Claude Code
Features Guide

Part 5 of the Claude Code Series

Claude Code Skills: How to Codify Your Agency's Entire Workflow

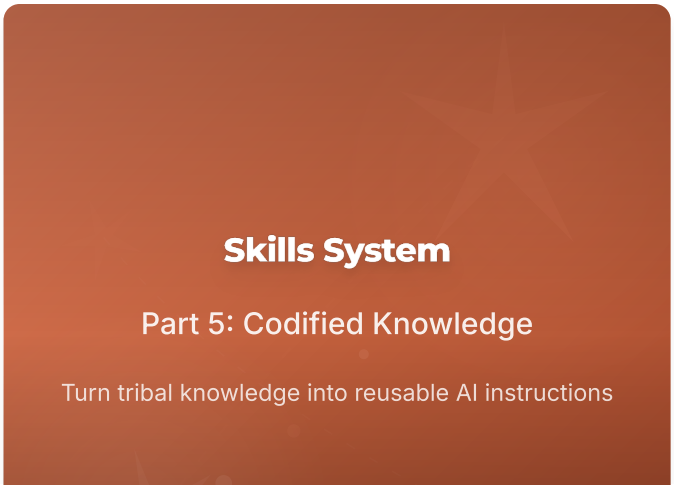
Learn how Claude Code skills let you package repeatable processes into reusable instructions. Turn brand guidelines, audit frameworks, and deployment checklists into skills any AI agent can execute.

9 February 2026 · 12 min read

Every agency has institutional knowledge that lives in people's heads. The way you run an SEO audit. The specific tone a client demands. The deployment checklist that prevents disasters. This knowledge is valuable, but it is fragile -- when a team member leaves or a project changes hands, critical context gets lost.

Claude Code's Skills system solves this problem. Skills let you package instructions, templates, scripts, and reference material into modular units that Claude can load on demand. Think of them as codified expertise: write your process once, and any agent (human or AI) can execute it consistently.

At Jordan James Media, we have been using skills to standardise everything from SEO audit frameworks to content style guides. Here is how the system works and why it matters for agencies and development teams.



What Are Claude Code Skills?

A skill is a folder containing a `SKILL.md` file with instructions that Claude adds to its toolkit. When the skill is relevant to what you are working on, Claude loads it automatically. You can also invoke a skill directly as a slash command -- type `/skill-name` and Claude executes those instructions.

The beauty of skills is efficiency. Claude only loads the skill's short description into its context window until the skill is actually needed. This means you can have dozens of skills available without bloating your context or slowing things down. The full instructions only load when invoked.

Skills follow the open Agent Skills standard from agentskills.io, which means they work across multiple AI tools -- not just Claude Code.

Progressive Skill Loading

Creating Your First Skill

Every skill starts with a `SKILL.md` file. The file has two parts: YAML frontmatter that tells Claude when and how to use the skill, and markdown content with the actual instructions.

Here is a simple example -- a skill that enforces your code review standards:

```
---
name: code-review
description: Review code for quality, security, and style complian
---

When reviewing code, check for:

1. **Security**: SQL injection, XSS, command injection, exposed sec
2. **Performance**: N+1 queries, unnecessary re-renders, missing in
3. **Style**: Consistent naming, proper error handling, no magic nu
4. **Tests**: Coverage for new logic, edge cases, error paths

Format your review as:
- Critical issues (must fix)
- Suggestions (should consider)
- Praise (what was done well)
```

Save this to `.claude/skills/code-review/SKILL.md` in your project, and Claude will use these standards whenever you ask it to review code. You can also type `/code-review` to invoke it explicitly.

Where Skills Live

Where you store a skill determines who can use it:

Location	Path	Who Can Use It
Personal	`~/.claude/skills/<name>/SKILL.md`	You, across all projects
Project	`.claude/skills/<name>/SKILL.md`	Anyone working on this project
Enterprise	Managed settings	Everyone in your organisation
Plugin	`<plugin>/skills/<name>/SKILL.md`	Wherever the plugin is enabled

For agencies, project-level skills are powerful because they travel with the repository. Commit your skills to version control and every team member -- human or AI -- has access to the same standardised processes.

