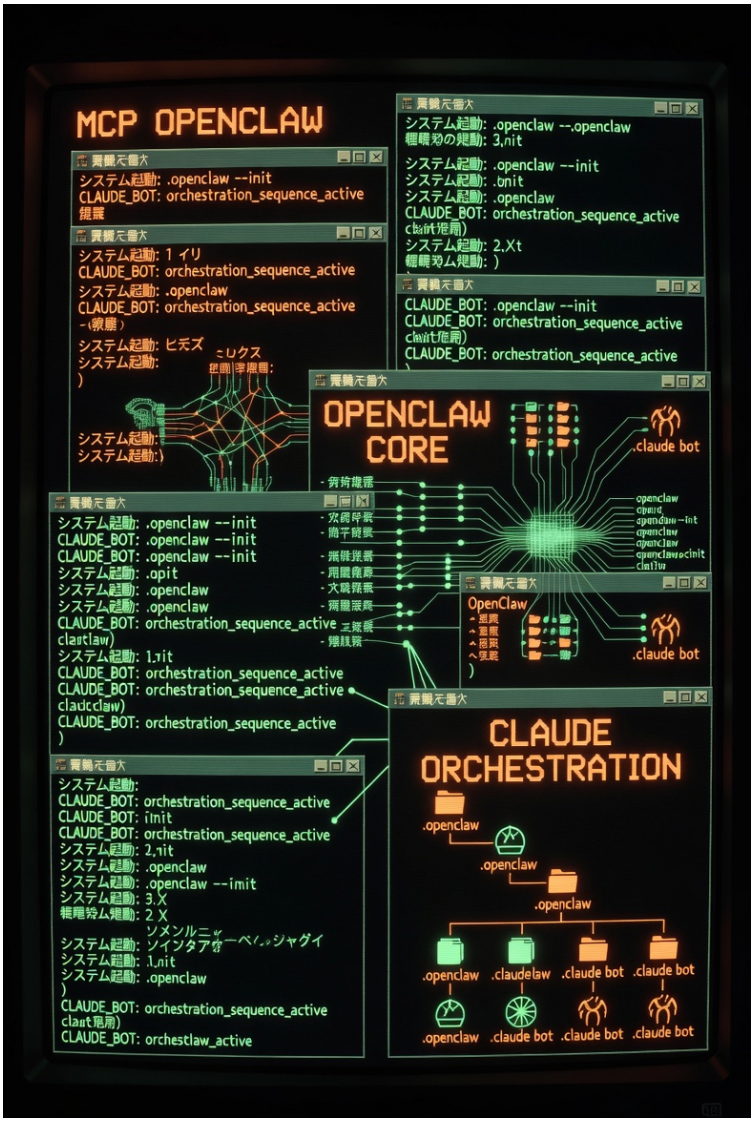


Agent Skills Architecture Guide

.claude & .openclaw & .kilocode & .codex folder structure,
skill registration, agent orchestration, and self-hosted models

Compiled by s.coy

@shaneswrld_ | github.com/shane9coy

February 2026

Table of Contents

01 How Skills Work Under the Hood
02 Canonical Folder Structures (Claude / KiloCode / Codex)
03 SKILL.md Anatomy & Frontmatter Reference
04 Skill Subdirectory Conventions
05 Precedence & Loading Order
06 CLAUDE.md vs SKILL.md vs Agents vs Commands
07 Claude Code vs KiloCode vs OpenClaw vs Codex
08 Agent Orchestration: Folders Map to Runtime
09 Self-Hosted Model Configuration
10 Troubleshooting: Skills Not Registering
11 Installing External Skills
12 Quick Skill Template
A The new-skill Installer Skill
B CLAUDE.md Starter Template

01 — How Skills Work Under the Hood

Skills are **on-demand prompt expansion** — not agents, not plugins, not tools. The framework scans skill folders, reads only the YAML frontmatter (name + description), and builds a compact <available_skills> XML list in the system prompt. When a user request matches a description, the full SKILL.md body is injected into context. The model follows the instructions and optionally runs bundled scripts. The core principle is **lazy loading** — skills cost almost nothing until triggered.

