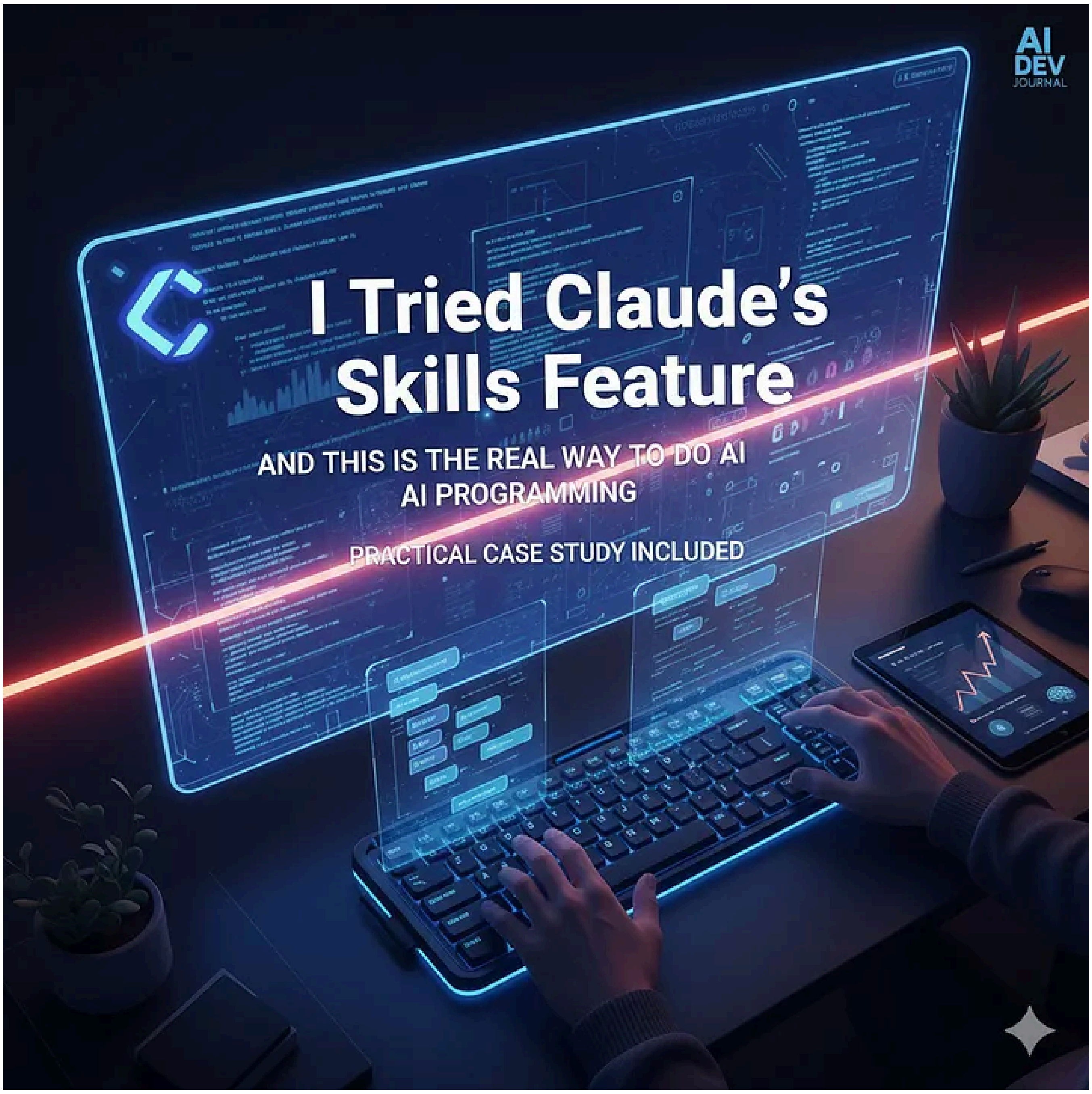
I Tried Claude's Skills Feature, and This Is the Real Way to Do AI Programming | Practical Case Study Included

9 min read · 1 day ago

Nick Follow


AI DEV JOURNAL
I Tried Claude's Skills Feature
AND THIS IS THE REAL WAY TO DO AI AI PROGRAMMING
PRACTICAL CASE STUDY INCLUDED

When the Skills feature just launched for Claude Code, many people thought it was just a tool for convenient, repeatable instructions.

