ANTHROPIC CLAUDE & WORKSPACE STUDIOS • FIELD GUIDE

Claude Skills

Packaging reusable domain expertise across Claude.ai, Claude Desktop, and Claude Code—mastering 3-tier progressive disclosure, `SKILL.md` anatomy, `skill-creator`, and executable scripts.

3-Tier Progressive Disclosure | `SKILL.md` Open Format | Claude.ai • Desktop • Code

`skill-creator` Meta-Skill | Bundled Python & Bash Scripts | `pptx` • `xlsx` • `docx` • `pdf`



Zero Context Bloat

Preload ~50–100 tokens of YAML metadata per skill so Claude carries 100+ capabilities without slowing down.

Write Once, Run Everywhere

One portable `SKILL.md` folder runs across Claude.ai, Claude Desktop, Cowork, Claude Code, and the Messages API.

Deterministic Script Execution

Offload fragile math, schema linting, and document checks into bundled Python and Bash scripts inside secure sandboxes.

Solving Context Window Bloat Forever

Pasting giant SOP manuals into every system prompt wastes thousands of tokens and dilutes Claude's attention. Anthropic's **3-Tier Progressive Disclosure** lets Claude keep 100+ specialized skills on standby at a ~50-token idle cost—opening full instructions and scripts only when needed.

Quick Navigation

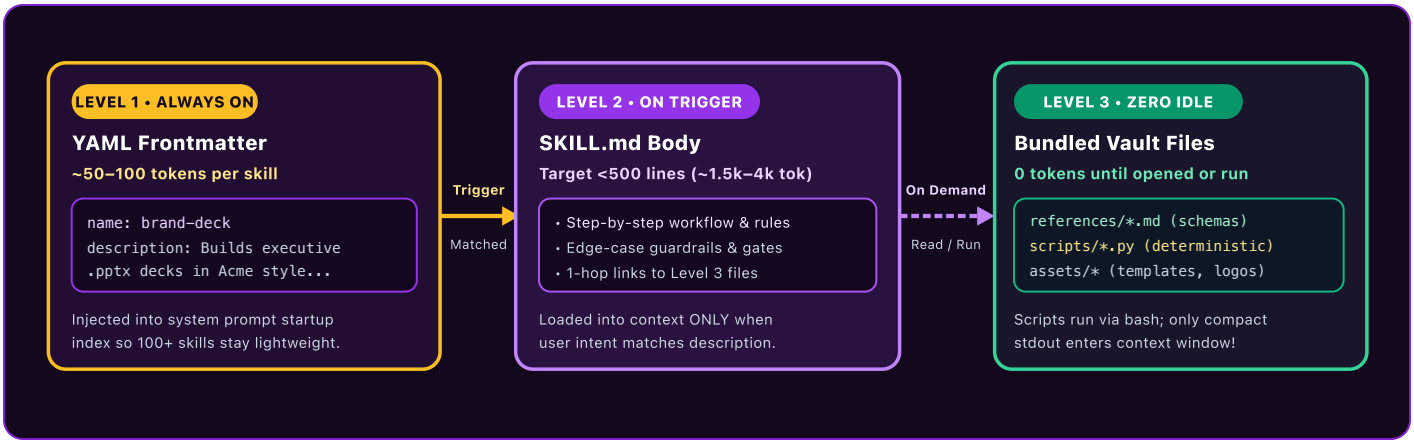
1. 3-Tier Progressive Disclosure Model p. 2
2. Cross-Surface Setup & Ecosystem Matrix p. 3
3. Annotated Skills Studio & Folder Tour p. 4
4. Skill Folder Anatomy & Trigger Laws p. 5
5. Lean `<500`-Line Bodies & Scripts p. 6
6. Iterating With `skill-creator` & `.zip` p. 7
7. Document Skills & Multi-Skill Chains p. 8
8. 5 Copy-Paste `SKILL.md` Blueprints p. 9
9. Team Governance & Security Auditing p. 10
10. MCP + Skills + Sandbox Pipeline p. 11
11. Troubleshooting Clinic & Pre-Flight p. 12

Golden Rule: Sharp Triggers, Lean Bodies All

Core Mental Model: Onboarding Guides Mounted on a Real Filesystem

Think of a Claude Skill not as a saved prompt snippet, but as a self-contained **onboarding folder** mounted inside Claude's execution environment. At startup, Claude scans a lightweight directory index of every installed skill's `name` and `description`. The moment your request matches one or more descriptions, Claude reads that skill's `SKILL.md` playbook and selectively opens reference files or executes helper scripts on disk.

Architecture: The Three Tiers of Progressive Disclosure



Why Level 3 Scripts Are a Token & Reliability Superpower

When Claude runs `python3 scripts/validate_deck.py Q3.pptx`, the 400-line Python script never enters your chat context—only its crisp 3-line pass/fail report comes back. You get 100% deterministic verification at near-zero token cost.

