CLAUDE CODE

Introduction to *Coding Agents*

FABIAN ROHLIK · CBC

The *plan*

01 What is agentic coding?

02 Claude Code

03 Context

04 Advanced Claude Code

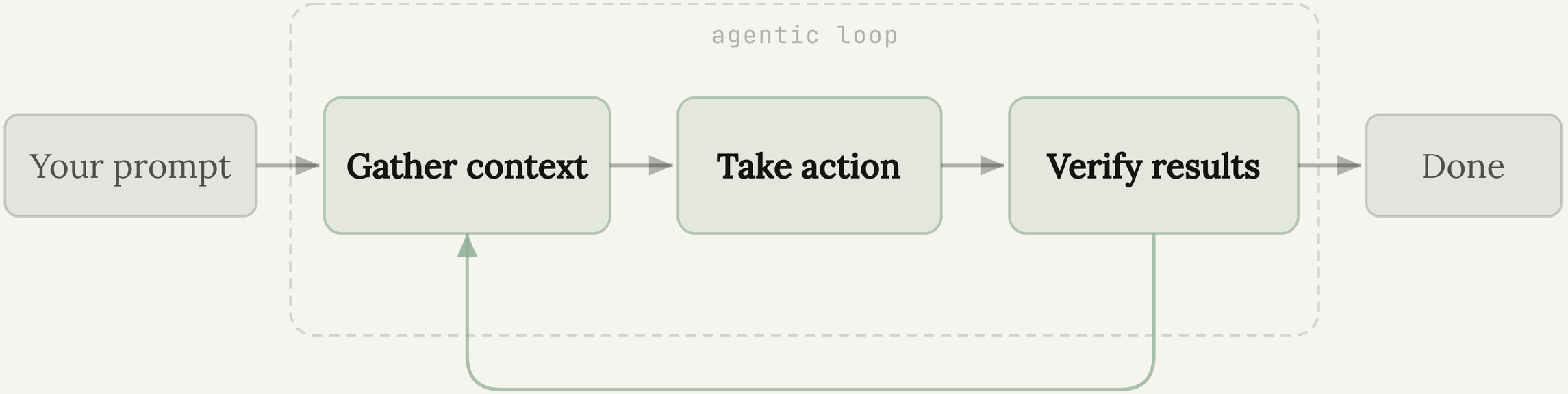
02

What is *agentic coding?*

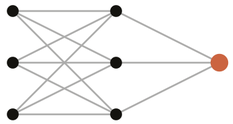
The length of tasks AIs can do is *doubling every 7 months*

Task length (at 50% success rate)

Think.
Act.
Observe.
Repeat.

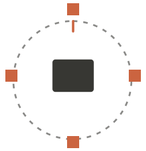

agentic loop
Your prompt
Gather context
Take action
Verify results
Done

Models, wrappers, and *interfaces*



The model

`Claude Opus 4.7`, `GPT-5.5`, `Gemini 3.5`.



The agent / wrapper

Adds the loop, tools, etc. Claude Code, Codex, OpenCode, OpenClaw, ...

The interface

CLI, IDE Integration, Application, etc.

03

Claude *Code*

The *terminal*

01 A text-based interface to your computer, you type commands, it runs them. No icons, no windows.

02 Direct access to files, processes, and the network. Everything Finder, TextEdit and co. do, but more.

Install & *set up*

MACOS · LINUX · WSL

```
$ curl -fsSL https://claude.ai/install.sh | bash
```

WINDOWS POWERSHELL

```
> irm https://claude.ai/install.ps1 | iex
```

WINDOWS CMD

```
> curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd
```

→ code.claude.com/docs/

Claude Code is *not* *just* for writing code

1. Discover	2. Design	3. Build	4. Deploy	5. Support & Scale
Explore codebase and history	Plan project	Implement code	Automate CI/CD	Debug errors
Search documentation	Develop tech specs	Write and execute tests	Configure environments	Large-scale refactor
Onboard & setup	Define architecture	Create commits and PRs	Manage deployments	Monitor usage & performance

Tip #1:

use codebase Q&A as a way to dip your feet into Claude Code

Tip #2:

practice prompting, and start to understand what Claude Code “gets” immediately vs. what needs more specific instructions

Permissions, modes & *shortcuts*

`Shift + Tab to change mode`

Plan

Agent outputs a written plan to approve.

