# Agent Skills

Skills are an open standard for extending agent capabilities. A skill is a folder containing a `SKILL.md` file with instructions that the agent can follow when working on specific tasks.

## What are skills?

Skills are reusable packages of knowledge that extend what the agent can do. Each skill contains:

- **Instructions** for how to approach a specific type of task
- **Best practices** and conventions to follow
- **Optional scripts and resources** the agent can use

When you start a conversation, the agent sees a list of available skills with their names and descriptions. If a skill looks relevant to your task, the agent reads the full instructions and follows them.

## Where skills live

Antigravity supports two types of skills:

| Location | Scope |
| --- | --- |
| `<workspace-root>/.agent/skills/<skill-folder>/` | Workspace-specific |
| `~/.gemini/antigravity/skills/<skill-folder>/` | Global (all workspaces) |

**Workspace skills** are great for project-specific workflows, like your team's deployment process or testing conventions.

**Global skills** work across all your projects. Use these for personal utilities or general-purpose tools you want everywhere.

## Creating a skill

To create a skill:

1. Create a folder for your skill in one of the skill directories
2. Add a `SKILL.md` file inside that folder

Every skill needs a `SKILL.md` file with YAML frontmatter at the top:

```
---
name: my-skill
description: Helps with a specific task. Use when you
need to do X or Y.
---

# My Skill

Detailed instructions for the agent go here.

## When to use this skill

- Use this when...
- This is helpful for...

## How to use it

Step-by-step guidance, conventions, and patterns the
agent should follow.
```

## Frontmatter fields

| Field | Required | Description |
|---|---|---|
| `name` | No | A unique identifier for the skill (lowercase, hyphens for spaces). Defaults to the folder name if not provided. |
| `description` | Yes | A clear description of what the skill does and when to use it. This is what the agent sees when deciding whether to apply the skill. |

Tip: Write your description in third person and include keywords that help the agent recognize when the skill is relevant. For example: "Generates unit tests for Python code using pytest conventions."

```
.agent/skills/my-skill/
├── SKILL.md        # Main instructions (required)
├── scripts/        # Helper scripts (optional)
├── examples/       # Reference implementations
(optional)
└── resources/      # Templates and other assets
(optional)
```

The agent can read these files when following your skill's instructions.

## How the agent uses skills

Skills follow a **progressive disclosure** pattern:

1. **Discovery**: When a conversation starts, the agent sees a list of available skills with their names and descriptions
2. **Activation**: If a skill looks relevant to your task, the agent reads the full `SKILL.md` content
3. **Execution**: The agent follows the skill's instructions while working on your task

You don't need to explicitly tell the agent to use a skill—it decides based on context. However, you can mention a skill by name if you want to ensure it's used.

## Best practices

### Keep skills focused

Each skill should do one thing well. Instead of a "do everything" skill, create separate skills for distinct tasks.

### Write clear descriptions

The description is how the agent decides whether to use your skill. Make it specific about what the skill does and when it's useful.

### Use scripts as black boxes

If your skill includes scripts, encourage the agent to run them with `--help` first rather than reading the entire source code. This keeps the agent's context focused on the task.

## Example: A code review skill

Here's a simple skill that helps the agent review code:

```
---
name: code-review
description: Reviews code changes for bugs, style
issues, and best practices. Use when reviewing PRs or
checking code quality.
---

# Code Review Skill

When reviewing code, follow these steps:

## Review checklist

1. **Correctness**: Does the code do what it's supposed
to?
2. **Edge cases**: Are error conditions handled?
3. **Style**: Does it follow project conventions?
4. **Performance**: Are there obvious inefficiencies?

## How to provide feedback

- Be specific about what needs to change
- Explain why, not just what
- Suggest alternatives when possible
```

‹ Rules / Workflows

Task Groups ›