Agent Skills: Interconnected AI agents sharing knowledge and capabilities across platforms

Agent Skills: The Universal Standard Transforming How AI Agents Work

The Promise of AI Agents, and the Problem

24 min read · 6 days ago

Rick Hightower Follow

Listen Share

A comprehensive guide to the open standard for AI agent capabilities, now supported by 25+ agent frameworks

The Promise of AI Agents, and the Problem

Imagine spending weeks building perfect workflows in Claude Code: code review checklists, refactoring patterns, and deployment procedures. Your team's hard-won

expertise is finally encoded in a way AI can use. Then your company switches to Gemini CLI, and everything is lost. You start over from scratch.

AI coding assistants have transformed how developers work. Tools like Claude Code, Cursor, Gemini CLI, and GitHub Copilot can read codebases, write functions, debug issues, and even refactor entire modules. But there's a critical limitation: each tool operates in its own silo.

If you've built workflows in Claude Code, they don't carry over to the Gemini CLI. If your team has defined best practices for code reviews in Cursor, those practices don't carry over to VS Code. Every time you switch tools, you lose the investment you've made in training your AI assistant to understand how your team works.

Agent Skills solves this problem by providing a universal standard for packaging and sharing AI agent capabilities. Think of it as npm or pip for AI expertise: a format that works everywhere, packages knowledge consistently, and lets you share capabilities across tools and teams.

For a deep dive into how Agent Skills have evolved beyond coding tools into legal, finance, and healthcare domains, see our comprehensive coverage at Agent Skills: Beyond Coding.

What Are Agent Skills?

Agent Skills are lightweight folders that bundle instructions, scripts, and resources, enabling AI agents to discover and load capabilities on demand. At their core, skills packages encapsulate domain expertise in a format that works with any compatible agent.

The structure is deliberately simple:

```
my-skill/
├── SKILL.md          # Required: instructions + metadata
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
└── assets/           # Optional: templates, resources
```

What this structure accomplishes:

- **SKILL.md**: The intelligence layer. Contains both metadata (what the skill does) and instructions (how to use it)
- **scripts/**: Executable automation. Python, bash, or any language the agent can run
- **references/**: Supporting documentation that gets loaded only when needed
- **assets/**: Templates, configuration files, or other resources

Why this approach works:

This simple folder structure makes skills easy to create, version control, and share. No complex packaging systems, no build steps; just files that humans can read and AI agents can understand.

Understanding SKILL.md: The Heart of Every Skill

The `SKILL.md` file uses a two-part structure that balances discoverability with detailed guidance:

```
---
name: pdf-processing
description: Extract text and tables from PDF files, fill forms, merge document
---

# PDF Processing

## When to use this skill
Use this skill when the user needs to work with PDF files,
extract text from documents, fill out forms, or merge multiple PDFs.

## How to extract text
1. Use pdfplumber for text extraction...

## How to fill forms
...
```

Understanding the components:

1. **YAML Frontmatter** (lines 1–4): Contains metadata for discovery. The agent reads this first to determine if the skill matches the user's task.

- `name` : Unique identifier for the skill
- `description` : Clear, searchable summary of what the skill does

1. **Markdown Instructions** (everything after line 5): The detailed guidance the agent follows when the skill is activated. This can include:

- Step-by-step procedures
- Code examples with explanations
- Decision trees for choosing between approaches
- References to external documentation

Why this matters:

The two-part structure enables progressive disclosure (explained in the next section). Agents can know what a skill does without reading all its instructions, keeping context usage efficient.

Real-World Example: Legal Contract Review

Here's how a law firm might encode their contract review process:

```
---
name: contract-review-saas
description: Review SaaS contracts for red flags in liability, IP rights, and t
---

# SaaS Contract Review Skill

## When to use this skill
Use when reviewing software-as-a-service agreements, subscription contracts,
or cloud service agreements.

## Review checklist

### 1. Liability Limitations
- Check liability cap (should be at least 12 months of fees)
- Verify exclusions for data breaches (red flag if excluded)
- Confirm indemnification coverage for IP claims

### 2. Intellectual Property
- Verify customer owns data created in the system
- Check for broad license grants (red flag)
- Confirm no automatic IP transfer to vendor
```

```
### 3. Termination Rights
- Verify data export provisions (30 days minimum)
- Check for auto-renewal terms (red flag if >1 year)
- Confirm termination for convenience exists

## Output format
Create a summary table with: [Clause] | [Status: Green/Yellow/Red] | [Notes]
```

What makes this effective:

- **Domain expertise encoded:** Years of legal experience distilled into a checklist
- **Actionable guidance:** Clear criteria for red flags versus acceptable terms
- **Consistent output:** Standardized format for review results
- **Portable knowledge:** Works with any AI agent that supports skills

This isn't just about code. Agent Skills can encode any domain expertise: legal review checklists, data analysis pipelines, compliance workflows, documentation standards, or onboarding procedures. As the official specification states, skills are "a simple, open format for giving agents new capabilities and expertise."

Progressive Disclosure: The Key Innovation

Progressive disclosure is the architectural pattern that keeps Agent Skills fast and efficient. Here's the problem it solves: if an agent loaded complete instructions for every available skill at startup, a collection of 50 skills could consume 250,000 tokens (50 skills × 5,000 tokens each). This would slow down every conversation and waste context on irrelevant capabilities.

Progressive disclosure loads information in three phases, each consuming only what's needed for that stage.

Understanding the Three-Phase Architecture

Phase 1: Discovery (approximately 100 tokens per skill)

At startup, agents scan available skills and load only the `name` and `description` metadata from the YAML frontmatter. This creates an index of capabilities without reading full instructions.

Real-world example:

```
---
name: pdf-processing
description: Extract text and tables from PDF files, fill forms, merge document
---
```

The agent reads these two lines (about 20 words, or 30 tokens) and now knows this skill exists. With 50 skills, the total cost is approximately 5,000 tokens; manageable for any agent.