Default

Asks for permission to edit things.

Accept-edits

Changes files without asking for permission. Needs permission for other actions.

```
--dangerously-skip-permissions
```

Commands

Esc

Press `Esc` to interrupt the agent mid-task. It stops immediately and waits for your next instruction.

@ files

Drop a file into the conversation explicitly. Claude reads it before reasoning.

/commands

clear, model, effort, init, usage, btw, etc.

Tip #3:

interrupt and course-correct as soon as you think you notice it's going off track — use `Esc`, `/rewind`, and continue

First *prompt*

Choose one track. Get Claude Code to produce something for the first time.

```
$ cd your-project && claude
```

Pick a track

A **I have a codebase**

Ask: "Check out this repo and explain the structure."

B **I do not have code**

Ask: “Create an HTML website about [something you care about]”

04 Context

What is *context*?

Everything the model can *see* while it's deciding what to do next.

Your prompt

The message you just typed.

Conversation history

Everything said earlier in the session.

Files & tools

Code it has read, command output, search results, project-level instructions.

Tip #4:

manage context aggressively — use `/clear`, `/compact`, `/rewind` wisely; run 2 separate Claude Code instances for 2 features; don't pollute chat history with wrong paths; use `/btw` for side questions

PROMPTING CLAUDE CODE

Show the full picture

> **Write tests** for autils/markdown.ts to make sure links render properly (note the tests **won't pass yet**, since links aren't yet implemented). Then **commit**. Then **update the code to make the tests pass**.

> **Implement [mock.png]**. Then **screenshot it with Puppeteer** and **iterate till it looks like the mock**.

> **commit, push, pr**

Agent *Skills*

Skills are prompts that extend Claude Code. Claude decides when to activate them based on your request and the Skill's description.

```
# .claude/skills/code-reviewer/SKILL.md
---
name: code-reviewer
description: Review code for best practices
  and potential issues. Use when reviewing
  code, checking PRs, or analyzing code
  quality.
---

# Code Reviewer
Review files for style, security, and
performance.
```

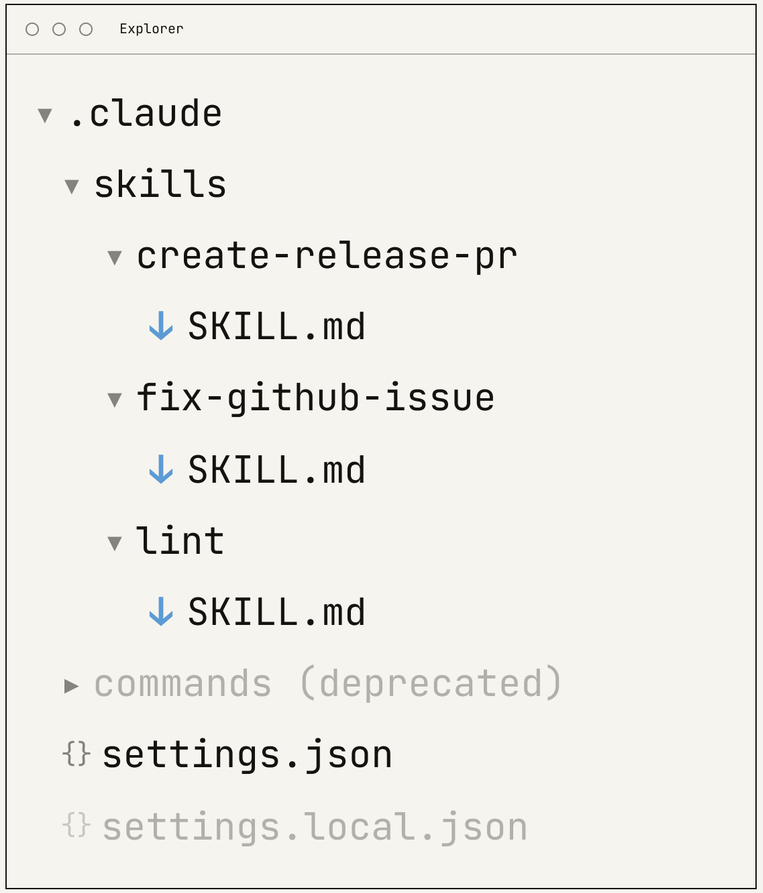
Global Skills – ~/.claude/skills/

Project Skills – .claude/skills/

Custom *slash commands*

How it works now

`.claude/commands/` is deprecated — don't add new shortcuts there. Implement each shortcut as an **Agent Skill**.



