Skills

Skills

OpenClaw uses **AgentSkills-compatible** skill folders to teach the agent how to use tools. Each skill is a directory containing a `SKILL.md` with YAML frontmatter and instructions. OpenClaw loads **bundled skills** plus optional local overrides, and filters them at load time based on environment, config, and binary presence.

Locations and precedence

Skills are loaded from **three** places:

1. **Bundled skills**: shipped with the install (npm package or OpenClaw.app)
2. **Managed/local skills**: `~/.openclaw/skills`
3. **Workspace skills**: `<workspace>/skills`

If a skill name conflicts, precedence is:

`<workspace>/skills` (highest) → `~/.openclaw/skills` → bundled skills (lowest)

Additionally, you can configure extra skill folders (lowest precedence) via `skills.load.extraDirs` in `~/.openclaw/openclaw.json`.

Per-agent vs shared skills

In **multi-agent** setups, each agent has its own workspace. That means:

- **Per-agent skills** live in `<workspace>/skills` for that agent only.
- **Shared skills** live in `~/.openclaw/skills` (managed/local) and are visible to **all agents** on the same machine.
- **Shared folders** can also be added via `skills.load.extraDirs` (lowest precedence) if you want a common skills pack used by multiple agents.

If the same skill name exists in more than one place, the usual precedence applies: workspace wins, then managed/local, then bundled.

Plugins + skills

Plugins can ship their own skills by listing `skills` directories in `openclaw.plugin.json` (paths relative to the plugin root). Plugin skills load when the plugin is enabled and participate in the normal skill precedence rules. You can gate them via `metadata.openclaw.requires.config` on the plugin's config entry. See **Plugins** for discovery/config and **Tools** for the tool surface those skills teach.

ClawHub (install + sync)

ClawHub is the public skills registry for OpenClaw. Browse at **https://clawhub.com**. Use it to discover, install, update, and back up skills. Full guide: **ClawHub**.

Common flows:

- Install a skill into your workspace:

```
clawhub install <skill-slug>
```

- Update all installed skills:

```
clawhub update --all
```

Sync (scan + publish updates):

```
clawhub sync --all
```

By default, `clawhub` installs into `./skills` under your current working directory (or falls back to the configured OpenClaw workspace). OpenClaw picks that up as `<workspace>/skills` on the next session.

Security notes

Treat third-party skills as **untrusted code**. Read them before enabling.

Prefer sandboxed runs for untrusted inputs and risky tools. See Sandboxing.

`skills.entries.*.env` and `skills.entries.*.apiKey` inject secrets into the **host** process for that agent turn (not the sandbox). Keep secrets out of prompts and logs.

For a broader threat model and checklists, see Security.

Format (AgentSkills + Pi-compatible)

`SKILL.md` must include at least:

```
---
name: nano-banana-pro
description: Generate or edit images via Gemini 3 Pro Image
---
```

Notes:

We follow the AgentSkills spec for layout/intent.

The parser used by the embedded agent supports **single-line** frontmatter keys only.

`metadata` should be a **single-line JSON object**.

Use `{baseDir}` in instructions to reference the skill folder path.

Optional frontmatter keys:

- `homepage` — URL surfaced as “Website” in the macOS Skills UI (also supported via `metadata.openclaw.homepage`).
- `user-invocable` — `true|false` (default: `true`). When `true`, the skill is exposed as a user slash command.
- `disable-model-invocation` — `true|false` (default: `false`). When `true`, the skill is excluded from the model prompt (still available via user invocation).
- `command-dispatch` — `tool` (optional). When set to `tool`, the slash command bypasses the model and dispatches directly to a tool.
- `command-tool` — tool name to invoke when `command-dispatch: tool` is set.
- `command-arg-mode` — `raw` (default). For tool dispatch, forwards the raw args string to the tool (no core parsing).

  The tool is invoked with params: `{ command: "<raw args>", commandName: "<slash command>", skillName: "<skill name>" }`.

Gating (load-time filters)

OpenClaw **filters skills at load time** using `metadata` (single-line JSON):

```
---
name: nano-banana-pro
description: Generate or edit images via Gemini 3 Pro Image
metadata:
  {
    "openclaw":
      {
        "requires": { "bins": ["uv"], "env": ["GEMINI_API_KEY"], "config": ["brows
        "primaryEnv": "GEMINI_API_KEY",
      },
  }
---
```

Fields under `metadata.openclaw`:

- `always: true` — always include the skill (skip other gates).
- `emoji` — optional emoji used by the macOS Skills UI.
- `homepage` — optional URL shown as “Website” in the macOS Skills UI.
- `os` — optional list of platforms (`darwin`, `linux`, `win32`). If set, the skill is only eligible on those OSes.
- `requires.bins` — list; each must exist on `PATH`.
- `requires.anyBins` — list; at least one must exist on `PATH`.
- `requires.env` — list; env var must exist **or** be provided in config.
- `requires.config` — list of `openclaw.json` paths that must be truthy.
- `primaryEnv` — env var name associated with `skills.entries.<name>.apiKey`.
- `install` — optional array of installer specs used by the macOS Skills UI (brew/node/go/uv/download).

Note on sandboxing:

- `requires.bins` is checked on the **host** at skill load time.

If an agent is sandboxed, the binary must also exist **inside the container**. Install it via `agents.defaults.sandbox.docker.setupCommand` (or a custom image). `setupCommand` runs once after the container is created. Package installs also require network egress, a writable root FS, and a root user in the sandbox. Example: the `summarize` skill (`skills/summarize/SKILL.md`) needs the `summarize` CLI in the sandbox container to run there.

