

Subagents in pratica

Tre subagenti completi da copiare: il code reviewer read-only, il lettore di database che non può scrivere, il tester con il browser tutto suo.

Aggiornato: luglio 2026 · Fonte: documentazione ufficiale Anthropic (code.claude.com/docs) · Vol. 10 (extra) della serie — teoria nel vol. 2.

1. L'anatomia in 60 secondi

un file markdown, il corpo è il system prompt

Un subagente è un file markdown con frontmatter YAML: solo `name` e `description` sono obbligatori, il **corpo diventa il suo system prompt**. Gira in un contesto tutto suo: non vede la tua conversazione e ti restituisce solo il risultato — l'output verboso resta da lui. Da v2.1.198 `/agents` non apre più il wizard: **chiedi a Claude di scrivere il file** o crealo a mano; le directory sono osservate, file nuovo o modificato = attivo in pochi secondi.

Dove	Vale per	Priorità a parità di nome
<code>managed settings</code>	Organizzazione	1 (vince su tutto)
<code>--agents '<json>'</code>	Solo quella sessione	2 — per test rapidi e script, niente file su disco
<code>.claude/agents/</code>	Il progetto	3 — committalo: lo usa e migliora il team
<code>~/.claude/agents/</code>	Tutti i tuoi progetti	4
<code><plugin>/agents/</code>	Dove il plugin è attivo	5 (nome scoped <code>plugin:nome</code>)

2. Esempio 1: il code reviewer

read-only per costruzione

```
# .claude/agents/code-reviewer.md
---
name: code-reviewer
description: Review esperta di qualità, sicurezza e manutenibilità. Usalo proattivamente subito dopo aver scritto o modificato codice.
tools: Read, Grep, Glob, Bash
model: inherit
---
```

Sei un senior code reviewer. Appena invocato:
1. Esegui git diff per vedere le modifiche recenti
2. Concentrati sui file modificati e inizia subito

Checklist: codice chiaro; nomi parlanti; niente duplicazione; error handling; nessun segreto esposto; input validati; test adeguati.

Feedback per priorità: critici / warning / suggerimenti, con esempi di come correggere.

Le scelte che contano

- `tools` è una **allowlist**: senza Edit e Write il reviewer non può toccare il codice, per costruzione (l'alternativa `disallowedTools` eredita tutto tranne quelli).
- "**Usalo proattivamente**" nella description spinge Claude a delegare da solo; il corpo dà workflow e formato di output — i subagenti vaghi restituiscono risultati vaghi.

Tre modi per invocarlo

- Naturale: *"usa il subagente code-reviewer sulle mie ultime modifiche"* — Claude decide.
- @-mention: digita `@agent-code-reviewer` — garantisce che giri proprio lui.
- Sessione intera: `claude --agent code-reviewer` — tutta la sessione col suo prompt, tool e modello.
- Bonus: con `memory: project` il reviewer **impara tra le sessioni** (`.claude/agent-memory/`, condivisibile via git).

3. Esempio 2: Bash sì, ma solo SELECT

quando tools non basta: hook PreToolUse

```
# .claude/agents/db-reader.md
---
name: db-reader
description: Esegue query read-only sul database. Usalo per analisi dati e report.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
```

```
# scripts/validate-readonly-query.sh
# (chmod +x) — blocca le scritture SQL
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r \
  '.tool_input.command // empty')

if echo "$COMMAND" | grep -iE \
  '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE)\b' \;
```

```
- type: command
  command: "./scripts/validate-readonly-
query.sh"
---
```

Sei un analista con accesso in sola lettura. Rispondi con query SELECT efficienti. Se ti chiedono di modificare dati, spiega che hai solo accesso in lettura.

```
> /dev/null; then
echo "Bloccato: solo query SELECT" >&2
exit 2
fi
exit 0
```

Il meccanismo

Il campo `tools` è tutto-o-niente: qui serve Bash, ma *solo* certi comandi. L'hook `PreToolUse` riceve la chiamata come JSON su stdin prima che esegua: `exit 2` la blocca, lo stderr torna a Claude. Determinismo, non speranza (vol. 2).

4. Esempio 3: il browser solo suo

mcpServers
inline

```
# .claude/agents/browser-tester.md
---
name: browser-tester
description: Testa le feature in un browser
  reale con Playwright.
mcpServers:
  # inline: esiste solo per questo subagente
  - playwright:
      type: stdio
      command: npx
      args: ["-y", "@playwright/mcp@latest"]
  - github # per nome: riusa un server già
  --- # configurato nella sessione
```

Usa i tool Playwright per navigare, fare screenshot e interagire con le pagine.

Perché inline

- Il server parte col subagente e si spegne quando finisce: **il main non vede mai quel tool** né ne paga le definizioni.
- Voci inline con lo stesso schema di `.mcp.json`; i riferimenti per nome (`- github`) riusano la connessione della sessione madre.
- Per toglierli invece: `disallowedTools: mcp_github` (tutto quel server) o `mcp_*` (tutti gli MCP).

Foreground, background, resume

- Da v2.1.198 girano in **background per default**; i permessi affiorano comunque da te. `Ctrl+B` = background per un task in corso.
- Ogni invocazione riparte da zero; per continuare: *"continua quella review, ora l'autorization"* — riprende lo stesso subagente con tutta la sua storia.

5. Frontmatter di riferimento

solo name e description
obbligatori

Campo	Effetto
<code>tools</code> / <code>disallowedTools</code>	Allowlist / denylist di tool; pattern MCP a livello server (<code>mcp_github</code>). Insieme: prima il deny, poi l'allow sul resto.
<code>model</code> / <code>effort</code>	<code>sonnet</code> , <code>opus</code> , <code>haiku</code> , <code>fable</code> , ID completo o <code>inherit</code> (default). Effort: <code>low</code> ... <code>max</code> .
<code>permissionMode</code>	<code>default</code> / <code>acceptEdits</code> / <code>auto</code> / <code>dontAsk</code> / <code>bypassPermissions</code> / <code>plan</code> ; in <code>bypass/acceptEdits/auto</code> vince il padre.
<code>skills</code> / <code>memory</code>	Skill precaricate per intero all'avvio / memoria persistente tra le sessioni (<code>user</code> / <code>project</code> / <code>local</code> in <code>agent-memory/</code> ; consigliato <code>project</code>).
<code>isolation: worktree</code>	Git worktree temporaneo: copia isolata del repo, ripulita se non tocca nulla.
<code>background</code> / <code>maxTurns</code> / <code>color</code> / <code>hooks</code> / <code>mcpServers</code>	Sempre in background / tetto di turni / colore nel transcript / hook e server MCP dedicati (es. 3 e 4; ignorati nei subagenti da plugin).

► I tre pattern che ripagano subito

1. Isolare il verboso: *"usa un subagente per i test e riportami solo i falliti"*. **2. Ricerca parallela:** *"esplora auth, database e API in parallelo con subagenti separati"*. **3. Catena:** *"code-reviewer trova i problemi, poi l'optimizer li corregge"*. Regola d'oro: subagente quando il lavoro è auto-contenuto e torna come riassunto; conversazione principale per il botta e risposta; `/btw` per una domanda al volo.