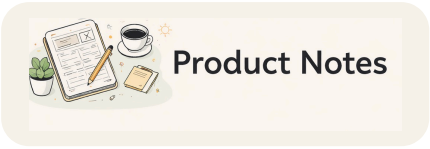

PM Skills

The Practical Guide — how to actually use 207 AI skills, with worked examples

207 skills | 21 professions | eval-scored 4.8/5 | MIT

Repo: github.com/mohitagw15856/pm-claude-skills
Run free (no install): mohitagw15856.github.io/pm-claude-skills

PM stands for Professional, not just Product Management.

What's inside

1 · What this is — and the one idea behind it

Ask any AI for a PRD, an exec update, or a launch plan and you get *plausible filler* — something 80% there and 100% generic that you still rewrite from scratch. The model can write fluently; it just doesn't know what **"good"** looks like for a specific professional deliverable.

PM Skills fixes that. Each "skill" is a structured `SKILL.md` file that encodes the real method a senior professional uses for one job — the sections, the rubric, the judgement, the anti-patterns. When the AI loads a skill, its first draft already has the right structure, the right questions asked, and the common mistakes avoided.

> **The one idea:** a skill is not a clever prompt — it's a *repeatable recipe for a good deliverable*. Same skill, two different users, two outputs that look like the same professional product.

There are **207 skills across 21 professions** — product, engineering, design, data, marketing, CS, legal, finance, HR, sales, ops, research, and more. They're open-source (MIT), eval-scored on a public rubric, and run anywhere a capable model reads instructions.

What you can make with them (a taste)

Give it…	Get back…	Skill
messy progress notes	a tight 250-word CEO briefing	`executive-update`
a vague feature idea	a structured PRD with metrics & risks	`prd-template`
a backlog of ideas	a ranked, defensible priority list	`rice-prioritisation`
a raw incident timeline	a blameless postmortem with owners	`incident-postmortem`
a meeting transcript	decisions, owners & next steps	`meeting-notes`

2 · When to reach for a skill (and when not to)

Great fit

- **Recurring professional deliverables** — anything you produce more than once and that has a “right shape”: PRDs, updates, postmortems, briefs, plans, reviews.
- **You're staring at a blank page** — the skill removes the structure debate so you spend time on judgement, not formatting.
- **You want consistency across a team** — everyone's PRD comes out in the same shape.

Poor fit (don't force it)

- **One-off, throwaway tasks** with no “right structure.”
- **Pure facts retrieval** — a skill gives *structure*, not ground truth. Feed it your real data (see §11).
- **Final say on judgement calls** — a skill drafts; **you** decide. Treat every output as a strong first draft to verify, not gospel.

> Rule of thumb: if you'd be happy to reuse the same outline next month, there's probably a skill for it — and it'll save you the most time.

3 · Get started in 60 seconds (browser) — and every other way

The fastest way to feel it: the **browser playground**. No install, runs with your own model key (stored only in your browser), or a built-in no-key model.

1. Open **mohitagw15856.github.io/pm-claude-skills**
2. Pick a skill (try *Executive Update*) — or describe your task and let it recommend one.
3. Paste your own Claude / OpenAI / Gemini key (or choose “In-browser, no key”).
4. Fill the short form → **Run**. The result streams in. Toggle **Compare** to see it with vs. without the skill.

Install it where you work

You use…	Command / how
Claude Code	`npx pm-claude-skills add --agent claude`
Cursor / Codex / 60+ agents	`npx skills add mohitagw15856/pm-claude-skills`
MCP (any client, on demand)	`claude mcp add pm-skills -- npx -y pm-claude-skills-mcp`
ChatGPT / Claude.ai / Cursor (hosted)	add connector URL `pm-skills-mcp.pm-claude-skills.workers.dev`
Python (LangChain / CrewAI)	`pip install pm-skills`
ChatGPT / Gemini (copy-paste)	paste an export from `exports/` into a Custom GPT / Gem

Everything is one source of truth: the same skill body powers all of these.

🌐 **Not in English?** Pick an **output language** in the playground and any skill responds in it — Spanish, Mandarin, Hindi, Arabic *(right-to-left)*, Portuguese, French, German, Japanese, Russian, or Indonesian. The frameworks are universal; the model localizes the whole output.