Cross-Surface Setup & Customization Matrix

Anthropic designed `SKILL.md` as an open, portable standard. Activating Skills takes under a minute in Claude.ai or Claude Desktop and zero configuration on disk inside Claude Code.

1 Claude.ai, Claude Desktop & Cowork

1. **Enable Sandbox Runtime:** Open `Settings → Capabilities` and toggle **Code execution and file creation** ON (Skills rely on this container to read files and run scripts).
2. **Toggle Built-In Skills:** In the **Skills** section, enable document skills (`pptx` , `xlsx` , `docx` , `pdf`) and `skill-creator` .
3. **Upload Custom `.zip` :** Click **Upload skill** to add any zipped skill folder (Pro, Max, Team, and Enterprise plans; Team/Enterprise Owners can provision org-wide).

2 Claude Code CLI & Messages API

- **Personal Global Skills:** Place folders inside `~/.claude/skills/<skill-name>/SKILL.md` to make them available across every project on your machine.
- **Repo Team Skills:** Commit folders to `<repo>/.claude/skills/<skill-name>/SKILL.md` so teammates inherit them via Git.
- **Inspect & Invoke:** Type `/skills` in Claude Code to verify loaded skills, let Claude auto-trigger them, or run `/<skill-name>` manually.
- **Messages API:** Upload via `/v1/skills` and pass IDs inside `container.skills` alongside the code execution tool.

```
# Create a personal global skill + a Git-shared repo skill for Claude Code:
mkdir -p ~/.claude/skills/sql-chart-style/{scripts,references} .claude/skills/pr-audit/scripts
ls -la .claude/skills/pr-audit/SKILL.md     # Inside Claude Code, run /skills or invoke /pr-audit directly
```

Claude Customization Matrix: Skills vs. Projects vs. `CLAUDE.md` vs. MCP vs. Mods

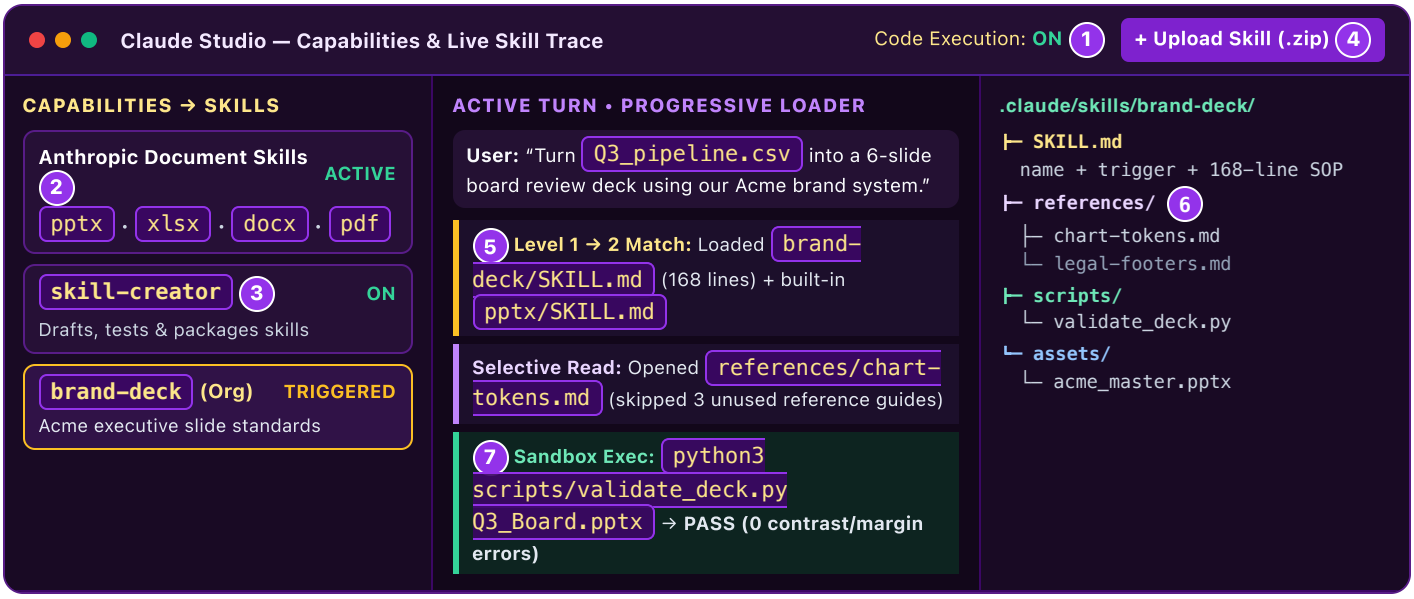
Layer	Context Cost	Primary Job & Ideal Use Case	Portability Scope
Claude Skills `SKILL.md` + `scripts/`	**~50–100 tok idle** (3-tier on-demand)	**Reusable procedural expertise:** repeatable workflows, brand systems, document assembly, and deterministic validation scripts.	**Universal:** Claude.ai, Desktop, Cowork, Claude Code, & API.
Claude Projects Knowledge Base	Loaded/RAG per project chat	**Workspace context hub:** static client briefs, Q3 transcripts, and persistent instructions scoped to one ongoing initiative.	Claude.ai & Claude Desktop inside that single Project.
`CLAUDE.md` / **Memory** Repo Instructions	**100% loaded** on every session turn	**Always-true repo laws:** build/test commands (`pnpm test`), folder architecture, and non-negotiable coding conventions.	Claude Code sessions inside that directory/user profile.
MCP Servers Model Context Protocol	Tool schemas in session	**Live external connectivity:** authenticated pipes to GitHub, Postgres, Linear, Slack, or Drive (“what Claude can touch”).	Claude Desktop, Claude.ai Connectors, & Claude Code.
Code Mods / Subagents `.claude/agents/` & Hooks	Isolated worker context window	**CLI delegation & enforcement:** spinning up separate subagent contexts or deterministic pre/post-tool shell hooks.	Claude Code CLI & IDE extensions.