The Lifecycle

Phase	What Happens	Context Cost
1. Discovery	Framework scans */SKILL.md in skill paths	0 tokens
2. Indexing	Builds <available_skills> from name + description	~24 tok/skill
3. Matching	User request matched against descriptions	0 extra
4. Injection	Full SKILL.md body injected into conversation	Varies (body size)
5. Execution	Model follows instructions, runs scripts	Script output tokens

System prompt overhead is deterministic: ~195 chars base + ~97 chars per skill (plus your name/description lengths). At ~4 chars/token, that's roughly 24 tokens per skill sitting in the system prompt at all times.

Why Skills Over Agents

Skills	Agents
Lightweight — just prompt text	Heavy — full sub-process with own context
Composable — multiple fire in one turn	Isolated — separate conversation thread
Cheap — only loaded when matched	Expensive — own system prompt + tools
Stateless — no memory between calls	Stateful — can maintain own context
Best for: workflows, conventions, tasks	Best for: long-running, risky, parallel work

02 — Canonical Folder Structures

All three platforms use the same AgentSkills spec (same SKILL.md, same frontmatter, same subdirectories) but use different folder paths. Here are the canonical layouts for each, followed by a cross-platform comparison.

Claude Code

```
your-project/
|-- .claude/
|   |-- CLAUDE.md                # Always loaded (project memory)
|   |-- settings.json            # Permissions, env vars
|   |-- commands/                # Custom slash commands
|   |   +-- review.md            # Invoked via /review
|   +-- skills/
|       |-- mcp-builder/         # Complex skill with resources
|       |   |-- SKILL.md
|       |   |-- scripts/
|       |   |   +-- scaffold.py
|       |   |-- references/
|       |   |   +-- mcp_best_practices.md
|       |   +-- assets/
|       |       +-- template.json
|       |-- new-skill/           # Skill installer
|       |   +-- SKILL.md
|       +-- deploy-prod/
|           +-- SKILL.md

~/.claude/                        # Global (all projects)
|-- CLAUDE.md
+-- skills/
    +-- formatting-standards/
        +-- SKILL.md
```

KiloCode

```
your-project/
|-- .kilocode/
|   |-- AGENTS.md              # KiloCode project memory (native standard)
|   |-- skills/                  # Generic skills (all modes)
|   |   +-- new-skill/
|   |       +-- SKILL.md
|   |-- skills-code/             # Code mode only
|   |   +-- linting-rules/
|   |       +-- SKILL.md
|   +-- skills-architect/        # Architect mode only
|       +-- microservices/
|           +-- SKILL.md
~/.kilocode/                     # Global
|-- AGENTS.md                    # Global instructions
|-- skills/                      # All modes, all projects
|   +-- my-global-skill/
|       +-- SKILL.md
+-- skills-code/                 # Code mode, all projects
    +-- typescript-patterns/
        +-- SKILL.md

# Mode directory pattern: skills-{mode-slug}
# Modes: code, architect, ask, debug, orchestrate
# Also reads .claude/skills/ as fallback
```

OpenAI Codex

```
your-project/
|-- .codex/                    # Codex native (current)
|   |-- AGENTS.md              # Project memory
|   +-- skills/                # Project skills
|       |-- new-skill/
|       |   +-- SKILL.md
|       +-- deploy-prod/
|           +-- SKILL.md
|-- .agents/                   # Legacy fallback
|   +-- skills/
|       +-- new-skill/
|           +-- SKILL.md
|-- AGENTS.md                  # Root-level project memory

~/.codex/                      # Global
|-- AGENTS.md                  # Global instructions
|-- AGENTS.override.md         # Override (takes priority)
|-- config.toml                # Skill enable/disable config
+-- skills/
    |-- .system/               # Built-in (plan, skill-creator)
    |   +-- plan/
    |       +-- SKILL.md
    +-- my-global-skill/       # Your global skills
        +-- SKILL.md

# Codex walks UPWARD from CWD to repo root
# checking every dir for .codex/skills/ and .agents/skills/
# AGENTS.override.md beats AGENTS.md at same level
```