4 · Anatomy of a skill: how to read and invoke one

Every skill is a markdown file with two parts: a tiny **frontmatter** (name + a description with trigger phrases) and a **body** (the method, an output template, quality checks, and anti-patterns).

```
--- frontmatter (all the model sees when deciding to load it) ---
name: prd-template
description: "Create a PRD... Use when asked to write a PRD,
  product spec, or feature specification. Produces a complete
  PRD with problem, user stories, requirements, success metrics."
--- body ---
# PRD Template Skill
## Required Inputs       # what it asks you for
## Template Structure    # the sections it produces
## Quality Checks        # the bar the output must pass
## Anti-Patterns         # mistakes it refuses to make
```

How to invoke one

- **Claude Code (native):** just describe the task — Claude auto-loads the matching skill from its description. Or name it: *"use the prd-template skill for…".* Some ship a slash command: `/prd`, `/rice`, `/sprint-plan`.
- **MCP:** *"search the skills for churn, then apply the best one to this account."*
- **Chat (ChatGPT/Gemini):** paste the skill body as the system prompt / custom instructions, then give your input.

> **Pro move:** a skill always asks for missing inputs rather than inventing them. Give it a *brief* up front (even bullet points) and it'll produce a full draft instead of interviewing you.

5 · Worked example 1 — a PRD from a fuzzy idea

skill `prd-template` command `/prd`

① Your input (a few bullets is enough)

```
Feature: referral program for B2B users.
Goal: +15% signups in Q3. Activated admins invite peers;
both sides get account credit. MVP only. Reviewers: Eng, Growth.
```

② What you ask

```
Use the prd-template skill to write a PRD for this. Ask me
anything critical that's missing; otherwise produce the draft.
```

③ What comes back (shape, abridged)

```
PRD — B2B Referral Program
1. Overview — Problem, goal (+15% Q3 signups), non-goals (MVP: no tiers)
2. Target user — Activated admins on paid plans
3. User stories
   US1: As an admin, I can invite a peer by email and see status…
   US2: As an invitee, I get a credit applied on signup…  (+ acceptance criteria)
4. Requirements — functional + edge cases (self-referral, fraud caps)
5. Success metrics — primary: invited→signup rate; guardrail: support tickets
6. Risks & open questions — abuse, attribution window, finance sign-off
```

Why it's better than a raw prompt: success metrics and a guardrail metric are there by default; abuse and attribution risks are surfaced; non-goals keep MVP honest. You spend your time deciding the attribution window — not remembering that a PRD needs one.

Next: feed this PRD into `rice-prioritisation` to position it, or `go-to-market` to plan the launch (see §8).

6 · Worked example 2 — an exec update from messy notes

skill `executive-update` command `/exec-summary`

① Your input (raw, unstructured)

```
shipped onboarding redesign to all new accounts, activation
looking way up. backend hiring is 2 roles behind, billing
revamp now at risk for Q3. NRR flat. need contractor budget.
audience: CEO. period: Q2.
```

② What comes back (abridged)

Executive Update — Q2, for the CEO
Headline: Activation up sharply after onboarding redesign; on track for Q2, with one staffing risk that needs a decision.
Key metrics: activation ▲ (best quarter); NRR flat at 112%.
Progress: onboarding redesign at 100% of new accounts; faster TTV.
Risk & decision needed: backend hiring 2 roles behind → billing revamp at risk for Q3. *Decision:* approve contractor budget to hold the date.
Next: roll the activation win into core; bring a Q3 staffing plan.

The skill's judgement: it leads with the headline and the *decision needed* (Pyramid Principle), keeps it ~250 words, and separates progress from the ask — exactly what a CEO wants. Add your real numbers and it's shippable.

7 · Worked example 3 — RICE prioritisation (with the helper script)

skill `rice-prioritisation` command `/rice` script computes the maths by rule

Some skills ship a stdlib-only helper so the numbers are computed, not vibed. RICE is one:

```
# initiatives.json: [{name, reach, impact, confidence, effort}, ...]
python3 scripts/rice_calculator.py initiatives.json
# → ranked table with RICE scores + auto-flags:
#   quick-win (high score, low effort), moonshot (high impact/effort),
#   low-confidence (≤50%)
```

Result (abridged)

```
Initiative          Reach  Impact  Conf  Effort  RICE   Flag
Onboarding revamp    5000   2.0     80%   3       2667   quick-win
Referral program     5000   3.0     50%   2       3750   low-conf
Enterprise SSO        800   3.0     100%  5        480   —
→ Sequence: validate referral confidence first; ship onboarding now.
```

How to use it well: let the script rank, then *investigate any surprising top item* before accepting it — RICE is a tool, not a verdict. The skill explicitly tells the model to do this, so it won't hand you a number it can't defend.