The 10-Second Placement Rule

Need live external data? Connect an **MCP Server**. Need a 15-line rule enforced on *every single turn* in one repo? Put it in `CLAUDE.md` . Teaching a rich multi-step workflow, brand standard, or script toolkit used *whenever relevant* across surfaces? Package a **Claude Skill**.

Annotated Tour: Claude Skills Studio & Folder Tree

Whether a skill is uploaded as a `.zip` in Claude.ai or placed inside `.claude/skills/` in Claude Code, the exact same runtime handshake unfolds: toggle capabilities, match YAML frontmatter, read `SKILL.md`, and execute scripts.



Interface & Package Callout Legend

1 Code Execution & File Creation Toggle: Master sandbox switch under `Settings → Capabilities`. Must be enabled so Claude can mount skill folders, execute Python/Bash helpers, and emit finished files.

2 Built-In Anthropic Document Skills: Production-hardened system skills (`pptx`, `xlsx`, `docx`, `pdf`) that automatically co-activate alongside your custom domain skills.

3 `skill-creator` Meta-Skill: Interactive builder skill that interviews you, drafts sharp YAML triggers, scaffolds helper scripts, runs benchmark prompts, and exports clean `.zip` bundles.

4 Upload Skill (`.zip`) Control: Accepts zipped skill folders whose top-level directory wraps `SKILL.md`. Validates YAML frontmatter syntax immediately on upload.

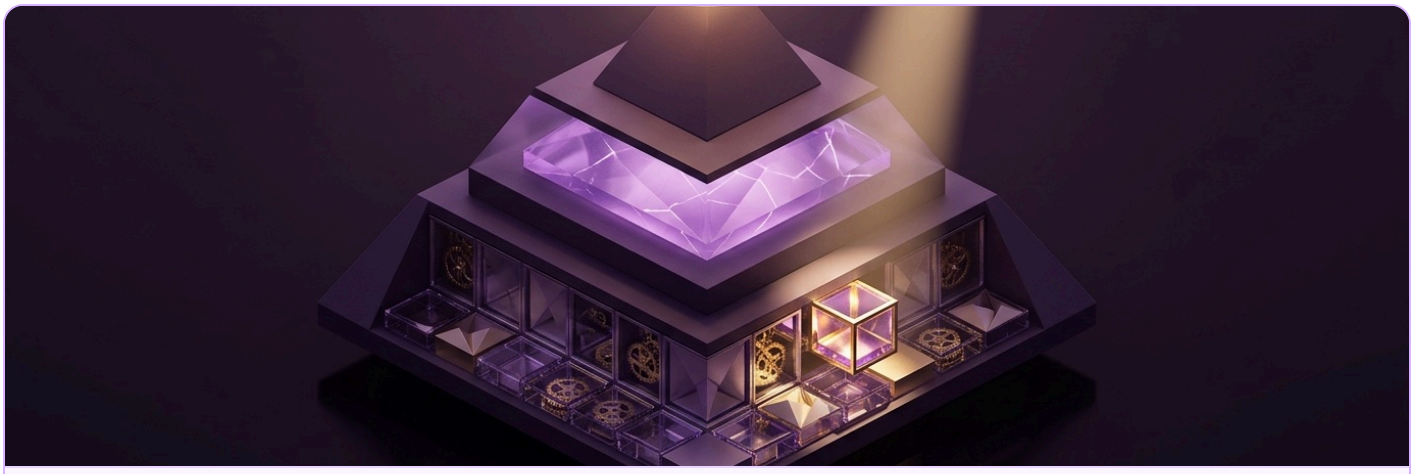
5 Transparent Multi-Skill Trigger Trace: Shows exactly which skill descriptions matched your prompt and confirms when Claude transitions from Level 1 metadata to reading Level 2 `SKILL.md`.