THE README FOR CLAUDE CODE

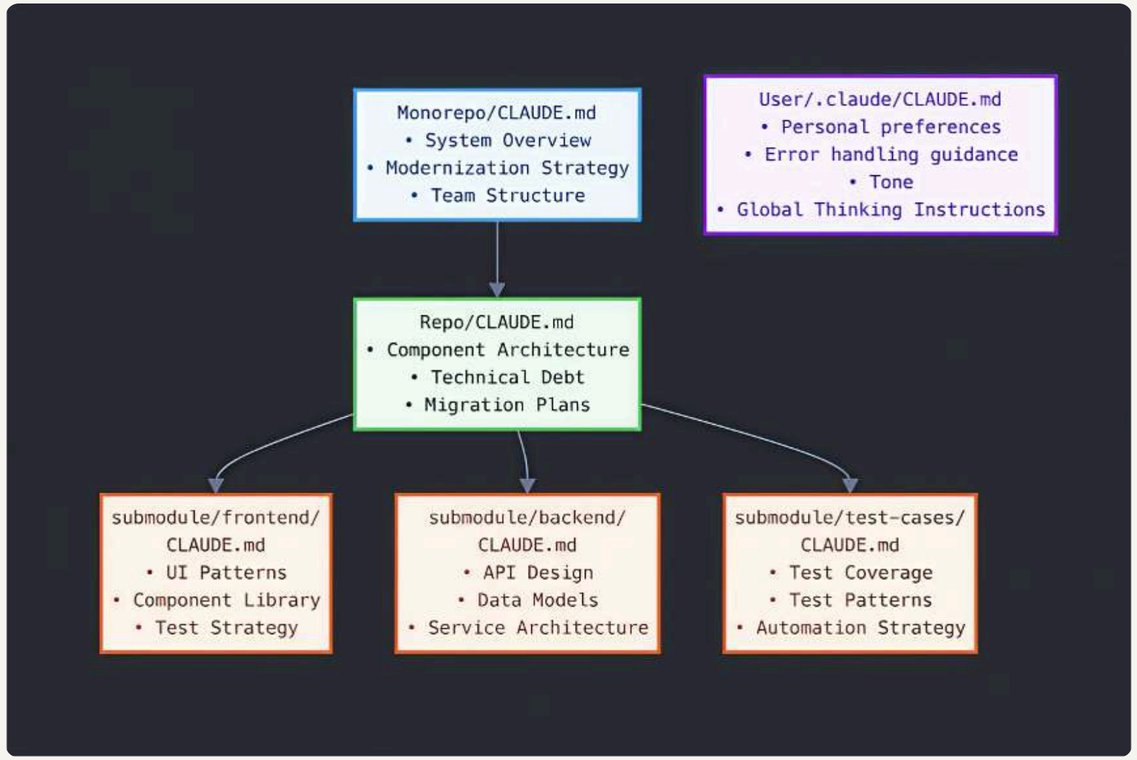
The *CLAUDE.md* file

Recursive Loading

Reads CLAUDE.md from current working directory

Recurses up to load other CLAUDE.md files

Discovers CLAUDE.md nested in subtrees under your current directory



Create a *skill*

Think of a repetitive task you do often, or a prompt that you keep typing. Then, create a skill to automate it.

```
$.claude/skills/recap/SKILL.md
```

Pick a skill

A **Using git**

Create .claude/skills/recap/SKILL.md so the skill runs git status --short and git diff --stat, then summarizes what changed and what looks risky.

B **No git yet**

Create .claude/skills/session-notes/SKILL.md that summarizes what you built, what is unclear, and three prompts to try next — without using git.

Write a *CLAUDE.md*

Take any code project or other folder, and add a CLAUDE.md file. Either write it manually, or use /init, or both. Keep it short and focused.

Add this file

01 **Purpose**
“This project is a one-page site about urban gardening.”