Other computed skills: sprint capacity (`sprint-planning`), customer health (`cs-health-scorecard`).

8 · Chaining skills — the workflow recipes

Individual skills are useful; **chaining** them is the superpower. A recipe runs several skills in order and **passes each output forward** as context for the next — a fuzzy idea comes out the other end as a joined-up set of artifacts.

```
/ship-a-feature  "a referral program for B2B users"

  ambiguity-resolver → prd-template → rice-prioritisation
   frame the problem    spec it        position it
        → roadmap-narrative → go-to-market
           place on roadmap    launch plan
        └──── each stage's output feeds the next ────┘
```

The built-in recipes

Recipe	Does
`/ship-a-feature`	idea → PRD → priority → roadmap → launch plan
`/close-the-quarter`	metrics → churn → exec update → board deck
`/launch-a-product`	competitors → positioning → GTM → checklist → press release
`/rescue-an-account`	health score → churn cause → escalation → renewal plan
`/run-discovery`	frame → interview guide → synthesis → prioritise

Two ways to run a chain: the slash command in Claude Code, or visually in the browser **Workflow Canvas** (drag skills into a chain) — and the **Auto-Agent** will even pick the chain for you from a plain-English goal.

> **Carry context forward.** If you're chaining by hand, paste the previous step's output into the next so each skill builds on the last instead of starting cold.

9 · Make it yours — Skill Memory & context

Generic output is the #1 complaint with AI. **Skill Memory** fixes it: tell the skills who you are *once*, and every skill produces output already tuned to your product, audience, and voice.

- **In Claude Code:** run `/setup-context` (or copy `templates/pm-context.example.md` → `pm-context.md`) and fill it in. Skills read it as standing context.
- **In the playground:** fill the 🧠 **Your context** box — saved in your browser, prepended to every run.

```
# pm-context.md (keep it short and concrete)
Company: Acme — collaborative analytics for RevOps teams
Audience I write for: CEO, board, eng team
Voice: crisp, no hype; lead with the decision; updates < 250 words
North-star metric: weekly active workspaces (define it exactly)
Competitors: BigCo (we win on time-to-value)
```

> Without context: “write an exec update” → generic, you rewrite it.
> With context: “write an exec update” → your voice, your metrics, your audience — shippable first try.

10 · The Professional Brain — memory + actions

Skill Memory is one file. The **Professional Brain** is the full version: a durable, local **markdown memory** that skills read before answering and write to after — so runs stop starting cold and decisions keep their *why*. No vector DB; it's grep-able, auditable, Obsidian-compatible.

```
brain/
  context.md   knowledge/   decisions/   hypotheses/
  stakeholders/   entities/   source/
```

Provenance — the trust mechanism

Every fact is tagged by how strong it is, so a hunch never poses as data:

`[data]` · `[interview]` · `[external]` · `[verbal]` · `[hunch]` (strongest → weakest)

The loop (with a real example)

1. **Set it up:** `cp -r templates/brain ./brain` (or run the `/brain` command → `init`).
2. **Recall:** the brain feeds context into a skill automatically. Manually:

   ```
   python3 skills/professional-brain/scripts/brain_query.py ./brain "retention"
   # → matches ranked by provenance, e.g.:
   #   [data]   knowledge/strategy.md: mobile NPS 6.2 vs web 8.1…
   #   [verbal] stakeholders/sarah.md: wants retention tied to a metric…
   ```

3. **Run a skill:** ask for a PRD — it reads `knowledge/strategy.md` and carries the tags through, instead of you re-pasting context.
4. **Record back:** after a meeting, `meeting-notes` proposes decisions to write to `decisions/` — append-only, **dry-run by default, approved before anything is written.**
5. **Act (optional):** the `action-runner` turns a skill's recommendations into real tickets / messages — risk-rated, approved per action, recorded back. *Nothing acts silently.*

Try it with no install: the in-browser Brain at **/brain.html** (round-trips with the on-disk folder via markdown import/export).