Cross-Platform Path Reference

	Claude Code	KiloCode	OpenAI Codex
Project skills	.claude/skills/	.kilocode/skills/	.codex/skills/
Global skills	~/.claude/skills/	~/.kilocode/skills/	~/.codex/skills/
System skills	Marketplace/bundled	Bundled	~/.codex/skills/.system/
Project memory	CLAUDE.md	AGENTS.md	AGENTS.md
Memory override	N/A	N/A	AGENTS.override.md
Config file	.claude/settings.json	VS Code settings	~/.codex/config.toml
Mode-scoped	No	Yes: skills-{mode}/	No
Scan direction	Project root down	Project root down	CWD upward to root
Skill installer	Marketplace UI	npx skills install	Built-in $skill-installer
Disable w/o delete	Toggle in settings	Toggle in VS Code	config.toml [[skills]]
Reads .claude/?	Yes (native)	Yes (fallback)	No
Reads .agents/?	No	No	Yes (fallback)
Reads .codex/?	No	No	Yes (native)

Key Differences

Codex scans upward — from your current working directory up to the repo root, checking every directory for .codex/skills/ and .agents/skills/ (legacy fallback). Claude Code and KiloCode scan downward from the project root. This means Codex can pick up skills from parent directories.

Codex uses AGENTS.md instead of CLAUDE.md for project memory, with an override system: AGENTS.override.md at the same level always wins. Global overrides live in ~/.codex/AGENTS.override.md. This lets you temporarily change global behavior without touching the base file.

KiloCode has mode-scoped skills — drop skills into skills-code/, skills-architect/, skills-debug/, etc. to restrict them to specific agent modes. Generic skills/ applies everywhere. Neither Claude Code nor Codex has this concept.

KiloCode's native memory file is AGENTS.md — the open standard also used by Cursor and Windsurf. KiloCode reads CLAUDE.md only as a fallback for cross-tool compatibility.

Codex config.toml lets you disable a skill without deleting it: `[[skills.config]]` entries with `enabled = false`. Claude Code toggles skills in the settings UI. KiloCode toggles in VS Code.

For maximum compatibility, ship skills in .claude/skills/, .codex/skills/, and .agents/skills/. KiloCode reads .claude/ as fallback, so three folders covers all four platforms.

Critical (all platforms): Each skill MUST be in its own named subfolder. skills/SKILL.md will NOT register — it must be skills/my-name/SKILL.md .

03 — SKILL.md Anatomy & Frontmatter

```
---
name: my-skill-name
description: "Concise trigger. Use when [X]. Handles [Y, Z]."
---

# Skill Title

## Instructions
Step-by-step instructions the model follows...

## Examples
Input/output pairs showing expected behavior...

## Error Handling
What to do when things fail...
```

Frontmatter Reference

Field	Req	Purpose
name	YES	Unique ID. Max 64 chars. Becomes /skill-name command.
description	YES	THE trigger. Max 200 chars. Must include WHAT + WHEN.
license	No	License info string
context	No	fork = subagent, inline = current context (default)
disable-model-invocation	No	true = only user can invoke (for destructive ops)
user-invocable	No	false = only model can invoke (background knowledge)
allowed-tools	No	Restrict tools: Read, Grep, Glob, Bash, Write
model	No	Override model for this skill only
metadata	No	Single-line JSON. Extra key-value data.

Description Do's and Don'ts

	Example
BAD	Helps with development tasks
BAD	This handles [brackets] and #tags
GOOD	"Create MCP servers. Use when building API integrations."
GOOD	"Deploy to production. Use when pushing releases or shipping."

Most common failure: Unquoted descriptions with special YAML characters cause the skill to be silently skipped. Always wrap descriptions in double quotes.

04 — Skill Subdirectory Conventions

Directory	Contents	How the Model Uses It
scripts/	Python, Bash, JS executables	Runs via Bash tool. Ref: {baseDir}/scripts/x.py
references/	API docs, schemas, guides	Read on-demand via progressive disclosure
assets/	Templates, configs, images	Copied or modified during output generation

Progressive Disclosure

Show the model just enough to decide, then reveal more as needed. This keeps context lean and costs low.