Why this matters: The agent can answer "what can you help me with?" without loading any actual skill logic.

Phase 2: Activation (under 5,000 tokens recommended)

When a user asks "extract the tables from this quarterly report PDF," the agent matches the request to the `pdf-processing` skill description. Only then does it read the complete `SKILL.md` file into context.

What gets loaded:

- Full instructions from the Markdown section
- Decision trees for choosing between approaches
- Code examples and usage patterns
- Error handling guidance

Why this matters: The agent gets deep expertise exactly when needed, not before.

Phase 3: Execution (as needed)

As the agent follows the skill's instructions, it may need supporting resources:

- Load scripts from the `scripts/` folder when ready to execute
- Read documentation from `references/` when encountering edge cases
- Access templates from `assets/` when generating output

Real-world scenario:

```
User: "Extract tables from quarterly_report.pdf"
│
├─> Phase 1: Agent scans 50 skill descriptions (5,000 tokens total)
│   └─> Matches "pdf-processing" skill
│
├─> Phase 2: Loads pdf-processing/SKILL.md (4,200 tokens)
│   └─> Reads instructions for table extraction
│
└─> Phase 3: Executes scripts/extract_tables.py
    └─> Loads references/table_formats.md for Excel conversion
```

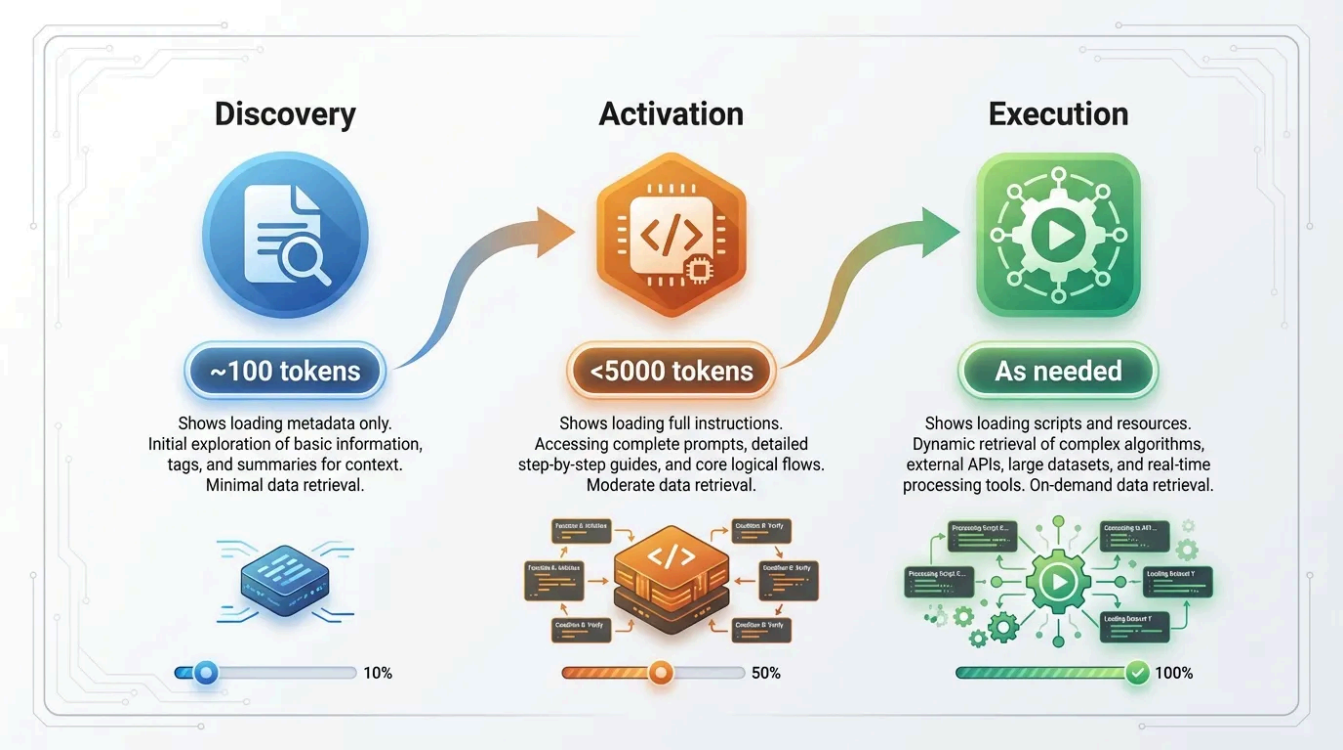
The Efficiency Advantage

Here's the token math:

Approach	Startup Cost	Task Cost	Total
Load everything	250,000 tokens	0 tokens	250,000
Progressive disclosure	5,000 tokens	4,200 tokens	9,200

Progressive disclosure uses 96% fewer tokens for the same capabilities. This translates to:

- Faster agent startup
- More context available for actual work
- Lower API costs
- Better responsiveness



Progressive Disclosure Architecture: Skills load in three phases — Discovery loads metadata only (~100 tokens per skill), Activation loads full SKILL.md instructions (<5000 tokens), and Execution loads scripts, references, and assets as needed

Trade-offs to consider:

Pros:

- Dramatically reduced token usage for agents with many skills
- Faster startup and response times
- Scales to hundreds of available skills without performance degradation

Cons:

- Requires agents to implement the discovery/activation pattern
- Slight complexity in the loading mechanism
- Skills must keep instructions concise (5,000 token recommendation)

When progressive disclosure excels:

- Agents with 10+ skills (the efficiency gains are substantial)
- Environments with token limits or cost concerns

- Production deployments where speed matters

Alternatives:

- **Load all skills:** Simpler but impractical beyond 5–10 skills
- **Manual skill activation:** User explicitly loads skills, but reduces discoverability

The specification chose progressive disclosure because it scales to hundreds of skills while maintaining fast, responsive agents. This architectural decision makes Agent Skills viable for enterprise deployments where teams might have dozens or hundreds of specialized capabilities.

