Automation

Cron Jobs

> **Cron vs Heartbeat?** See **Cron vs Heartbeat** for guidance on when to use each.

Cron is the Gateway's built-in scheduler. It persists jobs, wakes the agent at the right time, and can optionally deliver output back to a chat.

If you want *"run this every morning"* or *"poke the agent in 20 minutes"*, cron is the mechanism.

Troubleshooting: **/automation/troubleshooting**

TL;DR

- Cron runs **inside the Gateway** (not inside the model).
- Jobs persist under `~/.openclaw/cron/` so restarts don't lose schedules.
- Two execution styles:
  - **Main session**: enqueue a system event, then run on the next heartbeat.
  - **Isolated**: run a dedicated agent turn in `cron:<jobId>`, with delivery (announce by default or none).
- Wakeups are first-class: a job can request "wake now" vs "next heartbeat".

Webhook posting is per job via `delivery.mode = "webhook"` + `delivery.to = "<url>"`.

Legacy fallback remains for stored jobs with `notify: true` when `cron.webhook` is set, migrate those jobs to webhook delivery mode.

Quick start (actionable)

Create a one-shot reminder, verify it exists, and run it immediately:

```
openclaw cron add \
  --name "Reminder" \
  --at "2026-02-01T16:00:00Z" \
  --session main \
  --system-event "Reminder: check the cron docs draft" \
  --wake now \
  --delete-after-run

openclaw cron list
openclaw cron run <job-id>
openclaw cron runs --id <job-id>
```

Schedule a recurring isolated job with delivery:

```
openclaw cron add \
  --name "Morning brief" \
  --cron "0 7 * * *" \
  --tz "America/Los_Angeles" \
  --session isolated \
  --message "Summarize overnight updates." \
  --announce \
  --channel slack \
  --to "channel:C1234567890"
```

Tool-call equivalents (Gateway cron tool)

For the canonical JSON shapes and examples, see **JSON schema for tool calls**.

Where cron jobs are stored

Cron jobs are persisted on the Gateway host at `~/.openclaw/cron/jobs.json` by default. The Gateway loads the file into memory and writes it back on changes, so manual edits are only safe when the Gateway is stopped. Prefer `openclaw cron add/edit` or the cron tool call API for changes.

Beginner-friendly overview

Think of a cron job as: **when** to run + **what** to do.

1. **Choose a schedule**

   One-shot reminder → `schedule.kind = "at"` (CLI: `--at`)

   Repeating job → `schedule.kind = "every"` or `schedule.kind = "cron"`

   If your ISO timestamp omits a timezone, it is treated as **UTC**.

2. **Choose where it runs**

   `sessionTarget: "main"` → run during the next heartbeat with main context.

   `sessionTarget: "isolated"` → run a dedicated agent turn in `cron:<jobId>`.

3. **Choose the payload**

   Main session → `payload.kind = "systemEvent"`

   Isolated session → `payload.kind = "agentTurn"`

Optional: one-shot jobs (`schedule.kind = "at"`) delete after success by default. Set `deleteAfterRun: false` to keep them (they will disable after success).

Concepts

Jobs

A cron job is a stored record with:

- a **schedule** (when it should run),
- a **payload** (what it should do),
- optional **delivery mode** (`announce` , `webhook` , or `none`).
- optional **agent binding** (`agentId`): run the job under a specific agent; if missing or unknown, the gateway falls back to the default agent.

Jobs are identified by a stable `jobId` (used by CLI/Gateway APIs). In agent tool calls, `jobId` is canonical; legacy `id` is accepted for compatibility. One-shot jobs auto-delete after success by default; set `deleteAfterRun: false` to keep them.

Schedules

Cron supports three schedule kinds:

- `at` : one-shot timestamp via `schedule.at` (ISO 8601).
- `every` : fixed interval (ms).
- `cron` : 5-field cron expression (or 6-field with seconds) with optional IANA timezone.

Cron expressions use `croner` . If a timezone is omitted, the Gateway host's local timezone is used.

