Corp-OS Skill Map

23 skills, and only 13 of them are things you do *with* a knowledge base. The other 10 are things you do **to** it — and knowing which is which is most of knowing when to reach for what.

23	8	18	14	5
SKILLS	SLASH COMMANDS	SHIPPED SCRIPTS	REFERENCE SPECS	INVARIANTS

THE LOOP — WHAT YOU DO WITH THE OS

PHASE 1
Capture
Material lands in the append-only layer. Existing there means *said*, not true.

PHASE 2
Curate
Raw becomes citable — a kind, a confidence, a verbatim quote, a decay window.

PHASE 3
Consult
The only phase that pays you back. Questions answered with provenance attached.

PHASE 4
Correct
Confidence outlives its evidence unless something goes looking for it.

PHASE 5
Release
The boundary. Where an OS stops helping and starts costing you something.

↩ Release returns to Capture — what you sent out is itself material

PHASE 1 Capture

3 skills

WHAT IT IS FOR
Getting things said into a layer that is never edited, never summarized in place, never deleted outside a declared obligation. One sanctioned exception: flipping `processed` to true.

SKIP IT AND
Nothing downstream can cite anything. A claim with no retrievable original is the one thing this model treats as corrosive, because nothing later can detect it.

corp-os-intake

Takes material a person pastes, uploads, or hands over directly — a transcript, a chat thread, a document, an email, a personal note, a screenshot of something — and files it into a Corp-OS knowledge base's append-only raw layer with the right type and job tags, then proposes what should become claims.

"add this" *"log this"* *"capture this conversation"*

> Not for retrieving from a connected source (use corp-os-pull) and not for answering a question about material already captured (use corp-os-recall).

`/corp-os-capture` `As it happens`

corp-os-connect

Registers a data source in a Corp-OS knowledge base — establishing how it connects (MCP connector, API token, browser session, export file, or manual paste), what it feeds, which jobs it serves, its cadence, and critically what it is blind to.

"connect my meeting notes" *"hook up Slack"* *"add a data source"*

> Not for actually retrieving data from an already-registered source (use corp-os-pull) and not for one-off pasted content (use corp-os-intake).

`Once per source`

corp-os-pull

Retrieves new material from the sources registered in a Corp-OS knowledge base — meeting notes, chat threads, email, documents, issue trackers — writing each into the append-only raw layer with dedupe, then proposing what should become claims.

"catch me up" *"pull my meetings"* *"sync my OS"*

> Not for registering a new source (use corp-os-connect), not for pasted or handed-over content (use corp-os-intake), and not for answering a question about material already captured (use corp-os-recall).

`/corp-os-catchup` `Daily or weekly`

PHASE 2 Curate

5 skills

WHAT IT IS FOR

Turning a pile into knowledge, behind a review gate written to disk before it is presented. The declines are the valuable part — the only record of what someone chose not to know.

SKIP IT AND

You have a folder of transcripts. Search finds text; it does not find what you concluded, who said it, or whether it is still true.

corp-os-claims

Turns raw material in a Corp-OS knowledge base into reviewed, citable claims — each with a kind, a confidence level, a verbatim citation, the jobs it serves, and a decay window — and promotes, edits, merges, supersedes, or retires existing claims.

“turn this into claims”

“process my unprocessed queue”

“what do we actually know”

> Not for capturing new source material (use corp-os-intake or corp-os-pull) and not for the systematic staleness and contradiction sweep (use corp-os-reality-check).

`Weekly — work the queue`

corp-os-jobs

Adds, sharpens, splits, re-scopes, or retires the jobs to be done in a Corp-OS knowledge base, and keeps each job's evidence list current so intake and recall have a priority signal.

“add a job” *“I have a new thing I'm working on”*

“this job is too big”

> Not for setting up an OS from scratch (use corp-os-setup) and not for capturing source material (use corp-os-intake).

`When the work changes`

corp-os-decide

Tracks the forks a person has NOT decided yet in a Corp-OS knowledge base — each open choice with its named options, an owner, a decide_by date past which it gets made by default, what it is blocking, and what evidence would settle it — then records the outcome and hands it to corp-os-claims once the call is made. The test is tense: this is only for a choice still open.