It's not.

The essence of Skills is to equip Claude with a "toolkit" — allowing it to do things it couldn't do before.

What are Skills?

Let's start with an example.

You ask Claude to process a PDF form: extract fields, fill in data, and generate a new PDF. Claude can read the PDF's content, but it can't natively do things like fill the form or manipulate the PDF's structure.

After installing a PDF Skill, Claude can:

- Call the `pypdf` library to fill forms
- Merge and split PDFs
- Handle encrypted documents
- Preserve original formatting

This isn't "remembering instructions"; it's *gaining new abilities*.

Skills are essentially instructions, scripts, and resources packaged into a folder, which Claude automatically loads and uses when needed.

A Skill includes:

- `SKILL.MD` (Required): Tells Claude what this Skill does and how to use it.
- `scripts/` (Optional): Python/JS scripts that perform the actual operations.
- `resources/` (Optional): Template files, reference documents, datasets.

What's the difference with CLAUDE.MD?

Many people will ask: "Isn't this just `CLAUDE.MD`?"

Completely different.

`CLAUDE.MD`:

- Loaded when the project starts, occupies the context window continuously.
- Tells Claude "how this project works."
- Suitable for architecture descriptions, coding standards, environment configs.

Skills:

- Dynamically loaded on-demand, released after use.
- Tells Claude "how to complete a specific task."

- Can contain executable code, extending Claude’s original capabilities.

Let’s take a practical scenario: Your project’s `CLAUDE.MD` says: "We use React + TypeScript, with Jest for testing."

But you create an “Excel Report Generation” Skill, which contains:

- `SKILL.MD` explaining how to generate financial reports.
- `scripts/generate_excel.py` calling `openpyxl` to create complex formulas.
- `templates/report_template.xlsx` as the template file.

When Claude encounters a “generate Excel report” task, it will automatically load this Skill and run the script, rather than generating the code token by token.

Key difference: Skills can execute code; `CLAUDE.MD` is just context.

Why do we need Skills?

The core problem Skills solve is: **Giving a general-purpose AI specialized abilities without retraining the model.**

1. Extend capabilities Claude doesn’t have

Claude’s native abilities are limited:

- It can’t directly manipulate Excel formulas.
- It can’t fill PDF forms.
- It can’t generate complex PowerPoint animations.
- It can’t call specific library APIs.

The official document Skills (Excel/PPT/Word/PDF) work by using scripts to call libraries like `openpyxl`, `python-pptx`, etc., to achieve precise operations Claude couldn't do on its own. https://github.com/anthropics/skills/tree/main/document-skills

2. Code execution is more efficient than token generation

For example, you need to sort 100 rows of data.

- **Without Skills:** Claude generates sorting code (consuming tokens), which might have bugs, and has to be regenerated every time.
- **With Skills:** `scripts/sort.py` executes directly. It's fast, accurate, and saves money.

Anthropic's official stance emphasizes: For deterministic tasks (sorting, data processing, format conversion), script execution is far more efficient than having the model generate code.

3. Encapsulate team knowledge

Unlike the "project-level instructions" of `CLAUDE.MD`, Skills are "task-level tools."

You can create:

- A **Data Analysis Skill:** Contains data cleaning scripts, visualization templates.
- An **API Documentation Skill:** Automatically generates OpenAPI docs in your company's format.
- A **Code Review Skill:** Runs custom linter rules.

These Skills can be reused across projects and shared with the team via plugins.

How to create Skills?

Method 1: Let Claude create it for you

Claude has a built-in "skill-creator" tool. Just say:

> "Help me create a Skill for Git commit conventions."

