Overview

Tools

OpenClaw exposes **first-class agent tools** for browser, canvas, nodes, and cron. These replace the old `openclaw-*` skills: the tools are typed, no shelling, and the agent should rely on them directly.

Disabling tools

You can globally allow/deny tools via `tools.allow` / `tools.deny` in `openclaw.json` (deny wins). This prevents disallowed tools from being sent to model providers.

```
{
  tools: { deny: ["browser"] },
}
```

Notes:

- Matching is case-insensitive.
- `*` wildcards are supported (`"*"` means all tools).
- If `tools.allow` only references unknown or unloaded plugin tool names, OpenClaw logs a warning and ignores the allowlist so core tools stay available.

Tool profiles (base allowlist)

`tools.profile` sets a **base tool allowlist** before `tools.allow` / `tools.deny`. Per-agent override: `agents.list[].tools.profile`.

Profiles: ›

- `minimal`: `session_status` only
- `coding`: `group:fs`, `group:runtime`, `group:sessions`, `group:memory`, `image`
- `messaging`: `group:messaging`, `sessions_list`, `sessions_history`, `sessions_send`, `session_status`
- `full`: no restriction (same as unset)

Example (messaging-only by default, allow Slack + Discord tools too):

```
{
  tools: {
    profile: "messaging",
    allow: ["slack", "discord"],
  },
}
```

Example (coding profile, but deny exec/process everywhere):

```
{
  tools: {
    profile: "coding",
    deny: ["group:runtime"],
  },
}
```

Example (global coding profile, messaging-only support agent):

```
{
  tools: { profile: "coding" },
  agents: {
    list: [
      {
        id: "support",
        tools: { profile: "messaging", allow: ["slack"] },
      },
    ],
  },
}
```

Provider-specific tool policy

Use `tools.byProvider` to **further restrict** tools for specific providers (or a single `provider/model`) without changing your global defaults. Per-agent override: `agents.list[].tools.byProvider`.

This is applied **after** the base tool profile and **before** allow/deny lists, so it can only narrow the tool set. Provider keys accept either `provider` (e.g. `google-antigravity`) or `provider/model` (e.g. `openai/gpt-5.2`).

Example (keep global coding profile, but minimal tools for Google Antigravity):

```
{
  tools: {
    profile: "coding",
    byProvider: {
      "google-antigravity": { profile: "minimal" },
    },
  },
}
```

Example (provider/model-specific allowlist for a flaky endpoint):

```
{
  tools: {
    allow: ["group:fs", "group:runtime", "sessions_list"],
    byProvider: {
      "openai/gpt-5.2": { allow: ["group:fs", "sessions_list"] },
    },
  },
}
```

Example (agent-specific override for a single provider):

```
{
  agents: {
    list: [
      {
        id: "support",
        tools: {
          byProvider: {
            "google-antigravity": { allow: ["message", "sessions_list"] },
          },
        },
      },
    ],
  },
}
```

Tool groups (shorthands)

Tool policies (global, agent, sandbox) support `group:*` entries that expand to multiple tools. Use these in `tools.allow` / `tools.deny`.

Available groups:

`group:runtime`: `exec`, `bash`, `process`

```
group:fs : read , write , edit , apply_patch
group:sessions : sessions_list , sessions_history , sessions_send ,
sessions_spawn , session_status
group:memory : memory_search , memory_get
group:web : web_search , web_fetch
group:ui : browser , canvas
group:automation : cron , gateway
group:messaging : message
group:nodes : nodes
group:openclaw : all built-in OpenClaw tools (excludes provider
plugins)
```

Example (allow only file tools + browser):

```
{
  tools: {
    allow: ["group:fs", "browser"],
  },
}
```

Plugins + tools

Plugins can register **additional tools** (and CLI commands) beyond the core set. See for install + config, and for how tool usage guidance is injected into prompts. Some plugins ship their own skills alongside tools (for example, the voice-call plugin).

Optional plugin tools:

: typed workflow runtime with resumable approvals (requires the Lobster CLI on the gateway host).

LLM Task: JSON-only LLM step for structured workflow output (optional schema validation).

›

Tool inventory

apply_patch

Apply structured patches across one or more files. Use for multi-hunk edits. Experimental: enable via `tools.exec.applyPatch.enabled` (OpenAI models only). `tools.exec.applyPatch.workspaceOnly` defaults to `true` (workspace-contained). Set it to `false` only if you intentionally want `apply_patch` to write/delete outside the workspace directory.

exec

Run shell commands in the workspace.