“we're stuck between two options”

“what's still open” *“what am I waiting on”*

> Not for a decision already made — "we went with X", "why did we decide that" — which is a decision claim via corp-os-claims. Not for an outcome someone is working toward, like "decide how to handle renewals", which is a job (use corp-os-jobs).

`/corp-os-open` `When a fork opens`

corp-os-glossary

Builds and maintains the glossary in a Corp-OS knowledge base — internal jargon, product terms, acronyms, role names, and especially metric definitions — extracted from captured material and confirmed by the person.

“add this term” *“what does X mean here”*

“build my glossary”

> Not for company-level facts like pricing or funding (use corp-os-company) and not for general claims (use corp-os-claims).

`As terms surface`

corp-os-company
Researches and maintains company context in a Corp-OS knowledge base — what a company sells, users versus buyers, use cases, pricing mechanism, monetization, funding, current status, stated goals, and competitors — combining web research with what the person already knows, and recording each finding as a sourced claim with a decay window.

"research this company"

"what do we know about Acme"

"build out my company context"

> Not for defining internal vocabulary (use corp-os-glossary) and not for filing a call transcript (use corp-os-intake).

`Before a meeting`

PHASE 3 Consult

3 skills

WHAT IT IS FOR
Getting the knowledge back out with its provenance still attached. Three shapes, because a question, a standing digest, and a page you return to are genuinely different needs.

SKIP IT AND
You have built a write-only diary. This is the phase that repays the first two, and the one people quietly stop reaching for first.

corp-os-recall
Answers questions from a Corp-OS knowledge base with citations, and enriches whatever the person is currently working on with the relevant claims, glossary terms, people history, and company context the OS already holds.

"what do we know about X"

"what did they say about Y"

"brief me before this meeting"

> Not for capturing new material (use corp-os-intake or corp-os-pull) and not for the systematic staleness sweep (use corp-os-reality-check).

`/corp-os-recall` `Constantly`

corp-os-brief
Produces the recurring operating brief for a Corp-OS knowledge base — what came in, which jobs moved, what needs review, what went stale, what is blocked, and the single most useful thing to do next — and can set itself up as a scheduled daily or weekly run.

"brief me" *"what's the state of my OS"*

"my weekly review"

> Not for answering a specific question (use corp-os-recall) and not for a visual dashboard (use corp-os-dashboard).

`/corp-os-brief` `Daily or weekly`

corp-os-dashboard

Builds and refreshes dashboards from what a Corp-OS knowledge base actually holds — a job board, evidence gaps, claim health, source coverage, stakeholder map, company brief, or decision log — published as artifacts and recorded in the dashboard registry, following whatever design system the OS binds.

"build me a dashboard" *"show me my jobs"* *"visualize my OS"*

> Not for a written brief in chat (use corp-os-brief) and not for answering a specific question (use corp-os-recall).

`Build once, refresh on cadence`

PHASE 4 Correct

1 skill

WHAT IT IS FOR

The fifth invariant made routine: something has to remove things. Claims past their decay window, claims never verified, contradictions, assumptions still worn as facts.

SKIP IT AND

The OS keeps answering with confidence it has not earned since March. That is worse than having no OS, because you now trust the answer.

corp-os-reality-check

Sweeps a Corp-OS knowledge base for knowledge that has drifted from reality — claims past their decay window, claims never verified, contradictions between claims, assumptions still treated as facts, and jobs whose evidence no longer holds — then interrogates the person to resolve each one.

"is my OS still accurate" *"what's stale"* *"check my claims"*

> Not for creating new claims from source material (use corp-os-claims) and not for regenerating the whole derived layer (use corp-os-rebuild).

`/corp-os-check` `Monthly`

PHASE 5 Release

1 skill

WHAT IT IS FOR

One operator, but exports land in multi-person systems. Sensitivity is two axes: `sensitivity` says what must not leave, `bearing` says whether the OS can reason correctly without it.