11 · Run it anywhere — Claude Code, MCP, the REST API, n8n / Obsidian

On your real data (the big unlock)

A skill gives *structure*; a connector gives it *your facts*. Run the `pm-skills` MCP server alongside a data server (filesystem, GitHub, a database) and a skill acts on what that server exposes:

```
# Claude Code: add both, once
claude mcp add pm-skills  -- npx -y pm-claude-skills-mcp
claude mcp add github     -- npx -y @modelcontextprotocol/server-github

# then just ask:
"Get the prd-template skill, read GitHub issue acme/app#123, and write the PRD from it."
"Run churn-analysis on exports/q2.csv."
"Open a GitHub issue per item in the product-launch-checklist."
```

The REST API (for any HTTP / no-code tool)

```
# read-only, no auth, CORS-open — base:
# https://pm-skills-mcp.pm-claude-skills.workers.dev
GET /v1/skills                # list (?bundle= ?q= ?limit=)
GET /v1/skills/{name}?format=md  # the raw skill body
GET /v1/workflows             # the recipe chains
```

- **n8n** — feed a skill into your AI node; a trigger becomes a finished doc routed to Slack/Notion.
- **Obsidian** — every skill is a vault note you run as a Copilot / Text-Generator prompt.
- **Lovable** — build a skill-powered app; a knowledge snippet makes its generator skill-aware.

Full recipes in `connectors/` (n8n.md, obsidian.md, lovable.md).

12 · Trust the output — eval scores, Compare, Grade

Quality is measured, not claimed. Three ways to check before you rely on a result:

- **Eval scores** — an LLM judge rates each skill against a held-out case on four dimensions (structure · completeness · usefulness · grounding). **196 skills scored, avg 4.8/5** on the public leaderboard. A regression gate blocks any change that lowers a score.
- **Compare mode** (playground) — run the same input *with vs. without* the skill, side by side, to judge whether it actually helped.
- **Grade your draft** — paste an existing PRD/update and get a rubric score, ranked gaps, and a redline graded against the skill's framework. Great for improving work you already have.

Before you ship a plan, run `red-team-review` — it stress-tests the plan against a room of hostile expert personas and surfaces what breaks.

13 · Tips, anti-patterns & getting the most out of it

Do

- **Give a brief, not a blank.** Even three bullets gets you a full draft instead of an interview.
- **Set your context once** (§9) — it lifts every single skill.
- **Chain for real work** (§8). Most deliverables are a sequence, not one shot.
- **Compute, don't guess.** If a skill ships a helper script (RICE, sprint, health), run it.
- **Feed it your data** (§11) rather than pasting summaries — “draft the PRD *from* the ticket.”
- **Verify before you ship.** The skill drafts; you own the judgement. Check surprising numbers.

Don't

- **Don't treat output as fact.** Structure ≠ ground truth.
- **Don't accept a surprising ranking** without checking the estimates behind it.
- **Don't force a skill** onto a one-off with no “right shape” — you'll fight it.
- **Don't auto-run actions you haven't reviewed** — keep the Brain's approval gate on.

If you're rolling it out to a team

1. Install `pm-essentials` + share one `pm-context.md` so everyone's output matches your house style.
2. Pick the 5 deliverables your team writes weekly; standardise on those skills first.
3. Add the GitHub Action or Skill Bot so skills run in CI / from a PR comment.
4. Custom skills (your templates, your terminology) eliminate *rework*, not just the blank page — see the repo's “custom skills for your team.”

14 · Quick reference & where to go next

Five skills to try first

`executive-update` · `prd-template` · `rice-prioritisation` · `competitor-teardown` · `meeting-notes`

Commands worth memorising

`/prd` · `/rice` · `/sprint-plan` · `/retro` · `/exec-summary` · `/brain` · `/setup-context` · the recipes in §8

Where things live

All skills, browsable	SKILLS.md · the live catalog & Skill Galaxy (/galaxy.html)
One-page cheatsheet	CHEATSHEET.md (+ PNG/PDF in docs-assets)
The Brain	BRAIN.md · BRAIN_QUICKSTART.md · /brain.html
Integrations	connectors/ (n8n, Obsidian, Lovable) + the REST API
Quality	the leaderboard · evals/ · Compare & Grade in the playground
Contribute / request	CONTRIBUTING.md · Skill Studio (make a skill → PR)

Start here: open the playground, run *Executive Update* on your own messy notes, and flip on **Compare**. That 30-second moment — generic mush on one side, your shippable draft on the other — is the whole pitch.