A RESOURCE FOR TEAMS WORKING WITH CODING AGENTS

Skill file organization.

One folder, one SKILL.md, one description that does the work.

Name it. Describe it. Load it only when needed.

28	34	03
rules, each with its check	checks in the kit's validator	levels of progressive disclosure

The description decides whether the skill exists.

A skill is a folder with a `SKILL.md`: YAML frontmatter that says what it is and when to use it, and instructions that load only when it runs [S01, S03]. Until then Claude sees one line, the description. A vague one means the skill never triggers; a sprawling body means it costs thousands of tokens every time it does.

Skills follow the Agent Skills open standard, and Claude Code extends it with fields the standard does not have [S03]. Those extensions are fine in a repository and rejected when you upload the same folder to claude.ai or the Skills API. This guide covers the layout, the frontmatter, the description and the evaluation, and marks which rules come from the standard and which from Claude Code.

Version 1.0 / Sources checked 24 September 2026
Field Guide 29 of the Design × AI series, with 28 (CLAUDE.md), 30 (verification) and 31 (context), all using the Acme UI design system. Checked against the Agent Skills specification and the Claude Code skills docs; cross-checked with `claude plugin validate` in Claude Code 2.1.271.

Practical guidance, not a standard. Field names, limits and load behavior come from the cited specification and docs; the validator's warnings for description style and reference depth encode published advice, not hard rules. Prepared with AI assistance and edited by hand.

START HERE

Learn the parts, then pick a route.

A skill has three layers, and each loads at a different moment. Most mistakes come from putting content in the wrong layer.

Part	Loads	Budget
`name` and `description`	At startup, for every skill, into the skill listing	About 100 tokens; description at most 1,024 characters
`SKILL.md` body	When the skill is invoked; stays for the session	Under 500 lines, under about 5,000 tokens
`references/`, `assets/`	When the body tells Claude to read them	No fixed limit; one topic per file
`scripts/`	Executed, not read into context	Only their output costs tokens

Progressive disclosure as the Agent Skills specification defines it [S01], with the Claude Code limits [S03].

Writing a first skill

Copy `skills/create-component/`, rename the folder, rewrite the description (section 04) and run the validator.

Converting flat Markdown files

A `.claude/skills/<name>.md` file is not a skill. Make a folder with `SKILL.md`, or keep the file in `.claude/commands/`, where the older format still works.

Sharing beyond Claude Code

Run the validator with `--portable`. Only six frontmatter fields survive an upload to claude.ai or the Skills API.

A skill that never triggers

Section 04, then the trigger queries in item K26. The body is irrelevant until the description works.

Label	Meaning
`SPEC`	Required or recommended by the Agent Skills specification; applies in every client.
`DOCS`	Claude Code or Anthropic platform documentation.
`KIT CHECK`	Checked by `validate-skill.mjs`; the rule id is in the evidence line.
`PRACTICE`	A working method; the evidence line says what to review.

SECTION 01

Lay out the folder.

The folder is the unit. Its name, its location and its subfolders decide how the skill is invoked and what it can carry.

- [ ] **One folder per skill, with SKILL.md exactly**

  K01 SPEC KIT CHECK

  `create-component/SKILL.md`, not `create-component.md` and not `skill.md`. On a case-insensitive disk the lowercase name looks fine locally and breaks on Linux.

  Evidence: Validator rule `skill-file`: the bad fixture's `skill.md` is an error.

  Directory structure. S01, S03.

- [ ] **Put it where it should load**

  K02 DOCS

  `.claude/skills/` in the repo for the team, `~/.claude/skills/` for you, `<plugin>/skills/` to distribute. Skills in a subfolder's `.claude/skills/` load once Claude works on files there.

  Evidence: The skill appears in `/skills` in a fresh session.

  Where skills load. S03.

- [ ] **The folder name is the command**

  K03 DOCS KIT CHECK

  `.claude/skills/create-component/` becomes `/create-component`. In Claude Code, `name` is only a display label; the spec requires it to equal the folder name, so keep them identical.