SKIP IT AND

The wrong sentence goes out in a paste. Nothing is silently deleted — the cleaned copy and a private log of exactly what came out are written together, or neither is.

corp-os-redact

Produces a shareable version of anything leaving a Corp-OS knowledge base — a brief, a claim set, a dashboard, a company record, a raw transcript — with personal, personnel, compensation, credential, and confidential material removed, plus a private local log of exactly what was pulled and why.

"clean this up before I share it" *"scrub this"* *"make this safe to send"*

> Not for deciding what is worth sharing on relevance grounds — that judgment stays with the person.

`/corp-os-share` `Every time something leaves`

OFF THE LOOP The workbench

10 skills

These act on the OS itself rather than on what it holds, so they are not a sequence and there is no order to them. Only one has a slash command, and that is deliberate: the commands cover the moments that recur, and these are rare and deliberate enough that you name them.

corp-os-guide

Explains what Corp-OS is and routes to the right Corp-OS skill for the situation.

"what is Corp-OS" *"how does this work"* *"what can my OS do"*

> Not for performing any of the operations itself — this skill only orients and hands off.

`/corp-os` `Whenever you are unsure`

corp-os-setup

Interrogates someone about their role, mandate, jobs to be done, valuable data, input paths, connected tools and protocols, company, design system, and sensitivity needs — then scaffolds a Corp-OS personal work knowledge base tailored to those answers and builds their first dashboard.

"set up Corp-OS" *"build a personal OS"*

"build a second brain for my work"

> Not for adding content to an existing OS (use corp-os-intake or corp-os-pull) and not for assessing a system someone already built by other means (use corp-os-audit).

`Once`

corp-os-migrate

Brings an existing body of work into a Corp-OS — a previous knowledge system, a wiki or notes-app export, a folder of documents, a hand-grown second brain — filing it into the append-only raw layer, setting a cohort confidence ceiling for material whose originals cannot be re-fetched, spreading decay windows so the first sweep is clearable, and recording what did not come across and why.

"I have six months of notes to bring in"

"migrate my old system" *"import my wiki"*

> Not for one pasted item (use corp-os-intake), not for a source that is already registered (use corp-os-pull), and not for creating the OS itself (use corp-os-setup).

`Once, at the start`

corp-os-configure

Changes how a Corp-OS is shaped — renaming its vocabulary, adding or disabling layers, defining an entirely new custom layer with its own schema, adjusting decay windows, setting a retention or TTL policy on raw material, and tuning how strict the review gate is — handling any change that needs a migration as a migration.

"add a layer for X"

"call them findings instead of claims"

"my raw folder is getting huge"

> Not for setting up a new OS (use corp-os-setup) and not for content accuracy (use corp-os-reality-check).

`When the shape stops fitting`

corp-os-upgrade

Brings an existing Corp-OS up to date with a newer version of the plugin — refreshing the shipped scripts the OS carries its own copies of, recording which version it is now on, and naming the structural migrations the version gap implies.

"does my OS need anything now"

> Not for reshaping an OS to fit the person (use corp-os-configure) and not for creating one (use corp-os-setup).

`After every plugin update`

corp-os-rebuild

Regenerates the entire derived layer of a Corp-OS knowledge base — claims, jobs, people, topics, glossary, indexes — from scratch out of the full raw archive, splitting overloaded groupings, merging duplicates, retiring what raw no longer supports, and reporting exactly what changed structurally.

"rebuild my OS" *"regenerate my claims"*

"this has drifted, redo it from source"

> Not for a targeted staleness sweep (use corp-os-reality-check) and not for adding new material (use corp-os-intake or corp-os-pull).

`Rarely`

corp-os-audit

Audits any existing personal or team knowledge system — one built from Corp-OS, from a different model, or grown by accident — across ten dimensions, then reports the gaps worth closing, the new skills and workflow changes worth building, and the structures that system has which the Corp-OS model lacks, exporting the generalizable ones as an improvement packet.

