SearchBar React Vue
# $0=SearchBar $1=React $2=Vue
# Shell-style quoting:
# /my-skill "hello world" second
# $0=hello world $1=second
```

```
# Variant with named arguments:
# names map to positions in order
---
```

```
name: migrate-component
description: Migrates a component across frameworks
arguments: [component, from, to]
---
```

Migrate the component \$component from \$from to \$to.
Preserve existing behavior and tests.

Which form to choose

`$0 $1 $2` (or the equivalent `$ARGUMENTS[0]` ...) is fine for 2-3 obvious arguments; with `arguments:` in the frontmatter the body reads itself — `$component` says what it is. `$ARGUMENTS` is always still the whole string as typed.

5. Substitutions and dynamic context

the full toolbox

Placeholder	Expands to
<code>\$ARGUMENTS</code>	All arguments, as typed.
<code>\$ARGUMENTS[N]</code> / <code>\$N</code>	The argument at position N (0-based). Shell-style quoting for multi-word values.
<code>\$name</code>	Named argument declared in <code>arguments:</code> (maps positions in order).
<code>\${CLAUDE_SESSION_ID}</code>	Current session ID: logs, per-session files, correlation.
<code>\${CLAUDE_EFFORT}</code>	Active effort (<code>low...max</code>): adaptive instructions.
<code>\${CLAUDE_SKILL_DIR}</code>	The skill's directory: paths to bundled scripts, wherever it's installed.
<code>\${CLAUDE_PROJECT_DIR}</code>	Project root (the same path hooks receive). Also works inside <code>allowed-tools</code> .
<code>!`command` / `` `! block</code>	Dynamic injection: the command runs before Claude reads, its output replaces the line (vol. 9, ex. 1). Only with <code>!</code> at line start or after a space. Policy kill switch: <code>disableSkillShellExecution</code> .

► Governing commands with permissions

Rules in permissions: `Skill(deploy)` exact match, `Skill(review-pr *)` prefix with any arguments, `Skill` in deny turns off all Claude invocations. For skills you don't want to edit (shared repos): `skill0verrides` in settings with four states — `on` / `name-only` (only the name in context) / `user-invocable-only` / `off` — from the `/skills` menu with the space bar. And remember: a project skill's `allowed-tools` only counts after you've trusted the workspace — read a cloned repo's skills before trusting it.

Sources: code.claude.com/docs — skills (custom commands, string substitutions, invocation control, restrict skill access) · Vol. 9 for skills with supporting files, vol. 1 for the built-in commands.