

Create Your Own Codex Skills

A simple guide for turning repeated prompts into reusable Codex workflows

Who this guide is for

This guide is written for non-technical users. You do not need to know programming. The main idea is simple: if you keep giving Codex the same long instructions, turn those instructions into a skill.

Table of contents

1. What a Codex skill is
2. When to create one
3. The parts of a skill
4. A simple creation process
5. Best practices
6. Example from this session
7. How to use skills
8. Quick checklist

1. What A Codex Skill Is

A Codex skill is a small folder of instructions that teaches Codex how to do a repeated task. It is like a saved expert workflow. Instead of pasting a long prompt every time, you ask Codex to use the skill.

Plain-language definition

A skill is a reusable instruction pack. You can trigger it manually with a name like \$pay-period-summary, or Codex can use it automatically when your request clearly matches the skill description.

Skills are useful because they keep important rules in one place:

- What information Codex should ask for first.
- What files, tools, or sources Codex should inspect.
- How the final answer should be written.
- What mistakes Codex should avoid.
- What scripts or templates can make the work more reliable.

2. When To Create One

Create a skill when a task is repeated, important, or easy to get wrong.

Good reason	Example
You reuse a long prompt	A timesheet prompt used every other week.
The task has rules	Always ask for date range, repo, audience, and output type.
The task uses evidence	Read commits, sessions, notes, and Jira before writing.
The task needs a consistent style	Use simple academic language for PI-facing reports.
The task has fragile steps	Run a helper script so evidence collection is less manual.

3. The Parts Of A Skill

A skill is a folder. The folder usually contains these pieces:

- SKILL.md: the main instruction file. This is required.
- references/: longer notes, examples, style rules, schemas, or old prompts.
- scripts/: small helper programs for repeatable or fragile steps.

- `assets/`: templates, images, starter files, or other reusable materials.
- `agents/openai.yaml`: optional display information for the Codex interface.

Keep the main skill short

Put only the core workflow in `SKILL.md`. Move long examples, old prompts, and detailed rules into reference files. This keeps Codex focused and saves context.

4. A Simple Creation Process

1. Start with real examples

Describe what you normally ask Codex to do. Include one or two examples of successful output.

2. Ask the important questions

Decide what Codex should ask first, what it should infer, and what it should never assume.

3. Choose a clear name

Use lowercase words and hyphens, such as pay-period-summary.

4. Initialize the skill

Use the skill-creator helper so the folder has the expected structure.

5. Write the main workflow

Keep SKILL.md short, direct, and action-oriented.

6. Add references and scripts

Use references for long guidance and scripts for reliable repeated work.

7. Validate the skill

Run the validation tool to catch missing metadata or naming problems.

8. Forward-test it

Ask a fresh agent or fresh session to use the skill on a realistic task. Fix what breaks.

5. Best Practices

Use skills for guidance, not clutter

- Do not include extra README files unless they are truly needed.
- Do not paste every possible detail into SKILL.md.
- Write instructions that another Codex session can follow without knowing your conversation history.
- Use references for longer details.

Choose the right level of control

Some tasks need freedom; some need guardrails.

- High freedom: writing, planning, brainstorming, or judgment-heavy work.
- Medium freedom: repeated workflows where Codex can decide some details.
- Low freedom: fragile steps where a script or exact command sequence prevents mistakes.

Make the trigger description clear

Codex uses the skill name and description to decide when to load a skill automatically. The description should say what the skill does and exactly when it should be used.

6. Example From This Session

In this conversation, a repeated timesheet prompt became a Codex skill named `pay-period-summary`. The goal was to stop pasting the same long prompt every other week.

The important design choices were:

- The skill lives in Codex, under the Codex skills folder.
- It asks for the date range and repo paths first.
- It can use git history, AI sessions, manual notes, and approved Jira evidence.
- It treats non-code work as valid when it is project-related.
- It writes plain PI-facing summaries by default.
- It preserves the original long prompt as a reference file.
- It includes read-only helper scripts for evidence collection.
- It was validated and forward-tested. The forward-test caught a parser bug, which was fixed before final use.

7. How To Use Skills

Manual trigger

Manual triggering means you name the skill yourself. This is the clearest method. Example: Use `$pay-period-summary` to create my timesheet report for April 4 to April 17.

Use manual triggering when the task is important, private, or easy to misunderstand.

Automatic trigger

Automatic triggering means Codex decides the skill is relevant because your request matches the skill description. This works best when the skill description is specific.

Practical tip

If Codex does not pick the skill automatically, mention it by name. If Codex picks the wrong skill, make the skill description more specific.

Scheduled use and cron-style jobs

A scheduled job runs later or repeats on a schedule. In Codex, you can ask for an automation such as Every other Friday, use `$pay-period-summary` to prepare my timesheet draft.

For scheduled jobs, include enough information so the job can run without guessing:

- Which skill to use.
- The project or repo path.
- The reporting cadence.
- The default output location.
- Whether external sources like Jira are allowed.
- Whether the job should draft only or also write files.

Hooks

A hook is a small action that runs when something happens, such as a session ending or a file being saved. For non-technical workflows, the safest hook usually collects raw notes; the skill later turns those notes into polished output.

- Good hook use: save a short daily work log when a session ends.
- Good hook use: record the repo name, date, and summary of the session.
- Risky hook use: automatically edit important reports without review.

- Best practice: hooks collect evidence; skills interpret and write the final report.

8. Quick Checklist

- Do I repeat this task often?
- Can I describe clear examples of good output?
- Does Codex need to ask questions before starting?
- Are there sources Codex should always check?
- Are there sources Codex should never use without approval?
- Should long instructions move into references?
- Would a small script make any step safer?
- Did I validate the skill?
- Did I test the skill on a realistic request?

Bottom line

A good skill is not a bigger prompt. It is a small reusable workflow: clear trigger, clear steps, clear defaults, and enough tests to trust it.