"review my second brain"

"audit my knowledge base"

"compare my setup to yours"

`On someone else's system`

corp-os-improve

Analyzes how a Corp-OS knowledge base is actually being used — from its usage log, structure, and drift — and proposes specific changes to the OS's own shape, plus an anonymized improvement packet that can be sent back to whoever maintains the Corp-OS model.

"improve my OS"

"this is getting annoying to use" *"is this working"*

> Not for fixing content accuracy (use corp-os-reality-check) and not for regenerating the derived layer (use corp-os-rebuild).

`After enough real use`

corp-os-pattern

Turns an output someone has already built — a dashboard, an export, a deck, a recurring document — into a portable pattern that other people's OSes can bind, and adopts patterns from a shared team pack. A pattern addresses layers by role and field rather than by name, so it survives an OS that renamed its vocabulary, and binding either resolves or names exactly what is missing.

"make this repeatable"

"share this dashboard with my team"

"adopt our team's pack"

> Not for reshaping this OS's own layers (use corp-os-configure) and not for proposing changes to the corp-os plugin itself (use corp-os-contribute).

`Once per output worth repeating`

corp-os-contribute

Turns friction already surfaced in a Corp-OS operator's usage — by corp-os-improve, corp-os-audit, or raised directly in conversation — into apply-ready diffs against the corp-os plugin's own source. Reads the plugin's actual SKILL.md files, reference docs, and scripts; never the operator's claim bodies or person records.

"write this up for the plugin repo"

"give me a diff I can apply upstream"

"how would I fix corp-os itself"

> Not for changing this OS's own config (use corp-os-configure) and not for the routine, evidence-gated local-improvement pass (use corp-os-improve) — this skill assumes that work, or its equivalent, already happened.

`Maintainer path`

SUPPORTING LAYER

8 commands, named for the moment

Not one per skill. Someone reaching for the export boundary is thinking *share this*, not *redact*. Whether the one-line descriptions are actually distinguishable from each other was measured rather than assumed: picker cases give a person plain English and the roster and nothing else.

COMMAND	THE MOMENT	REACHES
`/corp-os-brief`	Run the recurring Corp-OS brief — what moved, what needs a decision, what went stale	`brief`
`/corp-os-capture`	File something into your Corp-OS — a transcript, a thread, a document, or a note	`intake`
`/corp-os-catchup`	Pull what is new from your connected sources and see what moved	`pull`
`/corp-os-check`	Sweep your OS for what has gone stale, contradicted itself, or was never verified	`reality-check`
`/corp-os-open`	See the decisions you have open — what is past its date, and what is blocked on whom	`decide`

COMMAND	THE MOMENT	REACHES
`/corp-os-recall`	Ask your Corp-OS a question, or fold what it knows into what you are working on	`recall`
`/corp-os-share`	Make something safe to send — a cleaned copy, plus a private record of what came out	`redact`
`/corp-os`	Open Corp-OS — explains the system, finds your OS, and routes to the right operation	`guide`

SUPPORTING LAYER

18 scripts, because instruction stopped working

Each of these was prose first. Each held at a stubborn rate — 33%, 67% — through two or more rewrites, and only moved when it became code. The rule they encode: **a step that has to happen every time, that nothing else will catch if it is skipped, belongs in code rather than in an instruction.** Your OS carries its own copies of the ones marked below, which is exactly why `upgrade_os.py` had to exist.

SCRIPT	WHAT IT MAKES DETERMINISTIC	LIVES
`bind_pattern.py`	Resolve a pattern's requirements against an OS, or say exactly what is missing.	In your OS
`build_index.py`	Deterministic INDEX.md regenerator and drift checker for a corp-os.	In your OS
`check_citations.py`	Find entries that share a citation and disagree about it.	In your OS
`check_connector.py`	Check every connector record against the shape it declares.	In your OS
`check_shield.py`	Prove a built output's screen-share shield is real and not decorative.	In your OS
`delete_source.py`	Destroy raw material under a retention obligation, in the right order.	In your OS
`file_raw.py`	File fetched items into raw/ in one call. Dedupe, name, write, cutoff.	In your OS
`find.py`	Return the entries that match, instead of reading the files that contain them.	In your OS
`friction_scan.py`	Count what the usage log actually supports, before anything is proposed.	In your OS
`log_run.py`	Close out a skill run: the usage-log row and the meta.json history entry.	In your OS