Layer	What's Exposed	When
1. Frontmatter	name + description (system prompt)	Always — every request
2. SKILL.md body	Full instructions + examples	When skill is triggered
3. references/*	Deep docs, schemas, API refs	When SKILL.md says to read them
4. scripts/*	Executable code + output	When instructions call for execution

Keep SKILL.md under 500 lines. In the body, write: `"For details, read {baseDir}/references/api-docs.md"`

05 — Precedence & Loading Order

Claude Code

Priority	Location	Scope
1 (highest)	<project>/.claude/skills/	This project only
2	~/.claude/skills/	All projects for this user
3 (lowest)	Marketplace / bundled plugins	Global defaults

OpenClaw

Priority	Location	Scope
1 (highest)	<workspace>/skills/	This agent only
2	~/.openclaw/skills/	All agents on machine
3	Bundled skills (npm package)	Ship-time defaults
4 (lowest)	skills.load.extraDirs entries	Shared skill packs

KiloCode

Priority	Location	Scope
1 (highest)	<project>/.kilocode/skills-{mode}/	This project, this mode only
2	<project>/.kilocode/skills/	This project, all modes
3	~/.kilocode/skills-{mode}/	Global, mode-specific
4	~/.kilocode/skills/	Global, all modes
5 (lowest)	.claude/skills/ (fallback)	Cross-tool compat

KiloCode's unique feature: **mode-specific skill directories**. Use skills-code/ for coding mode, skills-architect/ for architecture mode, skills-debug/ for debug mode, etc. Generic skills/ applies to all modes. The directory pattern is skills-{mode-slug} where mode-slug matches the agent mode identifier.

06 — CLAUDE.md vs SKILL.md vs Agents vs Commands

File	Loaded	Purpose	Best For
CLAUDE.md	Every session	Persistent memory	Stack, conventions, commands
skills/*/SKILL.md	On-demand	Specialized capability	Workflows, integrations
agents/*.md	When spawned	Isolated sub-assistant	Long-running, risky tasks
commands/*.md	User types /x	Slash command	Quick actions, shortcuts
settings.json	Session start	Permissions config	Tool access, env vars

Decision Flowchart

```
Does the model need this info EVERY time?
  YES --> CLAUDE.md
  NO
   |
   Does it need it SOMETIMES (triggered by task type)?
     YES --> skills/my-skill/SKILL.md
     NO
      |
      Is it a quick action the USER invokes?
        YES --> commands/action.md
        NO
         |
         Is it dangerous / long-running / needs isolation?
           YES --> agents/worker.md
           NO  --> Probably just put it in CLAUDE.md
```

07 — Claude Code vs KiloCode vs OpenClaw vs Codex

All four use the AgentSkills spec (SKILL.md with YAML frontmatter), but differ in folder paths, runtime, and orchestration.

CLAUDE CODE

```
project/
|-- .claude/
| |-- CLAUDE.md
| |-- settings.json
| +-- skills/
| +-- review/
| +-- SKILL.md
~/.claude/
+-- skills/ (global)
```

OPENCLAW

```
~/.openclaw/
|-- openclaw.json
|-- workspace/
| |-- AGENTS.md
| +-- skills/
| +-- review/
| +-- SKILL.md
|-- skills/ (shared)
+-- memory/
```

KILOCODE

```
project/
|-- .kilocode/
| |-- skills/
| | +-- review/
| | +-- SKILL.md
| +-- skills-code/
| +-- lint/
| +-- SKILL.md
~/.kilocode/
+-- skills/ (global)
```

CODEX

```
project/
|-- .codex/
|   |-- AGENTS.md
|   +-- skills/
|       +-- review/
|           +-- SKILL.md
|-- .agents/     # Legacy
|   +-- skills/
|-- AGENTS.md
~/.codex/
|-- AGENTS.md
|-- config.toml
+-- skills/ (global)
```