Core parameters:

- `command` (required)
- `yieldMs` (auto-background after timeout, default 10000)
- `background` (immediate background)
- `timeout` (seconds; kills the process if exceeded, default 1800)
- `elevated` (bool; run on host if elevated mode is enabled/allowed; only changes behavior when the agent is sandboxed)
- `host` (`sandbox` | `gateway` | `node`)
- `security` (`deny` | `allowlist` | `full`)
- `ask` (`off` | `on-miss` | `always`)
- `node` (node id/name for `host=node`)

Need a real TTY? Set `pty: true` .

Notes:

Returns `status: "running"` with a `sessionId` when backgrounded.

Use `process` to poll/log/write/kill/clear background sessions.

If `process` is disallowed, `exec` runs synchronously and ignores `yieldMs` / `background`.

`elevated` is gated by `tools.elevated` plus any `agents.list[].tools.elevated` override (both must allow) and is an alias for `host=gateway` + `security=full`.

`elevated` only changes behavior when the agent is sandboxed (otherwise it's a no-op).

`host=node` can target a macOS companion app or a headless node host (`openclaw node run`).

gateway/node approvals and allowlists: **Exec approvals**.

process

Manage background exec sessions.

Core actions:

- `list`, `poll`, `log`, `write`, `kill`, `clear`, `remove`

Notes:

- `poll` returns new output and exit status when complete.
- `log` supports line-based `offset` / `limit` (omit `offset` to grab the last N lines).
- `process` is scoped per agent; sessions from other agents are not visible.

loop-detection (tool-call loop guardrails)

OpenClaw tracks recent tool-call history and blocks or warns when it detects repetitive no-progress loops. Enable with

`tools.loopDetection.enabled: true` (default is `false`).

```
{
  tools: {
    loopDetection: {
      enabled: true,
      warningThreshold: 10,
      criticalThreshold: 20,
      globalCircuitBreakerThreshold: 30,
      historySize: 30,
      detectors: {
        genericRepeat: true,
        knownPollNoProgress: true,
        pingPong: true,
      },
    },
  },
}
```

`genericRepeat`: repeated same tool + same params call pattern.

`knownPollNoProgress`: repeating poll-like tools with identical outputs.

`pingPong`: alternating `A/B/A/B` no-progress patterns.

Per-agent override: `agents.list[].tools.loopDetection`.

web_search

Search the web using Brave Search API.

Core parameters:

`query` (required)

`count` (1–10; default from `tools.web.search.maxResults`)

Notes:

- Requires a Brave API key (recommended: `openclaw configure --section web`, or set `BRAVE_API_KEY`).
- Enable via `tools.web.search.enabled`.
- Responses are cached (default 15 min).
- See **Web tools** for setup.

web_fetch

Fetch and extract readable content from a URL (HTML → markdown/text).

Core parameters:

- `url` (required)
- `extractMode` (`markdown` | `text`)
- `maxChars` (truncate long pages)

Notes:

- Enable via `tools.web.fetch.enabled`.
- `maxChars` is clamped by `tools.web.fetch.maxCharsCap` (default 50000).
- Responses are cached (default 15 min).
- For JS-heavy sites, prefer the browser tool.
- See **Web tools** for setup.
- See **Firecrawl** for the optional anti-bot fallback.

browser

Control the dedicated OpenClaw-managed browser.

Core actions:

- `status`, `start`, `stop`, `tabs`, `open`, `focus`, `close`
- `snapshot` (aria/ai)

- `screenshot` (returns image block + `MEDIA:<path>`)
- `act` (UI actions: click/type/press/hover/drag/select/fill/resize/wait/evaluate)
- `navigate`, `console`, `pdf`, `upload`, `dialog`

Profile management:

- `profiles` — list all browser profiles with status
- `create-profile` — create new profile with auto-allocated port (or `cdpUrl`)
- `delete-profile` — stop browser, delete user data, remove from config (local only)
- `reset-profile` — kill orphan process on profile's port (local only)

Common parameters:

- `profile` (optional; defaults to `browser.defaultProfile`)
- `target` (`sandbox` | `host` | `node`)
- `node` (optional; picks a specific node id/name) Notes:
  - Requires `browser.enabled=true` (default is `true`; set `false` to disable).
  - All actions accept optional `profile` parameter for multi-instance support.
  - When `profile` is omitted, uses `browser.defaultProfile` (defaults to "chrome").
  - Profile names: lowercase alphanumeric + hyphens only (max 64 chars).
  - Port range: 18800-18899 (~100 profiles max).
  - Remote profiles are attach-only (no start/stop/reset).
  - If a browser-capable node is connected, the tool may auto-route to it (unless you pin `target`).