SCRIPT	WHAT IT MAKES DETERMINISTIC	LIVES
`migrate_schema.py`	Add a schema field across a layer, filling only what can be derived.	In your OS
`propose.py`	Write the gate's proposal file, and later record what was decided.	In your OS
`prune_config_notes.py`	Move `note` fields out of config.json and into the OS's own README.	In your OS
`scaffold.py`	Create the mandatory core of a Corp-OS. Not the content — the skeleton.	In the plugin
`source_yield.py`	What each source produced, against how much of it ever became anything.	In your OS
`stagger_decay.py`	Spread a migrated corpus's decay windows so the first sweep is clearable.	In your OS
`upgrade_os.py`	Bring an existing Corp-OS up to date with the installed plugin.	In the plugin
`write_export.py`	Write a redaction's two outputs, together, with fixed names.	In your OS

GETTING IT

Two things are called “update” and they are not the same

Updating the plugin replaces the skills. Updating your OS brings the folder those skills operate on in line with them — its copies of the shipped scripts above, the version it records, and any shape change the release implies. **Doing the first does nothing to the second**, because an OS carries its own copies so that it still works when the plugin is not loaded. That is the whole reason `corp-os-upgrade` exists.

DO THIS	TO
`/plugin marketplace add brentgann/corp-os`	Point your client at this repository’s marketplace manifest
`/plugin install corp-os@brentgann`	Install the plugin. In the desktop app, use the plugin browser rather than the slash command
`/plugin marketplace update brentgann`	Refetch. This is the update that replaces *the skills*
`ask for corp-os-upgrade`	The other update: refresh the script copies inside your OS, stamp the version, and name the migrations it refuses to perform for you

One thing that catches maintainers as often as users: a client caches an installed plugin **by version string**, so commits pushed without a bump in `plugin.json` reach nobody — no error, no warning, nothing to inspect from the inside. The full procedure is in `docs/INSTALL.md`.

MAINTAINER NOTES — SKIP IF YOU ARE HERE TO USE IT

What is measured, and what is not

The honest numbers behind the map above, read at build time from the committed run reports rather than typed in, and kept here rather than woven through, because an operator does not need them and a maintainer should not have to dig.

> **Routing: 100%** over 77 queries × 3 repeats at 23 skills, including near-miss pairs written to pull each new skill toward its nearest neighbours. The descriptions can be told apart from each other. Whether a skill fires in a live session competing with everything else installed is a different question, and unmeasured.

> **Conformance: 114/122** across 19 cases, 15 skills and 4 fixtures — at one repeat, so a baseline rather than rates. 8 skills still have no case.

> **Every conformance figure before v0.11.0 is void.** No shipped script had ever executed in a run: the harness ran with `acceptEdits`, which approves file edits and nothing else, so every `python3 scripts/…` was answered with *this command requires approval*. The file-level results were real; what produced them was never measured, because a model blocked from running a script writes the row by hand. It cost six runs and three instruction passes chasing a setup failure that was the instrument the whole time.

> **One hand-off lands 2 times in 5.** When a dashboard needs a layer the OS has not declared, the skill should hand off to `corp-os-configure` rather than improvise one inline. Five runs, three behaviours. A wording pass making the requirement explicit measured 0/2 and was reverted — prose that does not move the number is churn. It is on the record at 2/5 rather than reworded a fourth time.

> **The interrogation is untested by construction.** It needs a person who did not design it. The likeliest way setup fails is not a wrong answer but someone abandoning it halfway, and no check catches that — which is why the scaffolder runs first, so they still have a working OS when they do.

> **This page is generated, and that is the point.** A static skill map is a fact maintained in a second place, which is the defect this repo has now found four times. Everything above is read from the plugin at build time; only the phase a skill belongs to is declared by hand, and `validate.py` fails if a skill on disk has not been placed.

5 invariants: An append-only source layer · Provenance · A review gate · An index that can be scanned · Something that removes things.