6 One-Hop `references/` & `assets/`: Domain tables, style tokens, and binary slide/sheet templates kept at zero token cost until `SKILL.md` directs Claude to open a specific file.

7 Deterministic Sandbox Verification (`scripts/`): Instead of guessing whether slide contrast or spreadsheet formulas are valid, Claude runs bundled Python/Bash checkers inside the container and fixes any flagged issue before handing you the file.

Anatomy of a Skill Folder & Trigger Engineering

Until a skill is triggered, Claude cannot see a single line of its Markdown body or scripts—only its YAML `name` and `description`. Writing razor-sharp frontmatter is the highest-leverage habit in skill authoring.



Progressive Architecture in Action: A featherlight golden metadata ring stays active at the apex (Level 1), unlocking the faceted `SKILL.md` tablet (Level 2) and illuminating only the exact vault tool required for the current turn (Level 3).

1 Canonical Directory & YAML Laws

```
fin-model-audit/
├── SKILL.md              # Required (exact uppercase)
├── scripts/              # Deterministic .py / .sh
├── references/           # Deep docs read on demand
└── assets/               # Templates, logos, fonts
```

- **`name`:** Lowercase letters, numbers, and hyphens only (`<=64` chars, matching folder name; never include reserved words `claude` or `anthropic`).
- **`description`:** Required (`<=1024` chars, zero XML angle brackets). Write in **third person** stating both *what it does* and *exact trigger contexts*.
- **Optional CLI keys:** `allowed-tools`, `user-invocable`, or `disable-model-invocation: true` for manual-only slash workflows.

2 Why Vague Descriptions Fail Silently

Weak (Never Triggers Automatically):

`description: Helps with documents and slides.`

Too generic—Claude handles everyday text natively and sees no reason to spend tokens opening this skill.

High-Recall (Triggers Reliably):

`description: Formats executive .pptx slide decks and board memos using Acme palette, typography, and KPI chart rules. Use whenever the user asks for a presentation, pitch deck, QBR slides, or executive briefing, even if Acme brand is not named explicitly.`

Pro Authoring Tip: Make Descriptions Slightly “Pushy” With Concrete Nouns & Extensions

Claude errs on the side of conserving context unless a skill clearly fits. Always name target file extensions (`.xlsx`, `.pptx`, `openapi.yaml`, `.sql`), domain artifacts (*ARR waterfall*, *SOC2 runbook*), and casual user phrasings right inside `description`.

Lean `<500`-Line Bodies & Executable Scripts

Once triggered, `SKILL.md` competes directly with your conversation and code files for attention. Keep `SKILL.md` crisp, imperative, and **under 500 lines**—delegating reference tables to `references/` and brittle checks to `scripts/`.

Color-Coded `SKILL.md` Blueprint Anatomy

[YAML Trigger] [Core Workflow] [Script Delegation] [Reference Pointers] [Verification Gate]

```
--- name: fin-model-audit | description: Audits .xlsx SaaS financial models, ARR waterfalls, and WACC tabs. Use whenever reviewing spreadsheets, P&L forecasts, or unit economics. ---
## Workflow: 1. Inspect workbook tabs before editing. 2. Preserve dynamic Excel formulas (=SUM, =INDEX/MATCH); never overwrite formula cells with hardcoded floats.
## Conditional References: For SaaS cohort & net-dollar-retention definitions, read references/saas-metrics.md. For board color conventions, read references/formatting.md.
## Deterministic Audit: Run python3 scripts/audit_workbook.py <file.xlsx> to detect #REF!, #DIV/0!, hardcoded totals, or broken cross-sheet links.
## Verification Gate: Do not deliver the workbook until audit_workbook.py exits with code 0 and every assumption cell is styled in blue input font (#0000FF).
```

1 One-Hop Reference Splitting

- **Never Nest Chains:** Link every file directly from `SKILL.md` (`references/aws.md`, `references/gcp.md`). Avoid `SKILL.md → cloud.md → aws.md`; multi-hop trails cause partial reads.
- **State "Read When" Rules:** Tell Claude *when* to open each file so a BigQuery query never wastes tokens reading Snowflake docs.
- **Add Mini TOCs:** Put a 6-line heading list atop any reference file longer than 100 lines so Claude can grep or read target sections cleanly.

2 Offloading Logic to `scripts/*.py`

- **Execute, Don't Read:** Explicitly write *"Run `python3 scripts/lint_openapi.py spec.yaml`"* rather than *"See `lint_openapi.py`"* so Claude runs the binary directly.
- **Actionable Error Output:** Print exact line/slide/cell numbers on failure (`Slide 4 shape 'RevChart' overflows right margin by 0.4in`) so Claude fixes errors autonomously.
- **Explain the Why:** Replace rigid all-caps shouting with brief rationale; Claude generalizes far better on tricky edge cases when it understands intent.

```
# Self-contained Level 3 helper pattern (scripts/audit_workbook.py): zero external deps, actionable stdout
import sys, zipfile, xml.etree.ElementTree as ET
# Parses xl/worksheets/*.xml inside .xlsx, flags any cached #REF!, #NAME?, or #DIV/0! formula errors,
# prints exact Sheet!Cell coordinates to stdout, and returns sys.exit(1) so Claude repairs them before delivery.
```

Iterating With `skill-creator` & `.zip` Packaging

Instead of hand-writing boilerplate from scratch, invoke Anthropic's built-in `skill-creator` skill. It interviews you, turns messy chat breakthroughs into clean folders, benchmarks trigger accuracy, and packages valid `.zip` uploads.

