

Comandi custom in pratica

Il tuo /comando su misura in tre esempi: il deploy che parte solo se lo digiti tu, il fix-issue parametrico, la migrazione con più argomenti.

Aggiornato: luglio 2026 · Fonte: documentazione ufficiale Anthropic (code.claude.com/docs) · Vol. 11 (extra) della serie — skills nel vol. 9.

1. Comandi = skill, oggi

la fusione da sapere

I custom command sono stati **fusi nelle skill**: `.claude/commands/deploy.md` e `.claude/skills/deploy/SKILL.md` creano entrambi `/deploy` e funzionano allo stesso modo. I file in `.claude/commands/` continuano a valere (nome file senza estensione = nome comando) e supportano lo stesso frontmatter; a parità di nome vince la skill. Preferisci la forma skill per le feature in più: directory di file di supporto, controllo di chi la invoca, caricamento automatico quando rilevante. Questo volume copre il caso "comando": **workflow che lanci tu**, con argomenti.

Frontmatter	Tu puoi invocarla	Claude può invocarla	Quando in contesto
(default)	Sì	Sì	Description sempre; corpo all'invocazione
<code>disable-model-invocation: true</code>	Sì	No	Niente in contesto finché non la lanci tu
<code>user-invocable: false</code>	No	Sì	Description sempre; corpo all'invocazione

2. Esempio 1: /deploy — il bottone che premi solo tu

disable-model-invocation

```
# .claude/skills/deploy/SKILL.md
---
name: deploy
description: Deploy dell'applicazione in produzione
disable-model-invocation: true
---
```

Fai il deploy di \$ARGUMENTS in produzione:

1. Lancia la suite di test
2. Builda l'applicazione
3. Push verso il target di deploy
4. Verifica che il deploy sia riuscito

Perché bloccare Claude

`disable-model-invocation: true` = solo tu puoi lanciairla. È la regola per tutto ciò che ha **effetti collaterali** o di cui vuoi decidere il momento: `/deploy`, `/commit`, `/send-slack-message`. Non vuoi che Claude deployi perché il codice "gli sembra pronto". Bonus: la skill esce anche dal contesto (niente description in giro) finché non la invochi.

\$ARGUMENTS: il testo dopo il comando

- `/deploy api-gateway` → il corpo arriva a Claude con `$ARGUMENTS` sostituito da `api-gateway`.
- Se il corpo non contiene `$ARGUMENTS` ma tu passi argomenti, Claude Code li accoda comunque come `ARGUMENTS: <testo>`: niente va perso.
- Per un `$` letterale davanti a cifre o ad ARGUMENTS, escapa: `\$1.00`.

Pre-approvare i tool del workflow

```
---
name: commit
description: Stage e commit delle modifiche
disable-model-invocation: true
allowed-tools: Bash(git add *)
               Bash(git commit *) Bash(git status *)
---
```

Mentre la skill è attiva, i comandi git elencati non chiedono conferma: il workflow fila senza prompt, tutto il resto segue i permessi normali.

3. Esempio 2: /fix-issue 123

un argomento, un workflow

```
# .claude/skills/fix-issue/SKILL.md
---
name: fix-issue
description: Corregge una issue GitHub
argument-hint: [numero-issue]
disable-model-invocation: true
---
```

Correggi la issue GitHub \$ARGUMENTS seguendo i nostri standard di codice.

1. Leggi la descrizione della issue
2. Capisci i requisiti
3. Implementa la correzione
4. Scrivi i test
5. Crea un commit

I dettagli che migliorano l'uso

- `/fix-issue 123` → Claude riceve "Correggi la issue GitHub 123..." e parte col workflow.
- `argument-hint` compare nell'autocomplete del menu `/`: chi lo usa sa cosa passare senza aprire il file.
- Il corpo è una **procedura numerata**, non una descrizione: per i task-command le istruzioni passo-passo battono la prosa.
- Si può **impilare**: da v2.1.199 `/code-review /fix-issue 123` carica entrambe le skill e passa `123` come \$ARGUMENTS a ciascuna (fino a 6 in un messaggio).

4. Esempio 3: più argomenti posizionali

\$0 \$1 \$2 o nomi parlanti

```
# .claude/skills/migrate-component/SKILL.md
---
name: migrate-component
description: Migra un componente tra framework
---
```

Migra il componente \$0 da \$1 a \$2.
Preserva comportamento e test esistenti.

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

```
# Variante con argomenti nominati:
# i nomi mappano le posizioni in ordine
---
name: migrate-component
description: Migra un componente tra framework
arguments: [componente, da, a]
---
```

Migra il componente \$componente da \$da a \$a.
Preserva comportamento e test esistenti.

Quale forma scegliere

`$0 $1 $2` (o l'equivalente `$ARGUMENTS[0]` ...) va bene per 2-3 argomenti ovvi; con `arguments:` in frontmatter il corpo si legge da solo — `$componente` dice cosa è. `$ARGUMENTS` resta sempre l'intera stringa come digitata.

5. Sostituzioni e contesto dinamico

il toolbox completo

Placeholder	Si espande in
<code>\$ARGUMENTS</code>	Tutti gli argomenti, come digitati.
<code>\$ARGUMENTS[N]</code> / <code>\$N</code>	L'argomento in posizione N (0-based). Quoting shell-style per valori multi-parola.
<code>\$nome</code>	Argomento nominato dichiarato in <code>arguments:</code> (mappa le posizioni in ordine).
<code>\${CLAUDE_SESSION_ID}</code>	ID della sessione corrente: log, file per-sessione, correlazioni.
<code>\${CLAUDE_EFFORT}</code>	Effort attivo (<code>low ... max</code>): istruzioni adattive.
<code>\${CLAUDE_SKILL_DIR}</code>	Directory della skill: path a script inclusi, ovunque sia installata.
<code>\${CLAUDE_PROJECT_DIR}</code>	Root del progetto (lo stesso path che ricevono gli hook). Vale anche dentro <code>allowed-tools</code> .
<code>!`comando` / blocco ``!</code>	Injection dinamica: il comando gira prima che Claude legga, l'output sostituisce la riga (vol. 9, es. 1). Solo con <code>!</code> a inizio riga o dopo spazio. Kill switch di policy: <code>disableSkillShellExecution</code> .

► Governare i comandi coi permessi

Regole nei permessi: `Skill(deploy)` match esatto, `Skill(review-pr *)` prefisso con argomenti qualsiasi, `Skill` in deny spegne tutte le invocazioni di Claude. Per le skill che non vuoi editare (repo condivisi): `skillOverrides` in settings con quattro stati — `on` / `name-only` (solo il nome in contesto) / `user-invocable-only` / `off` — dal menu `/skills` con la barra spaziatrice. E ricorda: `allowed-tools` di una skill di progetto vale solo dopo che ti sei fidato del workspace — leggi le skill di un repo clonato prima di fidarti.

Fonti: code.claude.com/docs — skills (custom commands, string substitutions, invocation control, restrict skill access) · Vol. 9 per skills con file di supporto, vol. 1 per i comandi built-in.