Why Agent Skills Matter

1. Domain Expertise Beyond Code

The real power of Agent Skills emerges when you realize they're not limited to programming tasks. They can package any procedural knowledge that can be articulated in instructions.

Real-world examples across industries:

Legal workflows: A law firm might create a `contract-review-saas` skill encoding their standard checklist for reviewing software-as-a-service agreements. The skill ensures every junior attorney checks liability caps, IP ownership, and termination clauses consistently. What used to require a senior partner's review now happens automatically with AI assistance.

Financial analysis: A hedge fund creates an `earnings-report-analysis` skill that walks through their proprietary analysis framework: extract key metrics, compare to guidance, analyze margin trends, flag unusual items. The same analytical rigor applies whether you're analyzing tech companies or manufacturers.

Healthcare compliance: A hospital encodes HIPAA compliance checks as a `phi-data-handler` skill. When developers work on patient data features, the AI automatically follows proper anonymization, audit logging, and access control patterns. Compliance becomes built into the workflow, not a separate review step.

Operations runbooks: A DevOps team creates skills for incident response: `incident-triage`, `database-recovery`, `traffic-spike-mitigation`. When an alert fires at 3 AM,

even junior engineers have access to the team's accumulated expertise through their AI assistant.

What makes this valuable:

Skills preserve and scale expertise that traditionally lived only in senior team members' heads. A legal partner's contract review approach, a senior analyst's investigation framework, or a principal engineer's debugging methodology can now assist everyone on the team.

2. Write Once, Deploy Everywhere

The same skill folder works across Claude Code, Gemini CLI, OpenCode, Cursor, GitHub Copilot, and 20+ other platforms. This interoperability fundamentally changes the economics of building AI workflows.

Skill Portability: A single skill file deploys seamlessly across multiple AI coding platforms

What this enables:

No vendor lock-in: Your company standardizes on Cursor, but a team member prefers VS Code with GitHub Copilot. Both can use your company's skills. No need to maintain parallel implementations or force tool standardization.

Real scenario: A consulting firm builds skills for their standard deliverables (architecture reviews, code audits, security assessments). Consultants use whatever

tool they prefer, but everyone delivers consistent quality because the skills encode the firm's methodology.

Team consistency: When your code review skill checks for proper error handling, logging, and test coverage, it applies the same standards whether you're using Claude Code, Gemini CLI, or Cursor. Teams get consistency without sacrificing tool choice.

Real scenario: A distributed team across three continents uses different AI coding assistants based on regional availability and preference. Their shared skill library ensures everyone follows the same architectural patterns and review standards.

Knowledge preservation: Your team's accumulated expertise lives in version-controlled skill files, not scattered across tool-specific configurations. When a senior engineer leaves, their debugging techniques and code review standards remain encoded in skills.

Real scenario: A startup's first principal engineer creates skills encoding their preferred patterns for API design, database migrations, and testing strategies. Two years later, after the engineer has moved on, new team members still benefit from that expertise through the skills they left behind.

Trade-offs of the universal approach:

Pros:

- True portability across 25+ agents and counting
- Investment in skills pays off even when switching tools
- Easier to share expertise across teams using different tools

Cons:

- Skills can't leverage agent-specific features or APIs
- Must work within the common denominator of skill capabilities
- May not be as optimized as tool-specific solutions

When universal skills excel:

- Teams with mixed tool preferences
- Organizations that want to avoid vendor lock-in
- Long-term knowledge preservation (outlasts any single tool)

Alternatives:

- Tool-specific plugins: More powerful for single tools, but not portable
- Documentation: Universal but requires manual interpretation
- Code generation templates: Portable but less intelligent than skills

The specification chose universal compatibility over tool-specific optimization because portability provides more long-term value than specialized features. A skill that works everywhere for 10 years beats a slightly better implementation that only works with one tool for 2 years.

3. Portable and Version-Controlled

Skills are just files. This simple fact unlocks powerful workflows that feel natural to developers but are revolutionary for AI capabilities.

What you can do with skills:

Version control with Git:

```
git clone git@github.com:yourcompany/skills.git
cd skills
ls
# legal-review/
# security-audit/
# api-design/
# database-migration/
```

Your team's skills live in a repository just like your code. Every change is tracked, every contributor is credited, and the full history is preserved.

Review through pull requests:

```
Pull Request #127: Enhance security-audit skill with OAuth 2.1 checks

Changes:
- Add checks for PKCE requirement
- Flag deprecated grant types
- Verify token expiration settings

Reviewers: @security-lead, @senior-engineer
```

Skills improve through the same code review process your team already uses. Security leads review security skills, API architects review API skills, and everyone learns from the discussions.

Semantic versioning:

```
git tag -a legal-review-v2.0.0 -m "Add GDPR compliance checks"
git push origin legal-review-v2.0.0
```

Teams can pin to specific skill versions for stability or upgrade to get new capabilities. Just like dependency management for code libraries.

Cross-team sharing:

```
# Finance team shares their analysis skills
git push origin main

# Engineering team imports them
skilz install mycompany/skills/financial-analysis
```

The same skill repository can serve multiple teams. Finance develops analysis skills, engineering develops code-review skills, legal develops contract-review skills, and everyone benefits from the full library.

Publishing to marketplaces:

Upload to SkillzWave Marketplace to share with the broader community:

- Open source your best practices
- Build reputation for your methodology
- Benefit from community improvements

Real-world scenario:

A fintech company maintains a skill repository with:

- `pci-compliance-check` : Ensures payment handling follows PCI DSS
- `financial-calculation` : Standardizes interest, fee, and tax calculations
- `audit-log-generator` : Creates compliant audit trails for financial transactions

When regulators ask "how do you ensure consistent compliance?", the company points to version-controlled skills with a clear audit trail of who reviewed and approved each change. When a compliance requirement changes, they update one skill and all developers benefit immediately.