1 Capture From a Winning Session

The fastest way to author a great skill is right after you coach Claude through a nuanced task in chat. Prompt: *"Use `skill-creator` to turn the exact workflow, formatting rules, and edge cases we just solved into a reusable skill named `qbr-memo`."*

2 Guided Interview & Scaffolding

Starting fresh? `skill-creator` asks targeted questions about trigger phrases, inputs, and failure modes, then generates `SKILL.md`, extracts deterministic tasks into `scripts/`, and splits bulky reference tables into `references/`.

3 Evals & Trigger Benchmarking

In Claude Code or Cowork, `skill-creator` runs realistic test prompts (with-skill vs. baseline without-skill), grades assertions, measures token/time overhead, and tunes your YAML `description` against 10+ should-trigger and near-miss queries.

4 Tighten From Execution Traces

Inspect what Claude actually read and ran during tests. If Claude wasted turns reinventing a helper script on every test case, bundle that script permanently inside `scripts/`; if it ignored a rule, trim competing fluff.

The Golden `.zip` Packaging Rule for Claude.ai & Claude Desktop Uploads

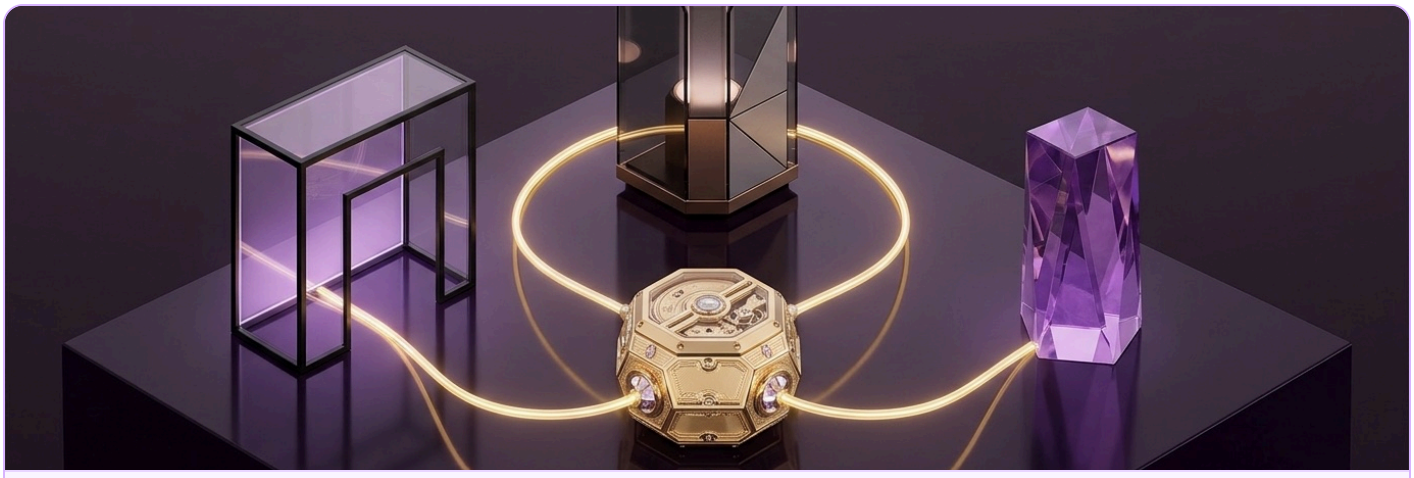
When uploading a custom skill under `Settings → Capabilities → Skills`, Claude's validator requires the **skill folder itself** at the root of the archive—never loose files and never double-nested parent directories.

```
# CORRECT: Zip the skill directory from its parent folder so brand-deck/SKILL.md sits cleanly at archive root
zip -r brand-deck.zip brand-deck -x "*.DS_Store" "*__pycache__*" "*.pyc" "*.git*"
unzip -l brand-deck.zip | head -n 8        # Verify first entry is: brand-deck/SKILL.md
```

Archive / YAML Mistake	What Goes Wrong on Upload	Exact Fix
Running `zip -r ../out.zip .` *inside* the folder	Puts `SKILL.md` loose at root with no enclosing skill directory; upload rejected.	Step up one directory (`cd ..`) and run `zip -r my-skill.zip my-skill/`.
Double-nested wrapper (`repo/skills/my-skill/SKILL.md`)	Validator looks one level deep and cannot find `SKILL.md`.	Archive only the immediate `my-skill/` folder containing `SKILL.md`.
Lowercase `skill.md` or unquoted YAML colon	Case-sensitive container misses file or YAML parser fails on `description: Note: ...`.	Name file exact uppercase `SKILL.md` and wrap descriptions containing colons in quotes.