  Evidence: Validator rule `name-matches-folder`: a warning in Claude Code, an error with `--portable`.

  How a skill gets its command name. S01, S03.

- [ ] **Supporting files go in scripts, references and assets**

  K04 SPEC

  Code to run in `scripts/`, documentation to read in `references/`, templates and data in `assets/`, evaluations in `evals/`. The first three are the spec's conventions; `evals/` is where its evaluation guide keeps test cases.

  Evidence: The worked skill uses all four folders.

  Optional directories. S01, S10.

- [ ] **Workflows are skills and hooks are settings**

  K05 DOCS

  There is no `.claude/workflows/` or `.claude/hooks/*.md`. A workflow that chains skills is another skill; a hook is JSON in `settings.json` (Field Guide 30). Never name a folder `synced`, which Claude Code reserves.

  Evidence: Validator rule `folder-reserved`; review for Markdown files that pretend to be hooks.

  Skills, hooks and the reserved name. S03, S07.

If the folder is wrong, nothing inside it matters.

SECTION 02

Write frontmatter that parses and means something.

Claude Code ignores what it does not recognize, without an error. That makes a typo invisible until the skill misbehaves.

- [ ] **Names follow the spec**

  K06 SPEC KIT CHECK

  1 to 64 characters: lowercase letters, digits and hyphens, no leading, trailing or double hyphen. Gerunds (`creating-components`) and actions (`create-component`) both work; pick one pattern per team.

  Evidence: Validator rule `name-format`. The bad fixture's "Component Helper" fails.

  Name field. S01. Naming conventions. S04.

- [ ] **No reserved words in the name**

  K07 DOCS KIT CHECK

  The Claude API and claude.ai reject names containing `anthropic` or `claude`. A repo-only skill will load, but it cannot be shared later without renaming.

  Evidence: Validator rule `name-reserved-word`.

  Skill structure requirements. S04.

- [ ] **Know which profile you ship**

  K08 SPEC DOCS KIT CHECK

  The spec has six fields: `name`, `description`, `license`, `compatibility`, `metadata`, `allowed-tools`. Claude Code adds 14 more, such as `argument-hint`. An upload with an extra field fails with an "Unexpected key" error.

  Evidence: The worked skill passes the Claude Code profile and fails `--portable` with 1 error, `argument-hint`.

  Using frontmatter outside Claude Code. S03.

- [ ] **Decide who may invoke it**

  K09 DOCS KIT CHECK

  `disable-model-invocation: true` for side effects you time yourself, such as deploys. `user-invocable: false` for background knowledge. Both together mean nobody can invoke it.

  Evidence: Validator rule `invocation-dead`: an error on the bad fixture.

  Control who invokes a skill. S03.

- [ ] **Frontmatter errors are silent**

  K10 DOCS KIT CHECK

  An unknown field is ignored. If the YAML does not parse, the skill loads with no fields at all, so `/name` works but Claude never matches the description. `allowed_tools` with an underscore is a common case.

  Evidence: Validator rules `frontmatter-parse` and `unknown-field` (with a "did you mean" hint).

  Frontmatter reference and troubleshooting. S03.

A misspelled field fails quietly. The validator makes it loud.

SECTION 03

Control how it runs.

A few fields change what the skill may do once it is running. Each is powerful, and each has a narrower scope than people assume.

- [ ] **allowed-tools pre-approves one turn**

  K11 DOCS

  The listed tools run without a prompt during the turn that invokes the skill; the grant clears with your next message. It does not restrict other tools, and workspace trust does not gate it: review it in repos you clone.

  Evidence: The worked skill pre-approves only its own scaffold script.

  Pre-approve tools for a skill. S03.

- [ ] **context: fork only for self-contained tasks**

  K12 DOCS KIT CHECK

  A forked skill runs in a subagent that does not see the conversation. Guidelines without a task come back empty. `agent` and `background` apply only with `context: fork`.

  Evidence: Validator rules `context-value` and `fork-only-field`.

  Run skills in a subagent. S03, S09.