Method 2: Create it manually

A standard Skill structure:

```
file-renamer/
├── SKILL.MD          (Required)
├── scripts/
│   └── rename.py     (Optional)
```

```
    └── templates/
        └── rules.json    (Optional)
```

`SKILL.MD` Format:

```
---
name: Batch File Renamer
description: Batch renames files according to rules. Use when you need to renam
---

# Batch File Renamer

## Functionality

Batch renames files based on user-specified rules:
- Add prefix/suffix
- Numeric sequencing
- Replace specific characters
- Rename by date

## Usage

1. User specifies the target directory and rules.
2. Run `python scripts/rename.py` to see a preview.
3. Confirm to execute the rename.

## Script Details

`scripts/rename.py` parameters:
- `--dir`: Target directory
- `--pattern`: Renaming pattern
- `--preview`: Only preview, do not execute

## Examples

Add date prefix:
`python scripts/rename.py --dir ./photos --pattern "YYYYMMDD_{name}"`

Batch numbering:
`python scripts/rename.py --dir ./docs --pattern "{001}_{name}"`
```

Key Points:

- **name:** Keep it clear and concise, under 64 characters.

- **description:** Clearly state “what it does” + “when to use it.” Claude uses this to decide whether to trigger the Skill.
- **Body:** Clear steps + practical examples + script explanations.

A Practical Case Study: Web Scraper Data Processing

Imagine you frequently need to process data captured by a web scraper: cleaning, deduplicating, and exporting to Excel.

Create `.claude/skills/data-processor/SKILL.MD`:

~~~
---
name: Scraper Data Processor
description: Cleans scraper data and generates an Excel report. Use when proces
---

# Scraper Data Processor

## Processing Workflow

1.  Read raw data in JSON/CSV format.
2.  Data cleaning: deduplication, fill missing values, standardize formats.
3.  Statistical analysis: counts, deduplication rate, anomaly detection.
4.  Generate Excel report (including data and charts).

## Script Functions

### scripts/process.py

Main processing script:
```python
import pandas as pd
from openpyxl import Workbook
from openpyxl.chart import BarChart

def clean_data(df):
    # Deduplicate
    df = df.drop_duplicates()
    # Fill missing values
    df = df.fillna('')
    # Standardize formats
    df['price'] = df['price'].astype(float)
    return df

def generate_report(df, output_file):
    # Generate Excel
    with pd.ExcelWriter(output_file, engine='openpyxl') as writer:
        df.to_excel(writer, sheet_name='Data', index=False)
~~~

~~~
        # Add statistics
        stats = df.describe()
        stats.to_excel(writer, sheet_name='Statistics')

        # Add chart
        wb = writer.book
        ws = wb['Statistics']
        chart = BarChart()
        # ...chart configuration
```

### scripts/dedupe.py
Quick deduplication tool:
```Python
import json

def dedupe_json(file_path, key_field):
    with open(file_path) as f:
        data = json.load(f)

    seen = set()
    unique_data = []

    for item in data:
        key = item.get(key_field)
        if key and key not in seen:
            seen.add(key)
            unique_data.append(item)

    return unique_data
```



## Usage Examples
Process JSON data:
python scripts/process.py data.json --output report.xlsx

Quick deduplication:
python scripts/dedupe.py data.json --key url


## Output Description
The generated Excel will include:
Data sheet: The complete cleaned data
Statistics sheet: Counts, deduplication rate, price distribution, etc.
Chart sheet: Visualized statistical results
~~~

After saving, create the corresponding script files.

In the future, when you ask Claude "process this scraper data and generate a report," it will:
1. Automatically identify the need for the "Scraper Data Processor" Skill.
2. Read `SKILL.MD` to understand the process.
3. Run `scripts/process.py` to perform the cleaning.
4. Generate the Excel report.

> *This is much faster than having Claude write the processing code from scratch every time, and the output is consistent.*

Three Types of Skills

1. Personal Skills

- **Location:** `~/.claude/skills/`
- **Scope:** All projects
- **Suitable for:** General-purpose tools (file processing, data conversion, formatting, etc.)

2. Project Skills

- **Location:** Project root dir `.claude/skills/`
- **Scope:** Current project
- **Suitable for:** Project-specific tools (deployment scripts, test data generation, project doc generation)
- Can be committed to Git and shared with the team.

3. Plugin Skills

- **Installation:** Via Claude Code plugins
- For example, the official document processing Skills:

```
/plugin marketplace add anthropics/skills  /plugin install document-skills@anth
```

- After installation, you gain advanced operational capabilities for PDF/Excel/PPT/Word.
- The `anthropics/skills` repository also provides example Skills for creative design, web testing, MCP server generation, etc.