Skills = Codified Expertise

Every agency has processes that only certain team members know. Skills turn that tribal knowledge into structured, reusable instructions any AI agent can execute.

Controlling Who Triggers a Skill

Not every skill should fire automatically. Claude Code gives you two frontmatter fields to control invocation:

`disable-model-invocation: true`

This prevents Claude from loading the skill on its own. Only you can trigger it with `/skill-name`. Use this for workflows with side effects -- deployments, sending messages, publishing content. You do not want Claude deciding to deploy because your code "looks ready."

```
---
name: deploy-staging
description: Deploy the application to the staging environment
disable-model-invocation: true
---

Deploy to staging:
1. Run the full test suite
2. Build production assets
3. Push to staging environment
4. Run smoke tests against staging URL
5. Post results to the team Slack channel
```

`user-invocable: false`

This hides the skill from the slash command menu entirely. Only Claude can load it, and only when it determines the skill is relevant. Use this for background knowledge that is not an action -- legacy system context, domain-specific terminology, architectural decisions.

```
---
name: legacy-billing-context
description: Context about the legacy billing system's quirks and c
user-invocable: false
---

The legacy billing system has these constraints:
- Invoice numbers must be sequential with no gaps
- Refunds cannot exceed the original transaction amount
- Currency conversion uses rates cached at midnight AEST
- The payment gateway times out after 30 seconds
```

Here is how the two fields interact:

Setting	You Can Invoke	Claude Can Invoke	Best For
Default	Yes	Yes	General-purpose skills
`disable-model-invocation: true`	Yes	No	Deployments, publishing, messaging
`user-invocable: false`	No	Yes	Background knowledge, domain context

4

Skills Built So Far

Client onboarding, blog publishing, site launch checklist, and brand guidelines — each encapsulating hours of process knowledge.

Adding Supporting Files

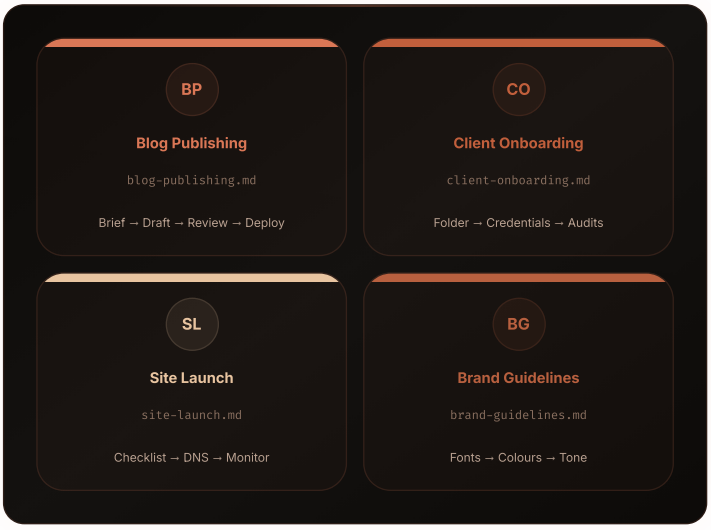
Skills are not limited to a single markdown file. A skill directory can contain templates, example outputs, reference documentation, and scripts. Keep `SKILL.md` focused and under 500 lines, then reference supporting files for the detail.

```
seo-audit/
 SKILL.md              # Main instructions
 checklist.json        # Audit checklist with scoring criter
 examples/
   sample-report.md    # Example of a completed audit
 scripts/
   check-meta-tags.js # Script to validate meta tags
```

Reference these files from your `SKILL.md` so Claude knows they exist and when to use them:

```
```

The Skills Library



Resources

- For the full scoring criteria, see checklist.json
- For an example of the expected output format, see examples/sample-report.md

```
Claude will read these files when it needs them, keeping you
```

```
---
```

Dynamic Context and Arguments

Skills support two powerful features that make them flexible enough for real work.

Arguments

Pass arguments when invoking a skill using the `$ARGUMENTS` placeholder:

```
---
name: fix-issue
description: Fix a GitHub issue by 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 tests
5. Create a commit
```

Type `/fix-issue 456` and Claude receives "Fix GitHub issue 456 following our coding standards." You can also access individual arguments by position with `$0`, `$1`, `$2`.