Why this matters:

Traditional AI workflows live in isolated configurations, tool settings, or undocumented practices. Skills make AI capabilities as manageable as code: reviewable, testable, improvable, and shareable.

The Ecosystem: Coordinated Standards

Agent Skills don't exist in isolation. They're part of a coordinated ecosystem of standards managed by the Agentic AI Foundation, each solving a distinct piece of the AI agent puzzle.

Understanding the Standard Relationships

Think of these standards as layers in a stack:

Model Context Protocol (MCP): The Data Access Layer

MCP answers: "What data sources and tools can this agent access?"

Released by Anthropic in November 2024, MCP is an open protocol for connecting AI models to databases, APIs, and external tools. With 97 million monthly SDK downloads, it's become the standard way agents connect to data.

Real example: An MCP server exposes your company's Postgres database, Jira API, and Slack workspace to AI agents. Any MCP-compatible agent can now query databases, create tickets, and send messages.

Agent Skills: The Expertise Layer

Skills answer: "How should this agent perform specific tasks?"

While MCP gives agents access to tools, skills teach them how to use those tools effectively. A skill might say "when analyzing customer churn, first query the database for usage patterns, then check support tickets for complaints, then calculate the churn risk score using this formula."

Real example: A `customer-health-analysis` skill uses MCP to access your CRM and support system, but the skill encodes your company's specific methodology for identifying at-risk customers.

AGENTS.md: The Project Context Layer

AGENTS.md answers: "What are the conventions and structure of this specific project?"

Launched in August 2025, AGENTS.md provides project-specific context. Over 60,000 open-source projects have adopted this format for communicating build instructions, architectural decisions, and coding conventions to AI assistants.

Real example: Your project's AGENTS.md specifies that you use React for frontend, Postgres for database, and Jest for testing. It documents your custom authentication approach and where different features live in the codebase. The agent reads this once per project.

How the Standards Work Together

Here's a concrete scenario showing all three standards in action:

```
Developer: "Analyze why customers from the enterprise plan are churning"
│
├─> AGENTS.md provides project context:
│    - Database: Postgres via MCP server on port 5432
│    - Customer data in analytics.customers table
│    - Support tickets in support.tickets table
│
```

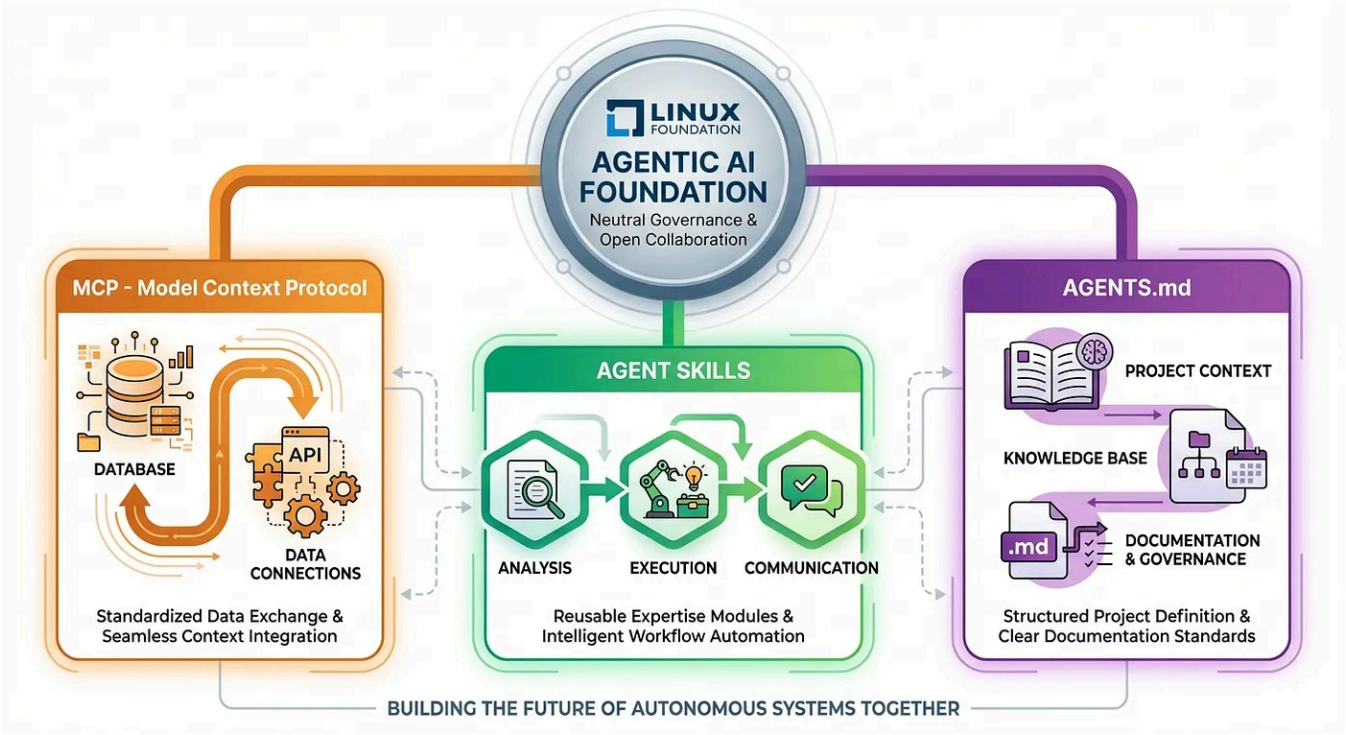
```
    ├─> Agent Skills provides methodology:
    │    - Load customer-health-analysis skill
    │    - Follow the analysis framework: usage → support → finance
    │    - Calculate risk score using company formula
    │
    └─> MCP provides data access:
         - Connect to Postgres MCP server
         - Query customer usage metrics
         - Fetch related support tickets
         - Pull payment history
```

Why this coordination matters:

Each standard focuses on its domain, avoiding overlap and redundancy:

- **MCP** handles the infrastructure layer (connectivity)
- **Agent Skills** handles the knowledge layer (methodology)
- **AGENTS.md** handles the project layer (local context)

An agent using all three can access your data (MCP), follow your company's analysis methodology (Skills), and understand your specific project structure (AGENTS.md).