Installer example:

```
---
name: gemini
description: Use Gemini CLI for coding assistance and Google search lookups.
metadata:
  {
    "openclaw":
      {
        "emoji": "♊",
        "requires": { "bins": ["gemini"] },
        "install":
          [
            {
              "id": "brew",
              "kind": "brew",
              "formula": "gemini-cli",
              "bins": ["gemini"],
              "label": "Install Gemini CLI (brew)",
            },
          ],
      },
  }
---
```

Notes:

If multiple installers are listed, the gateway picks a **single** preferred option (brew when available, otherwise node).

If all installers are `download`, OpenClaw lists each entry so you can see the available artifacts.

Installer specs can include `os: ["darwin"|"linux"|"win32"]` to filter options by platform.

Node installs honor `skills.install.nodeManager` in `openclaw.json` (default: npm; options: npm/pnpm/yarn/bun). This only affects **skill installs**; the Gateway runtime should still be Node (Bun is not recommended for WhatsApp/Telegram).

Go installs: if `go` is missing and `brew` is available, the gateway installs Go via Homebrew first and sets `GOBIN` to Homebrew's `bin` when possible.

Download installs: `url` (required), `archive` (`tar.gz` | `tar.bz2` | `zip`), `extract` (default: auto when archive detected), `stripComponents`, `targetDir` (default: `~/.openclaw/tools/<skillKey>`).

If no `metadata.openclaw` is present, the skill is always eligible (unless disabled in config or blocked by `skills.allowBundled` for bundled skills).

Config overrides (`~/.openclaw/openclaw.json`)

Bundled/managed skills can be toggled and supplied with env values:

```
skills: {
  entries: {
    "nano-banana-pro": {
      enabled: true,
      apiKey: "GEMINI_KEY_HERE",
      env: {
        GEMINI_API_KEY: "GEMINI_KEY_HERE",
      },
      config: {
        endpoint: "https://example.invalid",
        model: "nano-pro",
      },
    },
    peekaboo: { enabled: true },
    sag: { enabled: false },
  },
},
}
```

Note: if the skill name contains hyphens, quote the key (JSON5 allows quoted keys).

Config keys match the **skill name** by default. If a skill defines `metadata.openclaw.skillKey`, use that key under `skills.entries`.

Rules:

- `enabled: false` disables the skill even if it's bundled/installed.
- `env`: injected **only if** the variable isn't already set in the process.
- `apiKey`: convenience for skills that declare `metadata.openclaw.primaryEnv`.
- `config`: optional bag for custom per-skill fields; custom keys must live here.

`allowBundled` : optional allowlist for **bundled** skills only. If set, only bundled skills in the list are eligible (managed/workspace skills unaffected).

›

Environment injection (per agent run)

When an agent run starts, OpenClaw:

1. Reads skill metadata.
2. Applies any `skills.entries.<key>.env` or `skills.entries.<key>.apiKey` to `process.env` .
3. Builds the system prompt with **eligible** skills.
4. Restores the original environment after the run ends.

This is **scoped to the agent run**, not a global shell environment.

Session snapshot (performance)

OpenClaw snapshots the eligible skills **when a session starts** and reuses that list for subsequent turns in the same session. Changes to skills or config take effect on the next new session.

Skills can also refresh mid-session when the skills watcher is enabled or when a new eligible remote node appears (see below). Think of this as a **hot reload**: the refreshed list is picked up on the next agent turn.

Remote macOS nodes (Linux gateway)

If the Gateway is running on Linux but a **macOS node** is connected **with** `system.run` **allowed** (Exec approvals security not set to `deny`), OpenClaw can treat macOS-only skills as eligible when the required

binaries are present on that node. The agent should execute those skills via the `nodes` tool (typically `nodes.run`).

This relies on the node reporting its command support and on a bin probe via `system.run`. If the macOS node goes offline later, the skills remain visible; invocations may fail until the node reconnects.

Skills watcher (auto-refresh)

By default, OpenClaw watches skill folders and bumps the skills snapshot when `SKILL.md` files change. Configure this under `skills.load`:

```
{
  skills: {
    load: {
      watch: true,
      watchDebounceMs: 250,
    },
  },
}
```

Token impact (skills list)

When skills are eligible, OpenClaw injects a compact XML list of available skills into the system prompt (via `formatSkillsForPrompt` in `pi-coding-agent`). The cost is deterministic:

- **Base overhead (only when ≥1 skill):** 195 characters.
- **Per skill:** 97 characters + the length of the XML-escaped `<name>`, `<description>`, and `<location>` values.

Formula (characters):

```
total = 195 + Σ (97 + len(name_escaped) + len(description_escaped) + l
```

Notes:

- XML escaping expands `& < > " '` into entities (`&`, `<`, etc.), increasing length.
- Token counts vary by model tokenizer. A rough OpenAI-style estimate is ~4 chars/token, so **97 chars ≈ 24 tokens** per skill plus your actual field lengths.

Managed skills lifecycle

OpenClaw ships a baseline set of skills as **bundled skills** as part of the install (npm package or OpenClaw.app). `~/.openclaw/skills` exists for local overrides (for example, pinning/patching a skill without changing the bundled copy). Workspace skills are user-owned and override both on name conflicts.

Config reference

See **Skills config** for the full configuration schema.

Looking for more skills?

Browse **https://clawhub.com**.

‹ Slash Commands Skills Config ›