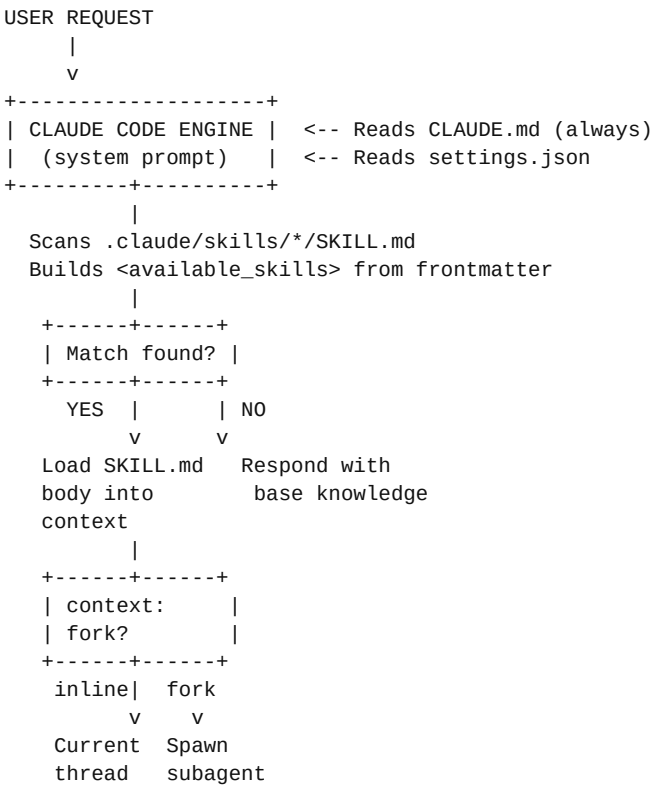
Feature Comparison

Feature	Claude Code	OpenClaw	KiloCode	Codex
Config	settings.json	openclaw.json	VS Code	config.toml
Memory	CLAUDE.md	AGENTS.md	AGENTS.md	AGENTS.md
Skills	.claude/skills/	workspace/skills/	.kilocode/skills/	.codex/skills/
Global	~/.claude/skills/	~/.openclaw/skills/	~/.kilocode/skills/	~/.codex/skills/
Modes	No	No	skills-{mode}/	No
Self-hosted	BASE_URL env	Native config	Any via key	Any via key
Cron	No	jobs.json	No	No
Messaging	CLI + IDE	Telegram/etc.	VS Code + CLI	CLI + IDE
Skill fmt	AgentSkills	AgentSkills	AgentSkills	AgentSkills

08 — Agent Orchestration: Folders to Runtime

How folder structure translates to runtime behavior. Understanding this mapping is key to structuring your project correctly.

Claude Code Runtime Flow

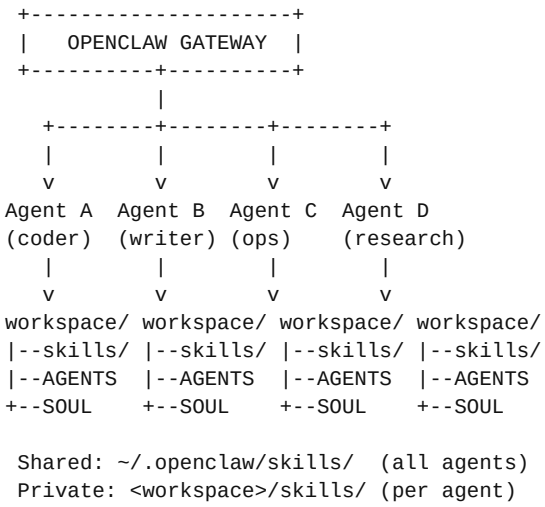


OpenClaw Runtime Flow

Folder-to-Runtime Mapping

Folder / File	Loaded At	Runtime Effect
CLAUDE.md / AGENTS.md	Session start	Injected into system prompt. Always present.
settings.json / openclaw.json	Session start	Sets permissions, env vars, model routing.
skills/x/SKILL.md (frontmatter)	Session start	Added to <available_skills> list.
skills/x/SKILL.md (body)	On trigger	Injected into conversation context.
skills/x/references/*.md	On demand	Read by model when SKILL.md instructs.
skills/x/scripts/*.py	On demand	Executed via Bash. Output returned to model.
commands/x.md	User types /x	Content becomes user message.
agents/x.md	When spawned	Creates isolated subagent.
cron/jobs.json (OpenClaw)	Gateway start	Schedules periodic invocations.
memory/*.md (OpenClaw)	Session start	Recent context + long-term memory.