`snapshot` defaults to `ai` when Playwright is installed; use `aria` for the accessibility tree.

`snapshot` also supports role-snapshot options (`interactive`, `compact`, `depth`, `selector`) which return refs like `e12`.

`act` requires `ref` from `snapshot` (numeric `12` from AI snapshots, or `e12` from role snapshots); use `evaluate` for rare CSS selector needs.

Avoid `act` → `wait` by default; use it only in exceptional cases (no reliable UI state to wait on).

`upload` can optionally pass a `ref` to auto-click after arming.

`upload` also supports `inputRef` (aria ref) or `element` (CSS selector) to set `<input type="file">` directly.

canvas

Drive the node Canvas (present, eval, snapshot, A2UI).

Core actions:

- `present`, `hide`, `navigate`, `eval`
- `snapshot` (returns image block + `MEDIA:<path>`)
- `a2ui_push`, `a2ui_reset`

Notes:

- Uses gateway `node.invoke` under the hood.
- If no `node` is provided, the tool picks a default (single connected node or local mac node).
- A2UI is v0.8 only (no `createSurface`); the CLI rejects v0.9 JSONL with line errors.
- Quick smoke: `openclaw nodes canvas a2ui push --node <id> --text "Hello from A2UI"`.

nodes

Discover and target paired nodes; send notifications; capture camera/screen.

Core actions:

- `status`, `describe`
- `pending`, `approve`, `reject` (pairing)
- `notify` (macOS `system.notify`)
- `run` (macOS `system.run`)
- `camera_snap`, `camera_clip`, `screen_record`
- `location_get`

Notes:

- Camera/screen commands require the node app to be foregrounded.
- Images return image blocks + `MEDIA:<path>`.
- Videos return `FILE:<path>` (mp4).
- Location returns a JSON payload (lat/lon/accuracy/timestamp).
- `run` params: `command` argv array; optional `cwd`, `env` (`KEY=VAL`), `commandTimeoutMs`, `invokeTimeoutMs`, `needsScreenRecording`.

Example (`run`):

```
{
  "action": "run",
  "node": "office-mac",
  "command": ["echo", "Hello"],
  "env": ["FOO=bar"],
  "commandTimeoutMs": 12000,
  "invokeTimeoutMs": 45000,
  "needsScreenRecording": false
}
```

image

Analyze an image with the configured image model.

Core parameters:

- `image` (required path or URL)
- `prompt` (optional; defaults to “Describe the image.”)
- `model` (optional override)
- `maxBytesMb` (optional size cap)

Notes:

- Only available when `agents.defaults.imageModel` is configured (primary or fallbacks), or when an implicit image model can be inferred from your default model + configured auth (best-effort pairing).
- Uses the image model directly (independent of the main chat model).

message

Send messages and channel actions across Discord/Google Chat/Slack/Telegram/WhatsApp/Signal/iMessage/MS Teams.

Core actions:

- `send` (text + optional media; MS Teams also supports `card` for Adaptive Cards)
- `poll` (WhatsApp/Discord/MS Teams polls)
- `react` / `reactions` / `read` / `edit` / `delete`
- `pin` / `unpin` / `list-pins`
- `permissions`
- `thread-create` / `thread-list` / `thread-reply`
- `search`
- `sticker`

`member-info` / `role-info`

`emoji-list` / `emoji-upload` / `sticker-upload`

`role-add` / `role-remove`

`channel-info` / `channel-list`

`voice-status`

`event-list` / `event-create`

`timeout` / `kick` / `ban`

Notes:

- `send` routes WhatsApp via the Gateway; other channels go direct.
- `poll` uses the Gateway for WhatsApp and MS Teams; Discord polls go direct.

  When a message tool call is bound to an active chat session, sends are constrained to that session's target to avoid cross-context leaks.

cron

Manage Gateway cron jobs and wakeups.

Core actions:

- `status` , `list`
- `add` , `update` , `remove` , `run` , `runs`
- `wake` (enqueue system event + optional immediate heartbeat)

Notes:

- `add` expects a full cron job object (same schema as `cron.add` RPC).
- `update` uses `{ jobId, patch }` (`id` accepted for compatibility).

gateway

Restart or apply updates to the running Gateway process (in-place).