How to make Skills work?

Skills are triggered automatically by Claude; you don't need to specify them manually.

Trigger Mechanism

1. On startup, Claude scans the `name` and `description` of all Skills (only takes ~100 tokens/Skill).
2. You initiate a task.
3. Claude determines which Skill is needed.
4. It reads the detailed `SKILL.MD` via a bash command to load the full content.
5. If necessary, it executes the scripts in the `scripts/` directory.
6. After completing the task, the Skill is released.

The entire process is automatic. You only see the final result.

What if a Skill doesn't activate?

Problem 1: The `description` isn't specific enough.

- ❌ **Bad:** `description: Processes data files`
- ✅ **Good:** `description: Cleans scraper JSON/CSV data and generates an Excel report. Use when processing scraper data, needing deduplication/cleaning, or generating statistical reports. Includes handling for common fields like URL/price/title.`
- **Key:** Clearly state "what it does" + "when to use it" + "what data type it handles."

Problem 2: Mention it explicitly.

- If Claude doesn't trigger it automatically, just say:

```
Use the 'Scraper Data Processor' Skill to process this file.
```

Problem 3: Check if it's loaded.

```
What Skills do you have right now?
```

Claude will list the recognized Skills.

Problem 4: Code execution is not enabled.

- **Claude Code:** Enabled by default.
- **claude.ai:** Settings > Features > Enable "Code Execution and File Creation"

Best Practices

1. Scripts First

If you can handle it with a script, don't make Claude generate code.

Deterministic tasks (sorting, format conversion, data cleaning): Write Python/JS scripts.

Creative tasks (generating copy, designing code architecture): Let Claude do its thing.

2. Stay Focused

One Skill, one job.

Don't stuff "Scraping + Cleaning + Analysis + Reporting + Emailing" into one Skill.

Break them up:

- web-scraper/ : Scraping logic
- data-cleaner/ : Data cleaning
- report-generator/ : Report generation

- email-sender/: Email sending

Claude will automatically combine multiple Skills when needed.

3. Include plenty of practical examples

Put real, runnable examples in `SKILL.MD`.

4. Attach reference documents

If it involves specific libraries, put key documentation inside:

```
references/
├── openpyxl-api.md
└── pandas-cheatsheet.md
```

Reference them in `SKILL.MD`: For detailed API, see `references/openpyxl-api.md`

5. Version Management

Skills will evolve. Note the version:

```
version: 1.2.0
```

```
## Changelog

### v1.2.0 (2025-10-20)
- Added support for Excel formulas
- Optimized processing speed for large files

### v1.1.0 (2025-10-15)
- Added chart generation feature
```

Answering a Common Concern: Will this blow up my context?

Some people might worry about Skills filling up the context window.

It won't.

Skills use "Progressive Disclosure": On startup, only metadata is loaded (~100 tokens). Only when a Skill is needed is its `SKILL.MD` read (within 5k tokens). Scripts are executed directly and don't occupy context.

If you install 20 Skills, it only takes ~2000 tokens at startup. Only when a specific Skill is used is its full content loaded, and it's released after use.

Claude Code's context window is 200k tokens. The progressive loading of Skills means you can install dozens of them without worry.

You can use the `/context` command to check context usage at any time.

Security Reminder

Skills can execute code, so be careful:

- ✅ Only use Skills you created yourself or official ones.
- ✅ Review the code before installing from GitHub/plugins.
- ❌ Don't use Skills from unknown sources.
- ❌ Don't put API keys in your Skills.

A malicious Skill could read your files, execute dangerous commands, or leak information. For Skills downloaded from the internet, you *must* first inspect all code in `SKILL.MD` and `scripts/`.

Ready-made Skills from the Plugin Market

Official Skills

```
# Document Processing (PDF/Excel/PPT/Word)
/plugin install document-skills@anthropic-agent-skills

# Example Skills (Creative Design, Web Testing, etc.)
/plugin install example-skills@anthropic-agent-skills
```

Share Your Skills

Developed a useful Skill? Package it as a plugin:

- Create a plugin repository, put Skills in the `skills/` directory.
- Add a `marketplace.json`.
- Publish to GitHub.
- Others can install it with `/plugin install your-skills@your-repo`.