- [ ] **Arguments are explicit**

  K13 DOCS

  Use `$ARGUMENTS` or named `arguments` and give an `argument-hint` such as `[ComponentName]`. The body says what to do when the argument is missing.

  Evidence: The worked skill asks for a PascalCase name if none was given.

  Pass arguments to skills. S03.

- [ ] **Deterministic work is a script**

  K14 DOCS PRACTICE

  File scaffolding, validation and formatting belong in `scripts/`, called through `${CLAUDE_SKILL_DIR}` so the path works from any directory. Scripts are executed, not loaded, and they behave the same every time.

  Evidence: `scaffold-component.mjs` writes five files and refuses to overwrite; the kit's tests run it.

  String substitutions; scripts executed, not loaded. S01, S03.

Scope the power to the turn, the task and the script.

SECTION 04

Write a description that triggers.

Claude reads the description, and only the description, when deciding whether to load a skill. It carries the entire burden of triggering [S02].

- [ ] **What it does and when to use it, in the third person**

  K15 DOCS KIT CHECK

  "Scaffolds a new Acme UI component ... Use when the user asks to add, create or build a component." Not "I can help" and not "you can use this": the description is injected into the system prompt.

  Evidence: Validator rules `description-when` and `description-third-person`: both fire on the bad fixture.

  Writing effective descriptions. S04.

- [ ] **Use the words users actually type**

  K16 SPEC PRACTICE

  Cover requests that never name the domain: "we need a Tooltip" is a component request. Say so: "even if they only name it".

  Evidence: At least half the should-trigger queries in `evals/trigger-queries.json` avoid the skill's own vocabulary.

  Optimizing descriptions. S02.

- [ ] **Name the near misses**

  K17 SPEC

  State what it is not for: "Not for changing an existing component or for app code outside packages/ui." Near-miss queries are the most useful negative tests.

  Evidence: The worked skill's trigger file has 10 should-not-trigger queries, all near misses.

  Should-not-trigger queries. S02.

- [ ] **Put the key use case first**

  K18 DOCS KIT CHECK

  Claude Code caps each entry at 1,536 characters, `description` plus `when_to_use`, and the whole listing at 1% of the context window. With many skills, the least-used lose their descriptions first.

  Evidence: Validator rule `listing-length`. The worked skill's description is 373 characters.

  Skill descriptions are cut short. S03.

- [ ] **Within the spec's limits**

  K19 SPEC KIT CHECK

  Non-empty, at most 1,024 characters, no XML tags. Longer than 40 characters in practice: "Helps with PDFs" is the spec's own example of a poor description.

  Evidence: Validator rules `description-length`, `description-xml`, `description-too-short`.

  Description field. S01, S04.

The body is irrelevant until the description works.

SECTION 05

Disclose progressively.

Once loaded, the body stays in context for the rest of the session, so every line is a recurring cost [S03]. Keep the body a table of contents for the task.

- [ ] **The body is short and says what to do**

  K20 SPEC DOCS KIT CHECK

  Under 500 lines and about 5,000 tokens. State what to do, not why the design system exists. The worked skill's body is 37 lines.

  Evidence: Validator rules `body-length` and `body-tokens` (characters divided by four, an estimate).

  Progressive disclosure. S01, S03.

- [ ] **Every referenced file exists**

  K21 KIT CHECK

  Links, `${CLAUDE_SKILL_DIR}` paths and prose paths under `scripts/`, `references/`, `assets/` and `evals/` must resolve. Commands inside code fences are treated as repository paths, not skill paths.

  Evidence: Validator rule `reference-missing`: 2 errors on the bad fixture.

  File references. S01.

- [ ] **References are one level deep**

  K22 SPEC DOCS KIT CHECK

  SKILL.md links to each reference directly, with a line on when to read it. A reference that links on to another file risks a partial read.

  Evidence: Validator rule `reference-depth`: `everything.md` linking to `details.md` is flagged.

  Keep references one level deep. S01, S04.

- [ ] **Long references have a table of contents**

  K23 DOCS KIT CHECK

  A reference over 100 lines starts with its contents, so a preview still shows its scope. Use forward slashes in every path.