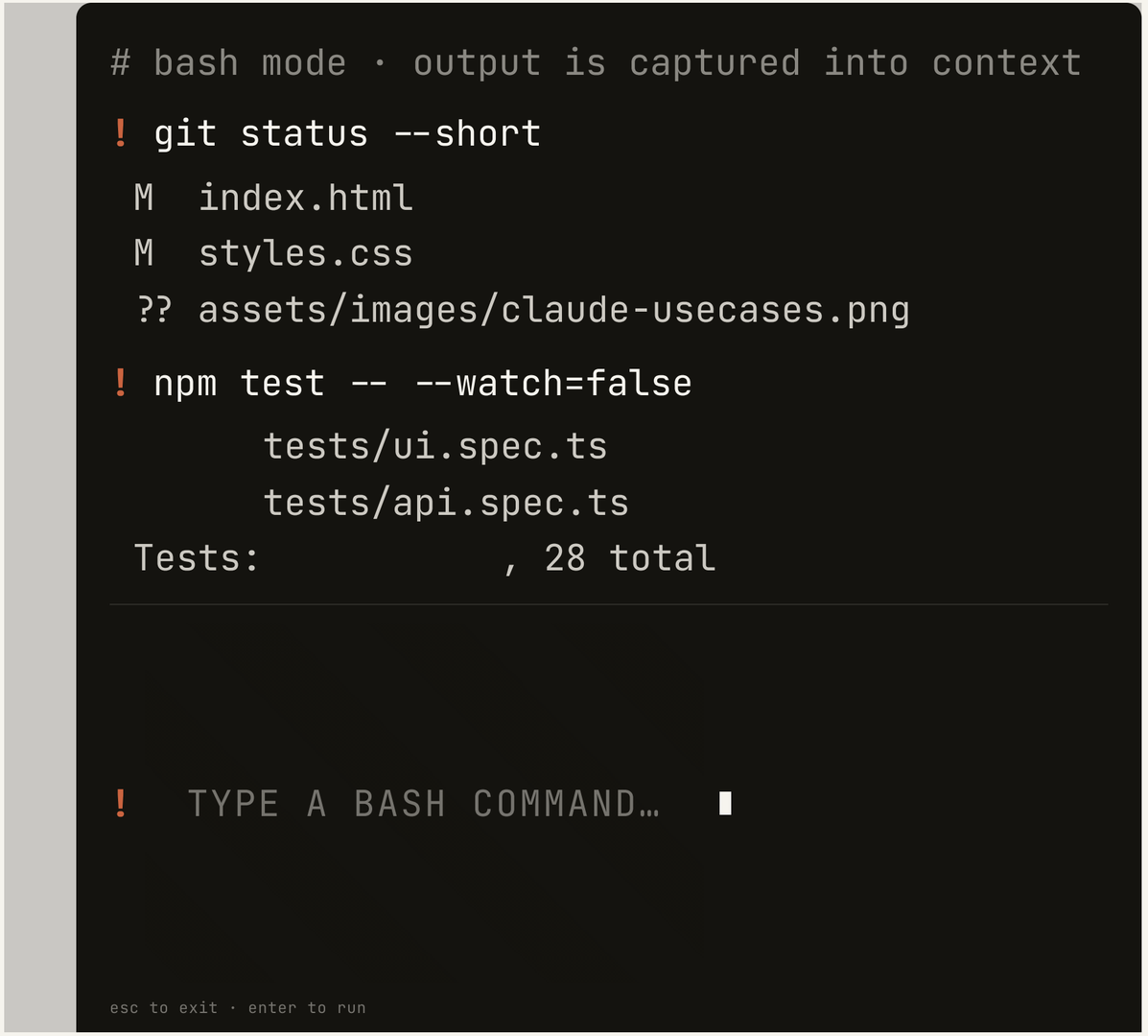
02 **Constraint**
“Only edit index.html unless I explicitly ask for more files.”

05

Advanced Claude *Code*

! for *bash mode*

Use ! to run bash commands yourself (if you don't feel like asking the model). Command and results are shown to the model so you can reference it later on.



Permissions

Control what Claude can do autonomously, or cannot do at all.

