Subagents in practice ⚙

Three complete subagents to copy: the read-only code reviewer, the database reader that cannot write, the tester with a browser of its own.

Updated: July 2026 · Source: official Anthropic documentation (code.claude.com/docs) · Vol. 10 (extra) of the series — theory in vol. 2.

1. The anatomy in 60 seconds

one markdown file, the body is the system prompt

A subagent is a markdown file with YAML frontmatter: only `name` and `description` are required, the **body becomes its system prompt**. It runs in a context of its own: it doesn't see your conversation and returns only the result — the verbose output stays with it. Since v2.1.198 `/agents` no longer opens the wizard: **ask Claude to write the file** or create it by hand; the directories are watched, new or modified file = active within seconds.

Where	Applies to	Priority on name conflict
`managed settings`	Organization	1 (wins over everything)
`--agents '<json>'`	That session only	2 — for quick tests and scripts, no file on disk
`.claude/agents/`	The project	3 — commit it: the team uses and improves it
`~/.claude/agents/`	All your projects	4
`<plugin>/agents/`	Where the plugin is active	5 (scoped name `plugin:name`)

2. Example 1: the code reviewer

read-only by construction

```
# .claude/agents/code-reviewer.md
---
name: code-reviewer
description: Expert review of quality, security
  and maintainability. Use it proactively right
  after writing or modifying code.
tools: Read, Grep, Glob, Bash
model: inherit
---

You are a senior code reviewer. When invoked:
1. Run git diff to see the recent changes
2. Focus on the modified files, start right away

Checklist: clear code; meaningful names; no
duplication; error handling; no exposed
secrets; validated inputs; adequate tests.

Feedback by priority: critical / warnings /
suggestions, with examples of how to fix.
```

The choices that matter

- `tools` is an **allowlist**: without Edit and Write the reviewer cannot touch the code, by construction (the alternative `disallowedTools` inherits everything except those).
- "**Use it proactively**" in the description pushes Claude to delegate on its own; the body sets workflow and output format — vague subagents return vague results.

Three ways to invoke it

- Natural: "*use the code-reviewer subagent on my latest changes*" — Claude decides.
- **@-mention**: type `@agent-code-reviewer` — guarantees that exact agent runs.
- Whole session: `claude --agent code-reviewer` — the entire session with its prompt, tools and model.
- Bonus: with `memory: project` the reviewer **learns across sessions** (`.claude/agent-memory/`, shareable via git).

3. Example 2: Bash yes, but SELECT only

when tools isn't enough: PreToolUse hook

```
# .claude/agents/db-reader.md
---
name: db-reader
description: Runs read-only queries on the database.
  Use it for data analysis and reports.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
```

```
# scripts/validate-readonly-query.sh
# (chmod +x) — blocks SQL writes
#!/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' \
  > /dev/null; then
```

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

You are an analyst with read-only access.
Answer with efficient SELECT queries. If
asked to modify data, explain that you
only have read access.
```

```
  echo "Blocked: SELECT queries only" >&2
  exit 2
fi
exit 0
```

The mechanism

The `tools` field is all-or-nothing: here you need Bash, but *only certain* commands. The `PreToolUse` hook receives the call as JSON on stdin before it runs: `exit 2` blocks it, the stderr goes back to Claude. Determinism, not hope (vol. 2).

4. Example 3: a browser of its own

inline mcpServers

```
# .claude/agents/browser-tester.md
---
name: browser-tester
description: Tests features in a real
  browser with Playwright.
mcpServers:
  # inline: exists only for this subagent
  - playwright:
      type: stdio
      command: npx
      args: ["-y", "@playwright/mcp@latest"]
  - github    # by name: reuses a server already
---           # configured in the session

Use the Playwright tools to navigate, take
screenshots and interact with pages.
```

Why inline

- The server starts with the subagent and shuts down when it finishes: **the main never sees those tools** nor pays for their definitions.
- Inline entries use the same schema as `.mcp.json`; by-name references (`- github`) reuse the parent session's connection.
- To remove them instead: `disallowedTools: mcp__github` (that whole server) or `mcp__*` (all MCPs).

Foreground, background, resume

- Since v2.1.198 they run in the **background by default**; permissions still surface to you. `Ctrl+B` = background a running task.
- Each invocation starts from scratch; to continue: "*continue that review, now authorization*" — resumes the same subagent with all its history.

5. Frontmatter reference

only name and description required

Field	Effect
`tools / disallowedTools`	Tool allowlist / denylist; MCP patterns at server level (`mcp__github`). Together: deny first, then allow on the rest.
`model / effort`	`sonnet`, `opus`, `haiku`, `fable`, full ID or `inherit` (default). Effort: `low`…`max`.
`permissionMode`	`default` / `acceptEdits` / `auto` / `dontAsk` / `bypassPermissions` / `plan`; in bypass/acceptEdits/auto the parent wins.
`skills / memory`	Skills **preloaded** in full at startup / persistent memory across sessions (`user`/`project`/`local` in `agent-memory/`; `project` recommended).
`isolation: worktree`	Temporary git worktree: isolated copy of the repo, cleaned up if untouched.
`background / maxTurns / color / hooks / mcpServers`	Always in background / turn cap / color in the transcript / dedicated hooks and MCP servers (ex. 3 and 4; ignored in plugin subagents).

▶ The three patterns that pay off immediately

1. Isolate the verbose: "*use a subagent for the tests and report only the failures*". **2. Parallel research:** "*explore auth, database and API in parallel with separate subagents*". **3. Chain:** "*code-reviewer finds the issues, then the optimizer fixes them*". Golden rule: subagent when the work is self-contained and comes back as a summary; main conversation for the back-and-forth; `/btw` for a quick question.

Sources: code.claude.com/docs — sub-agents · Ecosystem: vol. 2 (skills/hooks), vol. 6 (parallel).