  Evidence: Validator rules `reference-toc` and `forward-slashes`.

  Structure longer reference files. S04.

- [ ] **What matters most comes first**

  K24 DOCS

  After compaction Claude Code re-attaches each invoked skill's first 5,000 tokens, within 25,000 tokens for all skills. Truncation keeps the start of the file.

  Evidence: The worked skill's steps and verify commands are in its first 40 lines.

  Skill content lifecycle; what survives compaction. S03, S06.

SKILL.md is the index; the folder is the book.

SECTION 06

Prove it works.

Seeing a skill trigger tells you Claude found it, not that it helped [S03]. Check the folder in CI, and measure triggering and output separately.

- [ ] **The folder is validated in CI**

  K25 KIT CHECK

  Run `validate-skill.mjs .claude/skills` on every change. `claude plugin validate` checks that the YAML parses: it passes a copy of the bad fixture renamed to SKILL.md, on which the kit reports 6 errors and 8 warnings.

  Evidence: A CI job; exit code 1 on any error.

  Kit script; `claude plugin validate` run with Claude Code 2.1.271. S08.

- [ ] **Triggering is measured with labeled queries**

  K26 SPEC KIT CHECK

  About 20 realistic queries, half should-trigger and half near misses, each run three times, split into train and validation sets so the description does not overfit.

  Evidence: Validator rule `trigger-queries` checks the file's shape and balance. The worked skill has 10 and 10. The kit does not run them.

  Designing trigger eval queries. S02.

- [ ] **Output is compared with and without the skill**

  K27 DOCS SPEC

  Run each eval prompt in a fresh session with the skill and with it disabled, and grade both against assertions. skill-creator and `claude plugin eval` automate the loop.

  Evidence: `evals/evals.json` with three cases and assertions (not executed in the kit).

  Evaluate and iterate. S03, S10.

- [ ] **Unused skills are pruned**

  K28 DOCS

  Every listed skill costs context on every turn. `/skill-doctor` reports cost and use; set rarely used skills to `name-only` in `skillOverrides`, or remove them.

  Evidence: A quarterly `/skill-doctor` run in the review record.

  Find unused skills; override visibility. S03.

A skill is done when it triggers on the right prompts and beats the baseline.

APPENDIX A

A design-system skill, file by file.

`skills/create-component/` scaffolds an Acme UI component whose contract follows Field Guide 02 and whose styles follow Field Guide 03. It passes the validator with no errors or warnings.

skills/create-component/

```
create-component/
  SKILL.md                    37-line body: inputs, steps, verify, report
  references/
    contract-fields.md        contract fields in fill order (Field Guide 02)
    token-rules.md            the token rules a component touches (Field Guide 03)
  assets/                     five templates: tsx, css, contract, story, test
  scripts/
    scaffold-component.mjs    writes the files, adds the export, never overwrites
  evals/
    trigger-queries.json      10 should-trigger, 10 near-miss queries
    evals.json                3 output cases with assertions
```

skills/create-component/SKILL.md (frontmatter)

```
---
name: create-component
description: Scaffolds a new Acme UI component in packages/ui with its source file,
  contract, story, test and index export, following the design system's contract and
  token rules. Use when the user asks to add, create or build a component for the
  design system, even if they only name it ("we need a Tooltip"). Not for changing an
  existing component or for app code outside packages/ui.
argument-hint: "[ComponentName]"
allowed-tools: Bash(node ${CLAUDE_SKILL_DIR}/scripts/scaffold-component.mjs *)
---
```

The file keeps the description on one line; wrapped here for print. `argument-hint` is the one Claude Code-only field.

APPENDIX B

Run the validator.