Dynamic Context Injection

The `!`command`` syntax runs shell commands before the skill content reaches Claude. The output replaces the placeholder, so Claude receives real data:

```
---
name: pr-summary
description: Summarise changes in a pull request
---
```

Pull request context

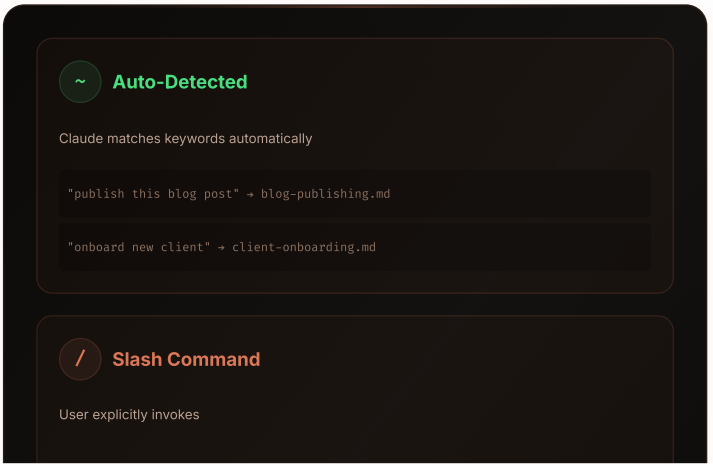
- PR diff: ! `` `gh pr diff` ``
- Changed files: ! `` `gh pr diff --name-only` ``

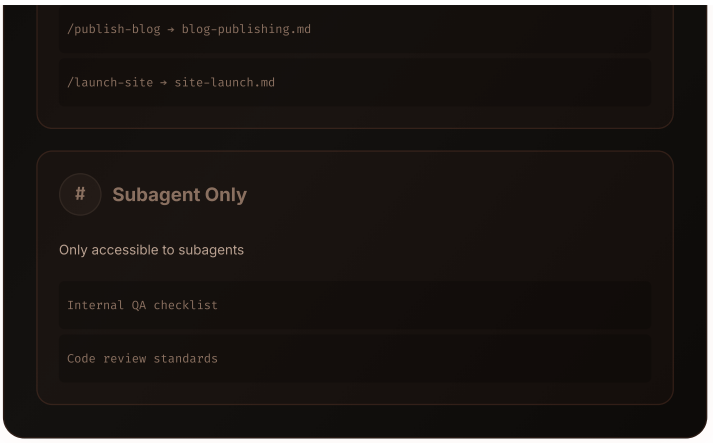
Summarise this pull request, highlighting breaking changes and areas that need review.

```
This is preprocessing -- the commands execute first, and Cla

---
```

Three Invocation Modes





Running Skills in Subagents

Add `context: fork` to your frontmatter to run a skill in an isolated subagent. The skill content becomes the prompt that drives the subagent, separate from your main conversation. This is ideal for research tasks that might otherwise pollute your context window:

```
---
name: deep-research
description: Research a topic thoroughly across the codebase
context: fork
agent: Explore
---

Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyse the code
3. Summarise findings with specific file references
```

The agent field determines the execution environment. Options include built-in agents like `Explore` (read-only, optimised for searching) and `Plan` (for planning tasks), or any custom subagent you have defined.

PRO TIP

Progressive Disclosure

Claude scans skill metadata first — only if a skill matches the current task does it load the full instructions. This means 20 skills don't bloat your context; only the relevant one loads.

The Agency Angle: Why Skills Change Everything

This is where skills become transformative for agencies. Every repeatable process your agency runs can become a skill.

Brand Guidelines as a Skill

Package a client's brand voice, terminology preferences, and content rules into a skill. Every piece of content Claude produces for that client follows the same standards -- without anyone having to remember or re-explain them.

SEO Audit Framework as a Skill

At our agency, we run standardised SEO audits using a checklist-driven framework. The checklist, scoring criteria, and reporting template all live in a skill directory. Any agent can pick up an audit task and execute it to the same standard. The process is documented, repeatable, and improvable.

Deployment Checklists as a Skill

Pre-launch checks, staging verification, DNS configuration, SSL validation -- package the entire deployment process as a skill with `` `disable-model-invocation: true` `` so it only runs when explicitly triggered.