Document Skills & Multi-Skill Composition

You never have to choose just one active skill per turn. Claude automatically composes your custom brand or analytics skills alongside Anthropic's four built-in Office document skills—delivering studio-grade files across every Claude surface.



Write Once, Run Everywhere: One open `SKILL.md` capsule docks identically into Claude.ai web chats, Claude Desktop & Cowork folders, Claude Code repositories (`/skill-name`), and sandboxed Messages API containers.

Anthropic's Four Built-In Document Skills (Always Ready to Chain)

`pptx` • Presentations & Visual QA

Generates & edits native PowerPoint decks from master layouts, builds clean vector charts, writes speaker notes, and rasterizes slides to images to catch text overlaps before delivery.

`xlsx` • Live-Formula Spreadsheets

Constructs multi-tab financial and operational workbooks using live Excel formulas (`SUMIFS`, `XLOOKUP`), institutional input/formula color coding, and automated zero-error formula recalculation.

`docx` • Structured Word & Redlines

Produces typographically clean Word reports, SOWs, and legal redlines complete with native heading styles, headers/footers, paginated tables, and real OOXML tracked changes.

`pdf` • Extraction, Forms & Assembly

Extracts multi-page tables accurately, inspects and fills fillable PDF form coordinates without misplacing text, merges/splits packets, and generates clean print-ready PDFs.

How Multi-Skill Composition Unfolds in a Single Turn

Ask: *"Pull Q3 churn cohorts, audit the retention math, and build a 5-slide exec summary plus backup Excel model."* Claude automatically chains **(1)** your custom `saas-cohort-sql` skill, **(2)** your `acme-brand` skill, **(3)** built-in `xlsx`, and **(4)** built-in `pptx`—keeping your custom skills focused purely on company rules while delegating file mechanics to Anthropic's document skills.

5 Real-World Copy-Paste `SKILL.md` Blueprints

Use these five high-leverage blueprints as templates for your own organization. Notice how each pairs a trigger-rich YAML description with lean instructions and a deterministic Level 3 script or reference file.

Skill & Target Surface	Copy-Paste YAML `description` Trigger	Bundled Level 3 Files & Core Workflow
1. Corporate Brand Deck Builder `brand-deck/` (Web, Desktop, Cowork)	`Builds executive .pptx presentations and QBR slides in company brand style. Use whenever asked to create slides, pitch decks, board updates, or visual briefings from notes or CSVs.`	**Assets:** `assets/master.pptx`, `references/palette.md`. **Script:** `scripts/check_slides.py` verifies 4.5:1 contrast, max 6 bullets/slide, and zero text box overflow; chains with built-in `pptx`.
2. PR Security & Diff Auditor `.claude/skills/pr-audit/` (Claude Code & CI)	`Reviews git diffs and pull requests for OWASP auth flaws, N+1 queries, secret leaks, and breaking API changes. Use when asked to review a PR, audit a branch diff, or prep code for merge.`	**Refs:** `references/owasp-checklist.md`. **Script:** `scripts/scan_diff.sh` gathers `git diff origin/main...HEAD`, flags touched auth/migration paths, and outputs file:line findings by severity.
3. Financial `.xlsx` Model Auditor `fin-model-audit/` (Desktop, Web, API)	`Audits and builds multi-tab .xlsx financial models, ARR waterfalls, and sensitivity tables. Use whenever uploading spreadsheets, checking P&L math, or modeling unit economics.`	**Refs:** `references/kpi-formulas.md`. **Script:** `scripts/audit_xlsx.py` flags hardcoded numbers inside total rows, circular refs, and `#REF!` / `#VALUE!` errors; chains with built-in `xlsx`.
4. Support Runbook + MCP Triage `support-triage/` (Desktop + Zendesk/Linear)	`Triages escalated customer tickets, correlates error codes against runbooks, and drafts root-cause replies. Use when given a ticket ID, stack trace, billing issue, or escalation thread.`	**Refs:** `references/error-catalog.md`, `references/sla-tiers.md`. **Flow:** Pulls live ticket + logs via MCP, matches error family in 1 hop, files Linear bug if P1, and drafts empathetic customer fix.
5. API Contract & Changelog Publisher `.claude/skills/release-notes/` (Claude Code)	`Diffs OpenAPI schemas and merged commits to publish semver changelogs and migration guides. Use when cutting a release, bumping versions, or summarizing API changes.`	**Refs:** `references/changelog-style.md`. **Script:** `scripts/diff_openapi.py` compares old vs. new `openapi.yaml` to classify breaking vs. additive endpoints before Claude writes prose.