`node scripts/validate-skill.mjs <skill-folder | skills-folder>`, zero dependencies. Output below is from the kit.

terminal

```
$ node scripts/validate-skill.mjs skills/create-component
1 skill(s), 0 error(s), 0 warning(s) [claude-code]

$ node scripts/validate-skill.mjs --portable skills/create-component
SKILL.md  error  unknown-field  Field `argument-hint` is not in the Agent Skills spec
1 skill(s), 1 error(s), 0 warning(s) [portable]

$ node scripts/validate-skill.mjs fixtures/bad-skills/Component_Helper
skill.md     error    skill-file         The file is named skill.md; it must be SKILL.md
skill.md     error    invocation-dead    ... neither you nor Claude can invoke the skill.
skill.md     error    context-value      `context` accepts only `fork`.
skill.md     error    name-format        "Component Helper" may use only lowercase ...
skill.md     error    description-xml    `description` cannot contain XML tags.
skill.md:12  error    reference-missing  `reference/api.md` is referenced but ...
skill.md:15  error    reference-missing  `scripts/make-component.py` is referenced ...
skill.md     warning  unknown-field      `allowed_tools` ... Did you mean `allowed-tools`?
...
1 skill(s), 7 error(s), 8 warning(s) [claude-code]
```

Abbreviated. The portable profile reports 10 errors and 6 warnings on the same fixture. The kit validator follows the rules of the reference `skills-ref` validator [S05] and adds the Claude Code checks.

Group	Checks	Rules
File and folder	SKILL.md exact name, reserved folder, frontmatter present and parsing	4
Fields and evals	Known fields per profile, value types, invocation, fork-only fields, trigger file	11
Name	Present, format, matches folder, reserved words	4
Description	Present, length, listing cap, XML, too short, when clause, person	7
Body and references	Empty, lines, tokens, slashes, missing, outside, depth, contents	8

34 rules in all, listed with their severity in `RULES` at the top of `validate-skill.mjs`.

KEEP WITH THE SKILLS FOLDER

Leave a skill review record.

One record per skill release. It shows whether the skill triggers, whether it helps, and what it costs.

Skill / folder	**Owner**
Validator run, both profiles (link)	**Description length / listing length**
Trigger rate: should / should not	**Output evals with and without (link)**
Body lines / estimated tokens	**Referenced files checked**
/skill-doctor cost and use	**Next review date**

> **Measure the description before the body.**
> A body nobody loads has no effect. Fix triggering first, then output quality.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. The Agent Skills specification is an open standard; Claude Code documents which fields it adds. Where the two differ, this guide says which applies.

S01 / Agent Skills, specification
Directory structure, the six frontmatter fields, name and description limits, progressive disclosure, file references.
https://agentskills.io/specification

S02 / Agent Skills, optimizing skill descriptions
Imperative, intent-focused descriptions; about 20 trigger queries; near misses; train and validation split.
https://agentskills.io/skill-creation/optimizing-descriptions

S03 / Claude Code docs, extend Claude with skills
Locations, command names, full frontmatter reference, invocation control, allowed-tools, context: fork, listing budget, lifecycle, evaluation, /skill-doctor.
https://code.claude.com/docs/en/skills

S04 / Anthropic, skill authoring best practices
Third person, reserved words, naming patterns, one-level references, tables of contents, forward slashes.
https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices

S05 / Agent Skills, skills-ref reference validator
The reference implementation of name, description and field checks.
https://github.com/agentskills/agentskills/tree/main/skills-ref

S06 / Claude Code docs, explore the context window
Skill bodies after compaction: 5,000 tokens each, 25,000 in total.
https://code.claude.com/docs/en/context-window

S07 / Claude Code docs, extend Claude Code
Skill versus hook versus subagent; context cost by feature.
https://code.claude.com/docs/en/features-overview

S08 / Claude Code docs, plugins
Plugin skills are namespaced; `claude plugin validate`.
https://code.claude.com/docs/en/plugins

S09 / Claude Code docs, subagents
What a forked or preloaded skill sees at startup.
https://code.claude.com/docs/en/sub-agents

S10 / Agent Skills, evaluating skill output quality
evals/evals.json, assertions, with-and-without comparison.
https://agentskills.io/skill-creation/evaluating-skills

Maintenance: recheck the Claude Code frontmatter table and the Agent Skills specification each quarter; update `CLAUDE_CODE_FIELDS` and `PORTABLE_FIELDS` in the validator when they change. Update the PDF, HTML, Markdown and JSON together.