Multi-Agent Orchestration (OpenClaw)



09 — Self-Hosted Model Configuration

Skills are framework-level — the same SKILL.md works identically whether backed by Claude API, Ollama, LM Studio, or vLLM. Only the provider block changes.

Ollama (easiest)

```
# Install + pull a model
curl -fsSL https://ollama.ai/install.sh | sh
ollama pull qwen3:8b

# openclaw.json provider config:
"ollama": {
  "baseUrl": "http://127.0.0.1:11434/v1",
  "apiKey": "ollama-local",
  "api": "openai-completions",
  "models": [{
    "id": "qwen3:8b",
    "contextWindow": 131072,
    "maxTokens": 8192
  }]
}
```

LM Studio (GUI, good for experimenting)

```
"lmstudio": {
  "baseUrl": "http://127.0.0.1:1234/v1",
  "apiKey": "lm-studio",
  "api": "openai-completions",
  "models": [{
    "id": "glm-4.7-flash",
    "contextWindow": 128000,
    "maxTokens": 8192
  }]
}
```

vLLM (production, 5-10x faster)

```
# Install and serve
pip install vllm
vllm serve meta-llama/Llama-3.1-70B-Instruct \
  --served-model-name llama-3.1-70b \
  --api-key YOUR_KEY \
  --port 8000 \
  --enable-auto-tool-choice \
  --max-model-len 131072

# openclaw.json provider config:
"vllm": {
  "baseUrl": "http://127.0.0.1:8000/v1",
  "apiKey": "YOUR_KEY",
  "api": "openai-completions",
  "models": [{
    "id": "llama-3.1-70b",
    "contextWindow": 131072,
    "maxTokens": 8192
  }]
}
```

Claude Code + Local Models

Claude Code can also route to local Ollama models by setting the ANTHROPIC_BASE_URL environment variable:

```
export ANTHROPIC_BASE_URL=http://localhost:11434
# Then run Claude Code normally — it hits Ollama instead of API
```

Model Recommendations by VRAM

VRAM	Model	Quant	Notes
< 2 GB	Qwen3 1.7B Thinking	Q4_K_M	Ultra-light, runs on anything
2-4 GB	Qwen3 4B Thinking	Q4_K_M	Outperforms most 8B at tool calling
4-8 GB	Qwen3 8B / Qwen3 4B	Q4_K_M	Sweet spot for older GPUs
8-16 GB	Qwen3 8B	Q6_K / FP16	Best all-around for most laptops
24 GB	GLM 4.7 Flash / GPT-OSS 20B	Q4_K_M	Strong coding + tool use
48 GB	GLM 4.7 Flash (30B)	Q4_K_M	Excellent coding + reasoning
48-80 GB	Qwen3 Coder (80B)	Q4_K_XL	Near-API quality
90+ GB	GPT-OSS 120B	Q4_K_M	Maximum local capability
192 GB (cloud)	MiniMax M2.1 (139B)	FP8	Free via AMD Dev Cloud

Set contextWindow to 32K+ minimum. OpenClaw sends all context per request. Low context = crashes and truncation.

Security Warnings

Self-hosted agents execute commands on your machine. Take precautions:

Risk	Mitigation
Command injection	Run in Docker sandbox with volume mounts
Network exposure	Never expose ports publicly without auth
Workspace escape	Use OpenClaw's workspace isolation (default)
Malicious skills	Audit third-party skills before installing
Data exfiltration	Use safeguard compaction mode

In January 2026, researchers found 42,000+ exposed OpenClaw instances — 93% were vulnerable. Always firewall your ports and use auth tokens.