Content Style Guides as a Skill

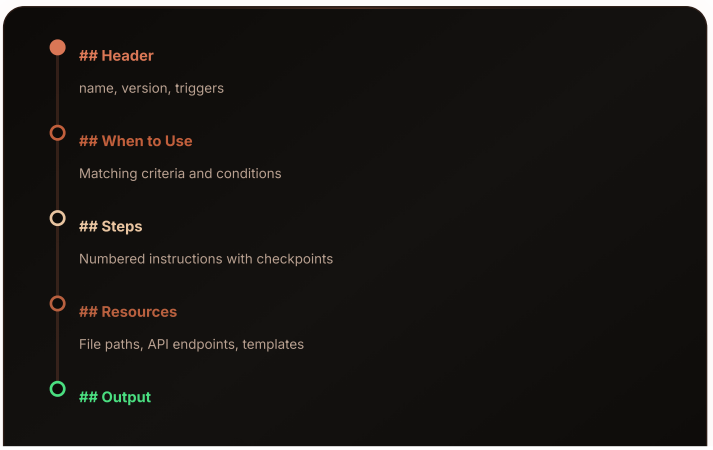
Australian English spelling (optimise, analyse, colour). First person plural. Expert but approachable tone. Technical accuracy without jargon. These rules become a skill that Claude loads whenever it is writing content for that client.

The Compound Effect

Skills compose together. An agent working on a client project might load the brand guidelines skill for tone, the SEO audit skill for technical checks, and the deployment skill when it is time to go live. Each skill is focused and modular, but together they encode your agency's entire operational playbook.

This is what separates agencies that scale from those that do not. When your processes live in people's heads, quality depends on who is working that day. When your processes live in skills, quality is consistent regardless of whether a human or AI agent picks up the task.

Anatomy of a SKILL.md File



Expected deliverable format

Best Practices

Based on our experience building and deploying skills across client projects:

1. **Keep skills focused.** One skill, one job. A "do everything" skill is just a long prompt. Break complex workflows into composable skills.

2. **Write specific descriptions.** The description field is how Claude decides when to load your skill. Vague descriptions lead to skills triggering at the wrong time -- or not at all.

3. **Use `disable-model-invocation` for anything with side effects.** Deployments, publishing, API calls that cost money. If the action cannot be undone, make it manual-only.

4. **Commit project skills to version control.** This makes them available to every team member and every agent session. Skills that live only on one machine are skills waiting to be lost.

5. **Start small, iterate.** Your first skill does not need to be perfect. Write the instructions, test them, refine based on the output. Skills improve with use.

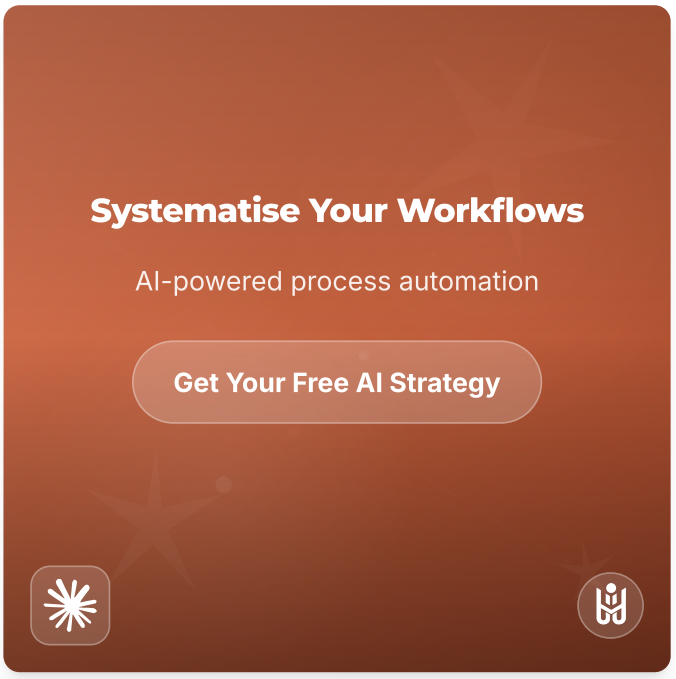
> *Skills solve the bus factor problem. If your best process person left tomorrow, their knowledge wouldn't go with them — it's codified and executable.*
>
> — Jordan James Media