Manage using `/permissions`

Global — `~/.claude/settings.json`

Repo level — `.claude/settings.local.json`

Subagents are *specialists*

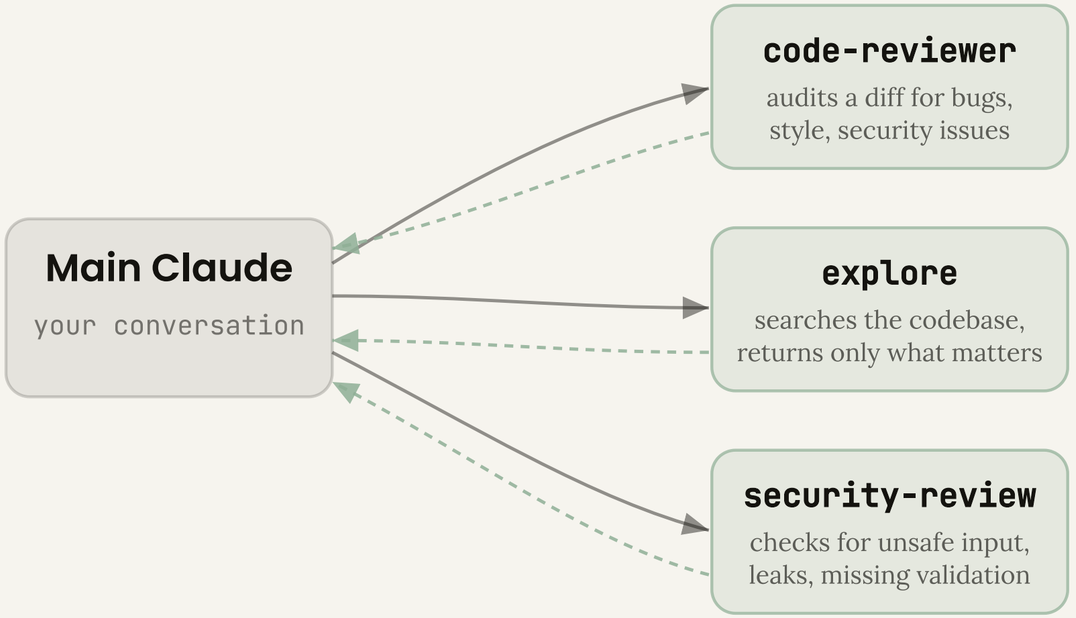
The idea

Claude delegates focused work to a separate agent with its own context. The subagent does the messy exploration, then returns only the useful result.

Use them

Let Claude pick from the description, or ask directly:

```
Use the code-reviewer agent...
```



Create them with */agents*

Where they live

Project agents: `.claude/agents/`

Personal agents: `~/.claude/agents/`

What you configure

Description, prompt, model, tool access, permission mode, hooks, skills, and MCP servers.

```
# .claude/agents/code-reviewer.md
---
description: Expert code reviewer.
  Use after meaningful code changes.
tools: Read, Grep, Glob, Bash
model: sonnet
---

# Code Reviewer
Review for correctness, security, tests, and
maintainability. Return findings first.
```

MCP

The *Model Context Protocol* connects Claude Code to outside services and data.

Manage

`/mcp` for status & auth. `.mcp.json` shares servers with the team.

```
$ claude mcp add --transport http notion http
s://mcp.notion.com/mcp
```

	Enterprise policy (shared)	Global (just me)	Project (shared)	Project (just me)
Memory	`/Library/Application Support/ClaudeCode/CLAUDE.md`	`~/.claude/CLAUDE.md`	`CLAUDE.md`	`CLAUDE.local.md`
Skills	–	`~/.claude/skills/`	`.claude/skills/`	–
Permissions	`/Library/Application Support/ClaudeCode/policies.json`	`~/.claude/settings.json`	`.claude/settings.json`	`.claude/settings.local.json`
MCP servers	–	`claude mcp`	`.mcp.json`	`claude mcp`

Tip #5:

Test the limits, think bigger. If everything you throw at it works, you're not pushing hard enough.

Tip #6:

Use your time while the agent is working. Add a second, third, forth Claude Code instance in new terminal windows/tabs.

Go Build

Always stay *curious*.

FEEDBACK

CONTACT AND RESOURCES

→ **rohlik.net/cbc**
Slides, Resources, Docs.

→ **hi@rohlik.net**
Write me!

→ **code.claude.com/docs**
Official install + guide.