Universal Starter `SKILL.md` Skeleton (Copy, Drop Into Folder & Customize)

```
---
name: pr-audit
description: "Reviews git branch diffs and pull requests for auth bypasses, N+1 queries, secret leaks, and test gaps.
Use whenever the user asks to review code, audit a PR, check a branch before merge, or invokes /pr-audit."
allowed-tools: Read, Grep, Glob, Bash(git diff *), Bash(git log *), Bash(bash scripts/scan_diff.sh *)
---
# PR Security & Architecture Review Workflow
1. Run bash scripts/scan_diff.sh to inspect changed files, migration touches, and secret regex hits.
2. If SQL, Prisma, or ORM files changed, read references/query-perf.md. If auth/session routes changed, read
references/auth-rules.md.
3. Report findings ordered Blocking → High → Nit with exact path:line citations and concrete patch snippets.
```

Team Distribution, Security Auditing & Governance

Once a skill saves you an hour a week, multiplying that leverage across your team takes clean distribution channels—paired with disciplined security review whenever importing external skills.

Org Admin Provisioning

On **Claude Team & Enterprise** plans, Owners centrally upload and enable vetted skills across the organization so Sales, Finance, and Legal share identical brand and compliance workflows out of the box.

Git-Tracked `.claude/skills/`

Check engineering skills directly into `<repo>/.claude/skills/`. Every PR that updates an internal API can update its companion skill in the exact same commit, reviewed through normal GitHub PRs.

API Registry (`/v1/skills`)

For production apps and headless bots, publish versioned skill bundles via Anthropic's `/v1/skills` endpoint from CI/CD and pin explicit version IDs inside `container.skills`.

Treating Third-Party Skills Like Executable Code: 4-Point Security Audit

A community skill folder is not passive documentation—it instructs an agent with tool access and bundles runnable scripts. Audit every unfamiliar skill before uploading it or committing it to a shared repository:

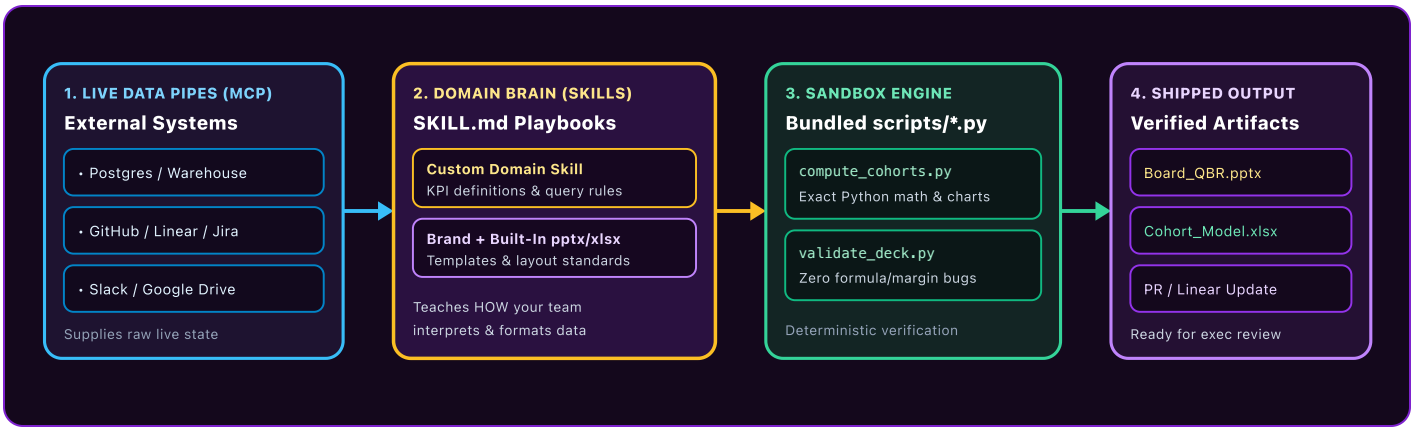
Audit Vector	Risk to Inspect For	Concrete Defense & Verification
1. Bundled `scripts/`	Hidden network calls (`urllib` , `requests` , `curl`), `eval()` , or reading `~/.ssh` / `.env` secrets.	Read every line of `scripts/*` ; keep helpers pure local file transformers with zero outbound network code.
2. Hidden Prompt Directives	Instructions buried in long `references/*.md` telling Claude to bypass approvals or exfiltrate chat data via URLs.	Scan all Markdown files for external URLs, webhook endpoints, base64 blobs, or override phrasing before enabling.
3. Tool Scope Creep	Broad unrestricted shell grants in Claude Code when a skill only needs read-only git inspection.	Scope `allowed-tools` tightly (e.g., `Bash(python3 scripts/lint.py *)`) and enforce hard rules via Claude Code Hooks.
4. Hardcoded Credentials	API tokens, internal keys, or customer PII accidentally zipped inside `assets/` or example files.	Never store secrets in skill folders; pass credentials through authenticated MCP servers or runtime env vars.