10 — Troubleshooting: Skills Not Registering

Symptom	Cause	Fix
Skill not in list	Flat file: skills/SKILL.md	Must be skills/<n>/SKILL.md
Silently skipped	Unquoted description	Wrap in double quotes
Never triggers	Vague description	Add specific trigger keywords
Frontmatter ignored	Missing --- markers	Must start and end with ---
1 of N loads	YAML special chars in others	Quote ALL descriptions
Triggers but fails	Missing referenced files	Check scripts/ and references/ exist
Too much context	SKILL.md > 500 lines	Move depth to references/
OpenClaw parse err	Multi-line YAML	Single-line frontmatter only
Model ignores skill	Trigger info in body	Move 'when to use' to description

Diagnostic Commands

```
# Find all skills
find .claude/skills -name "SKILL.md"

# Dump frontmatter
for f in .claude/skills/*/SKILL.md; do
  echo "=== $f ==="
  sed -n '/^---$/,/^---$/p' "$f"
done

# Find unquoted descriptions
grep -n "^description:" .claude/skills/*/SKILL.md | grep -v '"'

# Check sizes
wc -l .claude/skills/*/SKILL.md | sort -rn | head

# OpenClaw: restart after changes
systemctl --user restart openclaw-gateway
```

11 — Installing External Skills

From GitHub

```
git clone --depth 1 <repo-url> /tmp/skill-clone
SKILL=$(find /tmp/skill-clone -name "SKILL.md" | head -1)
NAME=$(basename "$(dirname "$SKILL")")
mkdir -p .claude/skills/$NAME
cp -r "$(dirname "$SKILL")"/* .claude/skills/$NAME/
rm -rf /tmp/skill-clone
```

From Zip

```
unzip skill.zip -d /tmp/install
SKILL=$(find /tmp/install -name "SKILL.md" | head -1)
NAME=$(grep "^name:" "$SKILL" | sed 's/name: *//' | tr -d '"')
mkdir -p .claude/skills/$NAME
cp -r "$(dirname "$SKILL")"/* .claude/skills/$NAME/
```

Post-Install Validation

```
ls -la .claude/skills/<n>/
head -5 .claude/skills/<n>/SKILL.md
# Then: ask your agent something matching the description
```

12 — Quick Skill Template

```
---
name: your-skill-name
description: "What it does. Use when [trigger]. Handles [X, Y]."
---

# Your Skill Name

## Overview
One paragraph explaining purpose.

## Instructions
1. First step
2. Second step
3. Third step

## Examples
### Example 1: [Scenario]
**Input:** "user says this"
**Action:** Do X, then Y
**Output:** Expected result

## Error Handling
- If X fails, do Y

## Notes
- Keep under 500 lines
- Move depth to references/
```

Appendix A — The new-skill Installer Skill

Install the **new-skill** SKILL.md in your .claude/skills/new-skill/ folder to give your CLI agent the ability to install, scaffold, validate, and manage other skills.

```
mkdir -p .claude/skills/new-skill
cp new-skill-SKILL.md .claude/skills/new-skill/SKILL.md
```

Built-in Workflows

#	Workflow	Trigger Example
1	Create from scratch	"create a new skill for X"
2	Install from URL/repo/zip	"install this skill: <url>"
3	Install MCP Builder	"set up the mcp-builder skill"
4	Validate a skill	"check if my skill is correct"
5	Audit .claude folder	"scan and fix my .claude structure"

Appendix B — CLAUDE.md Starter Template

```
# Project Configuration

## Stack
- Language: [Python 3.12 / TypeScript 5.x]
- Framework: [FastAPI / Next.js]
- Database: [PostgreSQL / SQLite]
- Deployment: [Docker / Vercel]

## Commands
- npm run dev    -- Start dev server
- npm test       -- Run test suite
- npm run build  -- Production build
- npm run lint   -- Lint and format

## Code Conventions
- TypeScript strict mode
- Functional components with hooks
- Tailwind for styling
- Tests colocated next to source

## Architecture
- /src/components  -- React components
- /src/lib         -- Shared utilities
- /src/api         -- Route handlers
- /src/types       -- Type definitions

## Notes
- Run tests before committing
- Conventional commits (feat:, fix:)
- Never commit .env files
```

Agent Skills Architecture Guide

Compiled by s.coy
@shaneswrld_ | github.com/shane9coy

Claude Code / KiloCode / OpenAI Codex / OpenClaw / Ollama / vLLM

v4.1 — February 2026