Agentic AI Foundation Ecosystem: Three complementary standards under neutral governance — MCP for data and tool access, Agent Skills for workflows and expertise, and AGENTS.md for project context

Platform Adoption

As of January 2026, 25+ major platforms support Agent Skills across the AI landscape:

Coding Assistants:

- Claude Code, Cursor, VS Code, GitHub Copilot
- Gemini CLI, OpenCode, Amp, Goose

AI Platforms:

- OpenAI Codex, Mistral AI Vibe, Qwen Code
- Piebald, Agentman, Command Code

Infrastructure & Data:

- Databricks, Spring AI, Factory
- Letta, Roo Code, TRAE

Why this adoption rate matters:

When 25+ platforms support the same standard, it creates network effects. Skills shared on GitHub or SkillzWave Marketplace immediately work across the entire ecosystem. Developers can invest in creating high-quality skills knowing they'll work with current and future tools.

The Agentic AI Foundation

In December 2025, the Linux Foundation announced the formation of the Agentic AI Foundation (AAIF). This organization provides neutral governance ensuring these standards evolve through community consensus rather than vendor control.

Founding platinum members:

- Amazon Web Services
- Anthropic
- Block
- Bloomberg

- Cloudflare
- Google
- Microsoft
- OpenAI

What neutral governance provides:

No single vendor control: Standards evolve based on community needs, not one company's product roadmap. If Amazon, Google, Microsoft, and OpenAI all have equal input, no single vendor can hijack the standard for competitive advantage.

Long-term stability: The Linux Foundation has governed open source projects for decades. Skills you create today will have a stable specification for years to come.

Transparent evolution: Changes to standards go through open review processes. Anyone can propose improvements, and decisions happen in public forums.

Cross-vendor compatibility: When all major AI providers participate in governance, they're committed to making their agents work with the standard. This ensures your skills truly work everywhere.

Real-world impact:

Without neutral governance, we'd likely see fragmentation: "Claude Skills" that only work with Claude, "Gemini Skills" with different formatting, and "Copilot Skills" with incompatible structure. The AAIF prevents this Balkanization, ensuring one skill format works across all agents.

This coordination ensures the standards work together and remain vendor-neutral, protecting your investment in skills from being locked into any single platform.

Skilz CLI: Universal Skill Installation

While Agent Skills provide a universal format, each agent has its own installation location. Claude Code looks in `~/.claude/skills/`, Gemini CLI uses `~/.gemini/skills/`, OpenCode checks `~/.config/opencode/skill/`, and 30+ other agents each have their own convention. Managing skills across multiple agents quickly becomes a tedious game of copying files to different directories.

This is where Skilz CLI comes in. Think of it as “npm install for AI skills”: one command that installs skills to any agent, handles updates, and manages dependencies.

Installation

```
pip install skilz
```

What this installs: A Python command-line tool that knows where every major AI agent stores skills. It handles agent detection, path resolution, and cross-agent installation automatically.

Basic Usage

Automatic agent detection:

```
skilz install anthropics/skills/algorithmic-art
```

What happens here:

1. Skilz scans your system for installed AI agents (Claude Code, Gemini CLI, etc.)
2. Detects which agent you’re currently using (by checking environment variables and active processes)
3. Installs the skill to the correct directory for that agent
4. Confirms the installation succeeded

Why this is better than manual installation: You don’t need to remember that Claude uses `~/.claude/skills/` while Gemini uses `~/.gemini/skills/`. Skilz knows all 30+ agent locations.

Installing to a specific agent:

```
skilz install anthropics/skills/brand-guidelines --agent opencode
```

When to use this: If you want to install a skill for OpenCode but you're currently using Claude, specify the agent explicitly. Useful for team leads setting up standardized skill libraries across multiple agents.

Project-level installation:

```
skilz install anthropics/skills/pdf-reader --agent gemini --project
```

What this does:

- Installs to `.skilz/skills/` in your current directory instead of `~/.gemini/skills/`
- Makes the skill available only for this project
- Gets version-controlled with your project repository
- Ensures everyone on the team uses the same skills

When to use project-level vs user-level:

Installation Level	When to Use	Example
User-level	Personal productivity skills	Text editors, command generators, shell helpers
Project-level	Team-shared expertise	Code review checklist, API design standards, deployment procedures

Skill Management

View installed skills:

```
skilz list
```

Output:

```
Installed skills for claude:
  - algorithmic-art (v1.2.0)
  - brand-guidelines (v2.0.1)
  - pdf-reader (v1.5.3)
```

```
Installed skills for gemini (project):
  - api-design-standards (v3.1.0)
  - database-migration (v2.2.0)
```

Find skills on GitHub:

```
skilz search excel
```

What this searches: Scans GitHub repositories with the `agent-skill` topic for skills matching "excel". Returns skills for Excel file processing, formula generation, data analysis, and more.