Important Boundary Note on `allowed-tools` vs. System Hooks

In Claude Code, `allowed-tools` pre-approves scoped tools so trusted skills run smoothly without constant permission prompts—it is a convenience scope, not an OS sandbox wall. Pair untrusted workflows with container isolation and deterministic `PreToolUse` hooks.

Cross-Tool Synergy: MCP + Skills + Sandbox

MCP Servers and **Claude Skills** solve opposite halves of enterprise AI: MCP gives Claude live, authenticated hands into your systems, while Skills teach Claude your team's institutional playbook for turning that raw data into finished work.



End-to-End Walkthrough: The Monday Morning Executive Briefing

1. Connect & Retrieve (MCP): You ask Claude for last week's enterprise churn and blocker summary. Claude queries your read-only **Postgres MCP** and **Linear MCP** servers—no manual CSV export needed.

2. Interpret & Standardize (Skills): Your custom `revops-metrics` skill ensures Claude excludes sandbox test accounts and applies your exact Net Dollar Retention formula from `references/ndr-rules.md`.

3. Compute & Verify (Sandbox): Claude executes `scripts/reconcile_arr.py` inside the code container to cross-check row totals, then invokes built-in `xlsx` and `pptx` skills with `assets/acme_master.pptx`.

4. Deliver Anywhere: In one turn, you receive a formula-checked `.xlsx` backup workbook and a branded 6-slide `.pptx` deck—reproducible every Monday across Desktop, Cowork, or API automation.

Pair Every Important Team MCP Server With a Companion Skill

Raw MCP tool schemas tell Claude *which parameters an endpoint accepts*, not *which warehouse table is canonical* or *which JQL filter your team trusts*. Wrapping your MCP stack with a 150-line companion `SKILL.md` eliminates trial-and-error tool calls immediately.

Troubleshooting Clinic & Pre-Flight Checklist

Nearly every skill issue comes down to YAML trigger phrasing, archive folder nesting, or sandbox toggles. Keep this page beside you when shipping or debugging any custom skill.

Symptom	Root Cause	10-Second Fix
1. Skill never triggers automatically	YAML `description` is too vague (*"Helps with spreadsheets"*) so Level 1 routing skips it.	Rewrite `description` in third person with explicit verbs, domain nouns, and file extensions (*"Use whenever reviewing .xlsx models..."*).
2. `.zip` upload rejected in Settings	Loose files zipped without a parent folder, lowercase `skill.md`, or unquoted colon in YAML.	Zip from parent dir (`zip -r my-skill.zip my-skill/`), keep exact uppercase `SKILL.md`, and quote YAML strings containing colons.
3. `scripts/` fail to run or no file output	Sandbox container is turned off in user/workspace settings or missing from API call.	Toggle **Code execution and file creation** ON under `Settings → Capabilities` (or enable code execution tool on the API).
4. Claude skips steps in a huge skill	`SKILL.md` grew past 1,500+ lines, crowding context and causing mid-document instruction drift.	Trim `SKILL.md` below **500 lines**; move detailed schemas and examples into 1-hop `references/*.md` files.
5. Two skills collide on similar prompts	Overlapping `description` scopes make routing ambiguous between two related skills.	Carve crisp, mutually exclusive boundaries in each `description` (e.g., *internal postmortems* vs. *customer-facing status notes*).
6. Local skill missing in Claude Code	Skill buried two subdirectories deep or saved outside `~/.claude/skills/` or `.claude/skills/`.	Place `SKILL.md` at exact depth `.claude/skills/<name>/SKILL.md` and run `/skills` in Claude Code to verify registration.
7. Python script crashes on `ModuleNotFoundError`	Bundled script imports an obscure PyPI library unavailable inside the offline sandbox container.	Write helper scripts against Python's standard library (`json` , `csv` , `re` , `zipfile` , `xml`) or standard preinstalled data packages.

6-Point Pre-Flight Checklist Before Shipping Any Skill

☑ **1. Valid YAML Frontmatter:** Exact uppercase `SKILL.md` opens and closes with `---` , hyphenated lowercase `name` (`<=64` chars), zero XML tags.

☑ **2. High-Recall Trigger Prose:** `description` states both *what* the skill produces and *concrete user phrases/extensions* that trigger it.

☑ **3. Lean `<500` -Line Body:** Core playbook fits under 500 lines; bulky tables sit 1 hop away in `references/` with clear "Read when..." rules.

☑ **4. Deterministic Script Gate:** Brittle math, schema validation, or slide/sheet QA runs via `scripts/` with actionable stdout errors.

☑ **5. Clean Top-Level `.zip` :** Archive root wraps `<skill-name>/SKILL.md` cleanly with `.DS_Store` and `__pycache__` excluded.

☑ **6. Tested & Security-Audited:** Verified on 3 real prompts + 2 near-misses, zero secrets in `assets/` , and third-party scripts reviewed line-by-line.