Key Takeaways

1. **Skills are modular instruction sets** that Claude loads on demand, keeping your context lean
2. **SKILL.md is the entry point** -- YAML frontmatter for configuration, markdown for instructions

3. **Invocation control** lets you decide whether skills trigger automatically or only on command
4. **Supporting files** keep complex skills organised without bloating the main instructions
5. **Dynamic context** and **arguments** make skills flexible enough for real-world workflows
6. **Agency processes become skills** -- brand guidelines, audit frameworks, deployment checklists
7. **Skills compose together**, encoding your entire operational playbook
8. **Skills follow an open standard**, working across multiple AI tools

---



Ready to Systematise Your Agency's Workflows?

Jordan James Media helps Australian businesses build AI-powered workflows that scale. From skills architecture to full agent deployment, we turn institutional knowledge into repeatable, automated processes.

Talk to Us About AI Workflow Automation

Related Reading:

- Claude Code's Biggest Update: 7 Features That Changed How We Build
- Claude Code Custom Subagents: Build Your Own Specialist AI Team
- Claude Code MCP Integration: Connect Your AI to Every Tool You Use

Key Takeaway

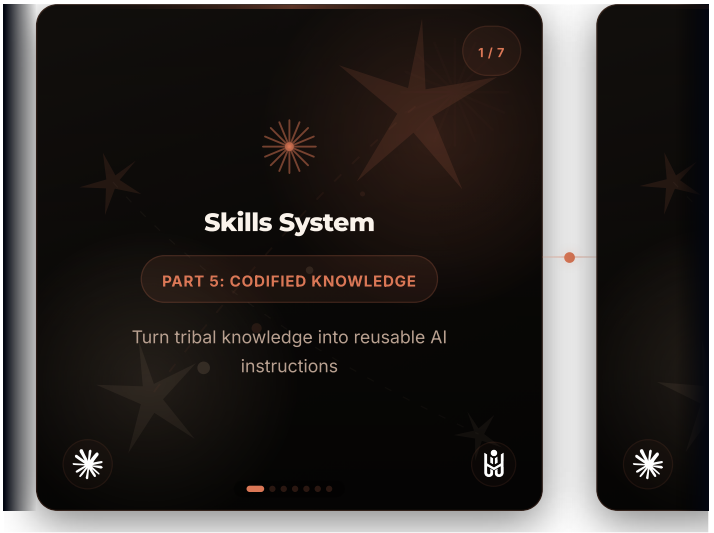
1. Skills turn tribal knowledge into reusable, AI-executable instructions
2. Progressive disclosure keeps context efficient — metadata first, full load only when relevant
3. Support arguments and dynamic context for flexible execution
4. Can be scoped to personal, project, or organisation level
5. Best starting point: codify your most repeated process as a skill

Social Media Carousel

7 cards — Scroll to browse, click any card to download

PDF for LinkedIn

ZIP Images



Share This Article

Spread the knowledge

Twitter LinkedIn Copy

Build With Claude Code

We use Claude Code to ship faster, smarter, and at scale. Let us build your next project.

Get Your Free AI Strategy →

App Development

Claude Code
8 parts

- Part 0
- Part 1
- Part 2
- Part 3
- Part 4
- Part 5
- Part 6
- Part 7

View All

Quick Links

- → Series Home
- → App Development
- → Get In Touch

Claude Code Stats

Model	Opus 4.6
Features	7
Series Parts	8

Agency Tested
Real projects, real results

Ready to outperform your competition?

Get your free AI strategy consultation today.

Start Your Free Consultation →

Australia's AI-first digital marketing agency. We build intelligent systems that drive exponential growth.

info@jordanjamesmedia.com

0475 897 737

Sydney, Australia

SERVICES

- SEO Services
- Google Ads
- Website Design
- Local SEO
- Content Marketing
- App Development

COMPANY

- About Us
- Blog
- Case Studies
- Pricing
- Tools
- Contact

LOCATIONS

- Sydney
- Melbourne
- Brisbane
- Perth
- Adelaide
- All Locations

All Locations →

Privacy Policy

Terms of Service

Portal Login

Built with Claude 4.6 Opus