Update all skills:

```
skilz update
```

What happens:

1. Checks each installed skill against its source repository
2. Downloads newer versions if available
3. Updates skills in place while preserving your local configurations
4. Reports which skills were updated

Remove a skill:

```
skilz uninstall theme-factory
```

Why you might uninstall:

- Skill no longer needed for current work
- Reducing clutter in skill discovery phase
- Replacing with an improved alternative

Understanding the Three-Tier Support System

Skilz supports 30+ agents through a clever three-tier architecture. Understanding which tier your agent uses helps you choose the right installation approach.

Tier 1: Full Native Support (15+ agents)

These agents have dedicated skill directories at both user and project levels. Skilz knows exactly where to install skills and the agent will discover them automatically.

Agent	User Level Path	Project Level Path
Claude Code	`~/.claude/skills/`	`.claude/skills/`
Gemini CLI	`~/.gemini/skills/`	`./gemini/skills/`
OpenCode	`~/.config/opencode/skill/`	`.opencode/skill/`
OpenAI Codex	`~/.codex/skills/`	`.codex/skills/`

Plus: Cline, Goose, Roo Code, Kilo Code, Trae, Windsurf, Qwen Code, and more.

When to use Tier 1 agents: If your preferred agent is in Tier 1, everything just works. Use `skilz install` without worrying about implementation details.

Tier 2: Bridged Support (8 agents)

These agents don't have native skill support yet, but Skilz provides a bridge that makes skills available:

- Aider
- Zed AI
- Crush
- Kimi CLI
- Plandex

How the bridge works:

1. Skilz installs skills to a universal location
2. Creates an AGENTS.md file that references the skills
3. The agent reads AGENTS.md and gains access to skill instructions
4. Updates propagate automatically when you modify skills

When to use Tier 2 agents: If you're using Aider or another bridged agent, skills work but with slight limitations. The agent may need to explicitly load skills rather than discovering them automatically.

With the latest update of Skilz, we support 30 plus agents.

Trade-offs of bridged support:

- **Pro:** Get skill benefits even without native support
- **Pro:** Bridge improves as AGENTS.md adoption grows
- **Con:** Slightly less seamless than native support

- **Con:** Agent may not auto-discover skills; may need explicit reference

Tier 3: Universal Mode (any AGENTS.md-compatible agent)

Any agent that reads AGENTS.md files can use skills through universal mode:

```
skilz install my-skill --agent universal --project
```

How this works:

1. Installs skill to `.skilz/skills/` in your project
2. Updates AGENTS.md to reference the skill
3. Agent reads AGENTS.md and sees skill instructions
4. Agent can follow skill guidance even without native skill support

When to use universal mode:

- Working with newer agents that aren't yet widely known
- Using custom or proprietary AI assistants
- Want maximum compatibility at the cost of some features

Choosing the right tier:

```
Do you need skills for Claude, Gemini, or another Tier 1 agent?
├─ Yes → Use standard installation, everything works perfectly
└─ No → Is your agent in Tier 2?
    ├─ Yes → Use standard installation, bridge activates automatically
    └─ No → Use --agent universal --project for AGENTS.md compatibility
```

Cross-Agent Workflows

One powerful Skilz feature is sharing skills between agents using local path installation:

```
# Copy a skill from Claude to Gemini
skilz install -f ~/.claude/skills/my-custom-skill --project --agent gemini
```

Real-world scenario:

You've developed an excellent `api-security-review` skill while using Claude Code. Your team lead uses Gemini CLI and wants the same skill. Instead of manually copying files and figuring out Gemini's directory structure, a single command provides it.

Why this matters:

- Skills can evolve in one agent and propagate to others
- Teams don't need to maintain separate skill repositories per agent
- Expertise flows freely across tool boundaries

Understanding Installation Modes

Skilz offers two installation modes with different trade-offs:

Copy Mode (default):

```
skilz install anthropics/skills/pdf-reader
# Copies skill files directly to ~/.claude/skills/pdf-reader/
```

How it works:

- Downloads skill from source
- Copies all files to the target agent's skill directory
- Each agent gets its own independent copy

When to use copy mode:

- Working with sandboxed or containerized agents

- Want skills to remain stable even if source changes
- Agents don't have universal file system access

Trade-offs:

- **Pro:** Works with all agents, including sandboxed environments
- **Pro:** Changes to source don't unexpectedly affect installed skills
- **Con:** Uses more disk space (one copy per agent)
- **Con:** Updates require explicit `skilz update` command

Symlink Mode:

```
skilz install anthropics/skills/pdf-reader --symlink
# Creates: ~/.claude/skills/pdf-reader -> ~/.skilz/skills/pdf-reader
```

How it works:

- Downloads skill to `~/.skilz/skills/` (universal location)
- Creates symbolic links from each agent's directory to the universal location
- All agents share the same skill files

When to use symlink mode:

- Managing many skills across multiple agents
- Want disk space efficiency
- Need changes to propagate automatically

Trade-offs:

- **Pro:** Saves disk space (single copy, multiple links)
- **Pro:** Edit skill once, all agents see changes immediately
- **Pro:** Easier to manage skill updates centrally

- **Con:** Doesn't work with sandboxed tools that can't access files outside their workspace
- **Con:** Breaking symlinks can cause confusion