Core actions:

- `restart` (authorizes + sends `SIGUSR1` for in-process restart; `openclaw gateway` restart in-place)
- `config.get` / `config.schema`
- `config.apply` (validate + write config + restart + wake)
- `config.patch` (merge partial update + restart + wake)
- `update.run` (run update + restart + wake)

Notes:

- Use `delayMs` (defaults to 2000) to avoid interrupting an in-flight reply.
- `restart` is disabled by default; enable with `commands.restart: true`.

sessions_list / sessions_history / sessions_send / sessions_spawn / session_status

List sessions, inspect transcript history, or send to another session.

Core parameters:

- `sessions_list`: `kinds?`, `limit?`, `activeMinutes?`, `messageLimit?` (0 = none)
- `sessions_history`: `sessionKey` (or `sessionId`), `limit?`, `includeTools?`
- `sessions_send`: `sessionKey` (or `sessionId`), `message`, `timeoutSeconds?` (0 = fire-and-forget)
- `sessions_spawn`: `task`, `label?`, `agentId?`, `model?`, `runTimeoutSeconds?`, `cleanup?`

`session_status` : `sessionKey?` (default current; accepts `sessionId`), `model?` (`default` clears override)

Notes: ›

`main` is the canonical direct-chat key; global/unknown are hidden.

`messageLimit > 0` fetches last N messages per session (tool messages filtered).

Session targeting is controlled by `tools.sessions.visibility` (default `tree` : current session + spawned subagent sessions). If you run a shared agent for multiple users, consider setting `tools.sessions.visibility: "self"` to prevent cross-session browsing.

`sessions_send` waits for final completion when `timeoutSeconds > 0` .

Delivery/announce happens after completion and is best-effort; `status: "ok"` confirms the agent run finished, not that the announce was delivered.

`sessions_spawn` starts a sub-agent run and posts an announce reply back to the requester chat.

Reply format includes `Status` , `Result` , and compact stats.

`Result` is the assistant completion text; if missing, the latest `toolResult` is used as fallback.

Manual completion-mode spawns send directly first, with queue fallback and retry on transient failures (`status: "ok"` means run finished, not that announce delivered).

`sessions_spawn` is non-blocking and returns `status: "accepted"` immediately.

`sessions_send` runs a reply-back ping-pong (reply `REPLY_SKIP` to stop; max turns via `session.agentToAgent.maxPingPongTurns` , 0–5).

After the ping-pong, the target agent runs an **announce step**; reply `ANNOUNCE_SKIP` to suppress the announcement.

Sandbox clamp: when the current session is sandboxed and `agents.defaults.sandbox.sessionToolsVisibility: "spawned"`, OpenClaw clamps `tools.sessions.visibility` to `tree`.

›

agents_list

List agent ids that the current session may target with `sessions_spawn`.

Notes:

- Result is restricted to per-agent allowlists (`agents.list[].subagents.allowAgents`).
- When `["*"]` is configured, the tool includes all configured agents and marks `allowAny: true`.

Parameters (common)

Gateway-backed tools (`canvas`, `nodes`, `cron`):

- `gatewayUrl` (default `ws://127.0.0.1:18789`)
- `gatewayToken` (if auth enabled)
- `timeoutMs`

Note: when `gatewayUrl` is set, include `gatewayToken` explicitly. Tools do not inherit config or environment credentials for overrides, and missing explicit credentials is an error.

Browser tool:

- `profile` (optional; defaults to `browser.defaultProfile`)
- `target` (`sandbox` | `host` | `node`)
- `node` (optional; pin a specific node id/name)

Recommended agent flows

Browser automation:

1. `browser` → `status` / `start`
2. `snapshot` (ai or aria)
3. `act` (click/type/press)
4. `screenshot` if you need visual confirmation

Canvas render:

1. `canvas` → `present`
2. `a2ui_push` (optional)
3. `snapshot`

Node targeting:

1. `nodes` → `status`
2. `describe` on the chosen node
3. `notify` / `run` / `camera_snap` / `screen_record`

Safety

- Avoid direct `system.run`; use `nodes` → `run` only with explicit user consent.
- Respect user consent for camera/screen capture.
- Use `status/describe` to ensure permissions before invoking media commands.

How tools are presented to the agent

Tools are exposed in two parallel channels:

1. **System prompt text**: a human-readable list + guidance.
2. **Tool schema**: the structured function definitions sent to the model API.

That means the agent sees both “what tools exist” and “how to call them.” If a tool doesn’t appear in the system prompt or the schema, the model cannot call it.

Lobster ›