Refer to the `anthropics/skills` repository structure.

Practical Application Scenarios

Frontend Development

- **Component Scaffolding Skill:** Generate standardized React/Vue components.
- **Icon Processing Skill:** Optimize SVGs, batch export different sizes.
- **Bundle Analysis Skill:** Run `webpack-bundle-analyzer` to generate a report.

Data Analysis

- **Data Cleaning Skill:** Standardize fields, deduplicate, fill missing values.
- **Visualization Skill:** Generate charts in a specific style (company colors, fonts).
- **Statistical Report Skill:** Automatically generate weekly/monthly reports.

DevOps

- **Log Analysis Skill:** Parse nginx/application logs, extract key metrics.
- **Deployment Check Skill:** Verify environment configs, dependency versions.
- **Performance Monitoring Skill:** Generate performance reports, flag anomalies.

Content Creation

- **Image Batch Processing Skill:** Compress, convert formats, add watermarks.
- **Video Editing Skill:** Call `ffmpeg` to generate previews, extract clips.
- **Document Conversion Skill:** Convert Markdown to PDF/Word, preserving format.

Conclusion

The core value of Skills is not to replace `CLAUDE.MD`, but to give Claude a "toolbox."

Three key points:

- **Extend Abilities:** Let Claude do things it couldn't do natively (precise Excel operations, complex script execution, calling specific libraries).
- **Improve Efficiency:** Use script execution for deterministic tasks — it's faster and more accurate than token generation.
- **Knowledge Reuse:** Encapsulate knowledge into modules, use them across projects, and share them via plugins.

Skills ≠ A convenient prompt management tool

Skills = A composable AI capability expansion pack

I recommend starting simple: file batch processing, data cleaning, format conversion. Once you experience the efficiency of "script execution," you can gradually build out your complete toolkit.

Claude Code Claude Skills Claude Ai AI Ai Tools

Follow

Written by Nick

0 followers · 1 following

No responses yet

Bgerby

What are your thoughts?

More from Nick

Nick

Forget the AI Super App. The Revolution Will Be Millions of “Super Useful” Apps.

Sep 27

Nick

How to Use Codex? A Guide from Beginner to Hardcore Configuration (with Tutorial)

Recently, some friends have been asking about the Codex programming tool. I tested it out.

1d ago

Nick

Has Your Social Feed Been Flooded by "Nano Banana" recently?

Honestly, when I first started playing with it, I just followed the crowd — changing a background, photoshopping a piece of clothing — and…

Sep 27

Nick

[n8n] Drowning in Monday Emails? This Open-Source Tool Can Automate Your Entire Week.

A deep dive into n8n, a comparison with Coze and Dify, and a step-by-step guide to building your first intelligent workflow.

Sep 21

See all from Nick

Recommended from Medium

In Coding Nexus by Code Coup

The Claude Skills Cookbook: Anthropic's New Context Engine Outperforms MCP??

Anthropic quietly released the Claude Skills Cookbook on GitHub. Initially, I assumed it was just another dull API document. But after...

3d ago 45

Ondřej Popelka

Generating user documentation from nothing

AI vision and reasoning to transform screen recordings into structured user guides

Oct 19 390 6

Will Lockett

You Have No Idea How Screwed OpenAI Actually Is

When you find yourself in a hole, at what point do you stop digging?

4d ago 4.96K 148

In ITNEXT by Mario Bittencourt

Up your AI Development Game with Spec-Driven Development

Spec Driven Development is new promising way to adopt AI and keep the developer in the loop. I will cover all aspects of SpecKit here.

Oct 20 365 2

In AI Software Engineer by Joe Njenga

Kimi CLI is Here — Another Hot Claude Code Alternative (Does it Level Up?)

Kimi CLI has just been released and it’s going to take center stage among AI terminal coding tools.

2d ago 27 2

In DevOps.dev by Yash Batra

10 CLI Tools That 10x Your Backend Productivity

I've spent the last few years building backend systems across multiple product-based companies—from scrappy startups to teams...

4d ago 29

See more recommendations