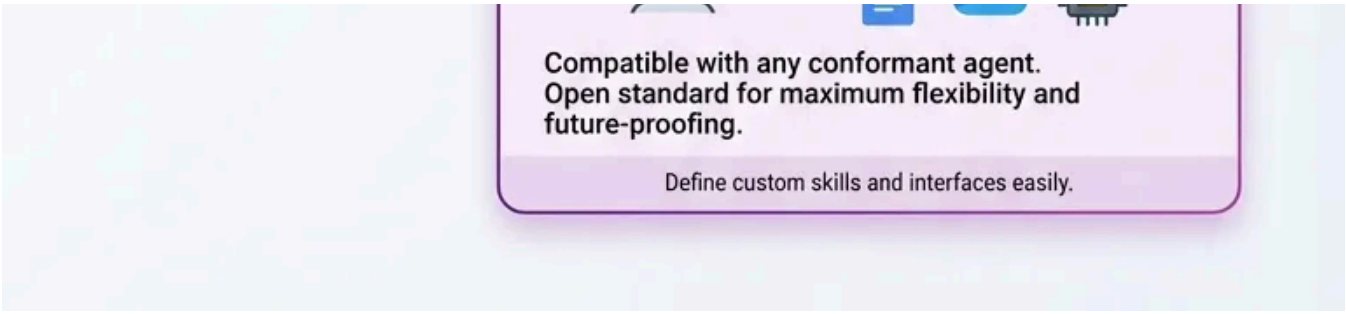
Choosing the right mode:

```
Is your agent workspace-sandboxed?
├─ Yes → Use copy mode (symlinks won't work)
└─ No → Do you manage skills across multiple agents?
    ├─ Yes → Use symlink mode (easier maintenance)
    └─ No → Use copy mode (simpler, more predictable)
```

Example decision tree:

- **Claude Code on macOS:** Not sandboxed; symlink mode works great
- **VS Code Copilot in Docker:** Sandboxed; must use copy mode
- **Gemini CLI with 20+ skills:** Symlink mode saves significant disk space
- **Single agent, few skills:** Copy mode is simpler


Skilz CLI: The Universal Package Manager for AI Tools
SKILZ CLI
pip install skilz - Universal Skill Installer
NATIVE SUPPORT
Claude
Gemini
Codex
Cursor
Goose
Direct integration for 15+ tools & platforms. Seamless skill deployment.
Supported environments: Python, Node.js, Rust, Go, and more.
BRIDGED SUPPORT
Aider
Zed AI
AGENTS.md
AI Agent
Leverage AGENTS.md bridge for extended compatibility. Enables structured communication between Skilz and specialized agents.
Bridging protocols for flexible integration.
UNIVERSAL MODE



Skilz CLI Architecture: Universal installer supporting three tiers — Tier 1 with native support for Claude Code, Gemini CLI, OpenAI Codex, Cursor, and Goose; Tier 2 with bridged support via AGENTS.md for Aider, Zed AI, and Plandex; and Tier 3 universal mode for any AGENTS.md compatible agent

Getting Started

Ready to start using Agent Skills? Here's the practical path from zero to productive.

1. Explore Example Skills

Before creating your own skills, see what's already available:

GitHub Skill Repository:

- **URL:** https://github.com/anthropics/skills
- **What you'll find:** Official examples from Anthropic covering common use cases
- **Categories:** PDF processing, web scraping, data analysis, code generation, documentation

Why start here: See how experienced developers structure skills, write instructions, and organize resources. These examples follow best practices and demonstrate the range of what's possible.

SkillzWave Marketplace:

- **URL:** https://skillzwave.ai/
- **What you'll find:** Community-contributed skills across domains
- **Categories:** Development, data science, legal, finance, operations, creative
- **Features:** Ratings, download counts, version history

Why explore here: Discover specialized skills for your industry. If someone has already solved your problem, use their skill instead of reinventing the wheel.

2. Install Your First Skill

Let's install a practical skill that demonstrates immediate value:

```
# Install Skilz CLI
pip install skilz

# Install a PDF processing skill
skilz install anthropics/skills/pdf-reader
```

What this skill does: Extracts text from PDFs, handles tables, processes forms, and merges documents. Try it immediately:

```
You (to Claude/Gemini/etc.): "Extract the tables from quarterly_report.pdf"

Agent: *activates pdf-reader skill, follows extraction instructions,
outputs structured data*
```

Why this skill is a good first choice:

- Immediately useful for common tasks
- Demonstrates how agents activate skills based on user requests
- Shows the value of encoded expertise (the skill knows which library to use and how to handle edge cases)

3. Create Your Own Skill

Once you've seen how skills work, create one for your own domain. Start simple:

```
# Create a skill directory
mkdir -p my-first-skill
cd my-first-skill

# Create the SKILL.md file
cat > SKILL.md << 'EOF'
---
name: my-first-skill
description: Describe what this skill does and when to use it
---
```

```
# My First Skill

## When to use this skill
Explain the situations where an agent should activate this skill.
Be specific about trigger phrases and use cases.

## Instructions

### Step 1: [First major task]
Detailed guidance on how to accomplish this step.
Include examples, decision criteria, and error handling.

### Step 2: [Second major task]
Continue with clear, actionable steps.

## Examples

Show concrete examples of using this skill:
- Input: [What the user might ask]
- Process: [How the agent should respond]
- Output: [What gets produced]
EOF
```

Both Codex and Claude ship with a skill creator skill. Just describe your skill and they will help you create it.

Understanding the template:

1. **YAML frontmatter:** Name and description for discovery

- Keep description under 30 words
- Use specific keywords the agent can match

2. When to use this skill: Helps the agent decide if this skill applies

- List trigger phrases: “when the user asks to…”, “when working with…”
- Be specific: “PDF files” not “documents”

3. Instructions: The step-by-step guidance

- Write for an intelligent but inexperienced assistant
- Include decision trees for choosing between approaches

- Anticipate common mistakes and provide guidance

4. Examples: Concrete demonstrations

- Show input/output pairs
- Demonstrate edge cases
- Clarify ambiguous situations

Real-world first skill example:

Let's create a skill for code review that checks for common issues:

```
---
name: code-review-basics
description: Review code for common issues including error handling, logging, a
---

# Code Review Basics

## When to use this skill
Use when the user asks to review code, check pull requests,
or evaluate code quality. Applies to any programming language.

## Review checklist

### 1. Error Handling
- [ ] All external calls wrapped in try-catch or equivalent
- [ ] Errors logged with context (not just thrown)
- [ ] User-facing errors have helpful messages

### 2. Logging
- [ ] Important operations logged at INFO level
- [ ] Errors logged at ERROR level with stack traces
- [ ] No sensitive data (passwords, keys) in logs

### 3. Test Coverage
- [ ] New code has unit tests
- [ ] Happy path and error cases covered
- [ ] Tests are independent (no shared state)

### 4. Code Clarity
- [ ] Functions under 50 lines
- [ ] Variable names are descriptive
- [ ] Complex logic has comments explaining why

## Output format
```

```
Create a review comment with:
- ✅ Strengths: What the code does well
- ⚠️ Issues: Problems that must be fixed
- 💡 Suggestions: Optional improvements
```

Why is this skill effective?

- **Clear criteria:** Specific items to check, not vague “code quality”
- **Actionable:** Each item can be verified or fixed
- **Language-agnostic:** Works for Python, Java, JavaScript, etc.
- **Consistent output:** Team gets standardized reviews

Install your skill:

```
# Install to your current agent
skilz install -f ./my-first-skill

# Or install for all team members (project-level)
skilz install -f ./my-first-skill --project --agent claude
```

4. Learn More

Official specification:

- **URL:** https://agentskills.io/specification
- **What you’ll find:** Complete technical specification, file format details, best practices

Integration guide:

- **URL:** https://agentskills.io/integrate-skills
- **For:** Tool builders integrating skill support into their agents
- **Covers:** Discovery protocols, activation triggers, context management

Reference library:

- **URL:** https://github.com/agentskills/agentskills
- **What you'll find:** Reference implementations, validation tools, community examples

The Future of AI Workflows

Agent Skills represent a fundamental shift in how we think about AI agent capabilities. Instead of building workflows into specific tools where they become trapped, we can now package expertise as portable, version-controlled, shareable skills that outlast any individual platform.

Why This Matters Long-Term

The knowledge portability problem is solved.

Your investment in teaching an AI assistant is no longer tied to a specific tool. When Claude Code releases a major update, your skills work. When a new competitor emerges with better models, your skills work. When your company standardizes on a different platform, your skills work.

For the first time, the expertise you encode for AI agents is a durable asset, not a temporary configuration.

Organizational knowledge becomes tangible.

The senior engineer's debugging methodology, the legal team's contract review framework, the data science team's analysis approach: these traditionally lived only in people's heads or scattered documentation. Skills make this expertise tangible, improvable, and scalable.

When a team member leaves, their expertise doesn't walk out the door. When a new hire joins, they have access to accumulated wisdom from day one.

Standards create network effects.

The convergence under the Agentic AI Foundation means these standards will continue to evolve with input from all major AI providers. As more platforms adopt skills, as more developers create and share them, and as more industries discover their value, the ecosystem strengthens.

The marketplace of skills will grow, best practices will emerge, and the quality of available expertise will compound over time.

The Convergence of Standards

Agent Skills work alongside Model Context Protocol (for data access) and AGENTS.md (for project context). Together, these standards create a complete picture:

- **MCP** tells agents what they can access
- **Agent Skills** tell agents how to work
- **AGENTS.md** tells agents about specific projects

An agent equipped with all three can access your company's data, follow your company's methodologies, and understand your company's projects. This is the foundation for AI agents that truly understand how your organization works.

What's Next

The write-once, deploy-everywhere promise of software development is finally coming to AI agents. Whether you're using Claude, Gemini, GPT, or the next breakthrough model, your skills will come along for the ride.

Start small: install a skill, try it out, see how it feels. Then create a skill encoding something you do repeatedly. Share it with your team. Watch as your expertise scales beyond your individual contributions.

The future of AI workflows is portable, shareable, and standards-based. Agent Skills are how we get there.

Key Takeaways

- **Agent Skills are portable**: Write once, deploy across 25+ AI agents
- **Progressive disclosure is efficient**: Load only what's needed, when it's needed
- **Skills encode expertise**: Package any domain knowledge, not just code
- **Standards converge**: MCP for data, Skills for workflows, AGENTS.md for context

- **Neutral governance protects investment**: The Linux Foundation ensures long-term stability
- **Skilz simplifies installation**: One CLI for all agents, consistent experience

Next Steps

1. **Try it now:** `pip install skilz && skilz install anthropics/skills/pdf-reader`
2. **Browse examples:** Visit https://github.com/anthropics/skills
3. **Create your first skill:** Package one workflow you use regularly
4. **Share with your team:** Install skills at project level for consistency
5. **Contribute back:** Publish your best skills to SkillzWave Marketplace

Resources:

- Agent Skills Specification
- Example Skills Repository
- Skilz CLI on PyPI
- Agentic AI Foundation
- Agent Skills: Beyond Coding
- SkillzWave Marketplace

About the Author

Rick Hightower is a technology executive and data engineer who led ML/AI development at a Fortune 100 financial services company. He created skilz, the universal agent skill installer, supporting 30+ coding agents including Claude Code, Gemini, Copilot, and Cursor, and co-founded the world's largest agentic skill marketplace. CConnect with Rick Hightower on LinkedIn or Medium.

The Claude Code community has developed powerful extensions that enhance its capabilities. Here are some valuable resources from Spillwave Solutions (Spillwave Solutions Home Page):

Integration Skills

- Notion Uploader/Downloader Agent Skill: Seamlessly upload and download Markdown content and images to Notion for documentation workflows
- Confluence Agent Skill: Upload and download Markdown content and images to Confluence for enterprise documentation
- JIRA Integration *Agent Skill*: Create and read JIRA tickets, including handling special required fields

Agent Skills | Claude Agent Skill | Openai Codex | Claude Code | Agentic Ai

Follow

Written by Rick Hightower

1.1K followers · 35 following

From building systems at Apple to pioneering AI at Capital One and the NFL: I've spent 20 years making enterprise software truly intelligent. I architect AI.

No responses yet

Write a response

What are your thoughts?