To reduce top-of-hour load spikes across many gateways, OpenClaw applies a deterministic per-job stagger window of up to 5 minutes for recurring top-of-hour expressions (for example `0 * * * *` , `0 */2 * * *`). Fixed-hour expressions such as `0 7 * * *` remain exact.

For any cron schedule, you can set an explicit stagger window with `schedule.staggerMs` (`0` keeps exact timing). CLI shortcuts:

- `--stagger 30s` (or `1m`, `5m`) to set an explicit stagger window.
- `--exact` to force `staggerMs = 0`.

Main vs isolated execution

Main session jobs (system events)

Main jobs enqueue a system event and optionally wake the heartbeat runner. They must use `payload.kind = "systemEvent"`.

- `wakeMode: "now"` (default): event triggers an immediate heartbeat run.
- `wakeMode: "next-heartbeat"`: event waits for the next scheduled heartbeat.

This is the best fit when you want the normal heartbeat prompt + main-session context. See **Heartbeat**.

Isolated jobs (dedicated cron sessions)

Isolated jobs run a dedicated agent turn in session `cron:<jobId>`.

Key behaviors:

- Prompt is prefixed with `[cron:<jobId> <job name>]` for traceability.
- Each run starts a **fresh session id** (no prior conversation carry-over).
- Default behavior: if `delivery` is omitted, isolated jobs announce a summary (`delivery.mode = "announce"`).
- `delivery.mode` chooses what happens:
  - `announce`: deliver a summary to the target channel and post a brief summary to the main session.

`webhook` : POST the finished event payload to `delivery.to` when the finished event includes a summary.

`none` : internal only (no delivery, no main-session summary).

`wakeMode` controls when the main-session summary posts:

`now` : immediate heartbeat.

`next-heartbeat` : waits for the next scheduled heartbeat.

Use isolated jobs for noisy, frequent, or “background chores” that shouldn’t spam your main chat history.

Payload shapes (what runs)

Two payload kinds are supported:

`systemEvent` : main-session only, routed through the heartbeat prompt.

`agentTurn` : isolated-session only, runs a dedicated agent turn.

Common `agentTurn` fields:

`message` : required text prompt.

`model` / `thinking` : optional overrides (see below).

`timeoutSeconds` : optional timeout override.

Delivery config:

`delivery.mode` : `none` | `announce` | `webhook` .

`delivery.channel` : `last` or a specific channel.

`delivery.to` : channel-specific target (announce) or webhook URL (webhook mode).

`delivery.bestEffort` : avoid failing the job if announce delivery fails.

Announce delivery suppresses messaging tool sends for the run; use `delivery.channel` / `delivery.to` to target the chat instead. When `delivery.mode = "none"`, no summary is posted to the main session.

If `delivery` is omitted for isolated jobs, OpenClaw defaults to `announce`.

Announce delivery flow

When `delivery.mode = "announce"`, cron delivers directly via the outbound channel adapters. The main agent is not spun up to craft or forward the message.

Behavior details:

- Content: delivery uses the isolated run's outbound payloads (text/media) with normal chunking and channel formatting.
- Heartbeat-only responses (`HEARTBEAT_OK` with no real content) are not delivered.
- If the isolated run already sent a message to the same target via the message tool, delivery is skipped to avoid duplicates.
- Missing or invalid delivery targets fail the job unless `delivery.bestEffort = true`.
- A short summary is posted to the main session only when `delivery.mode = "announce"`.
- The main-session summary respects `wakeMode`: `now` triggers an immediate heartbeat and `next-heartbeat` waits for the next scheduled heartbeat.

Webhook delivery flow

When `delivery.mode = "webhook"`, cron posts the finished event payload to `delivery.to` when the finished event includes a summary.

Behavior details:

The endpoint must be a valid HTTP(S) URL.

No channel delivery is attempted in webhook mode.

No main-session summary is posted in webhook mode.

If `cron.webhookToken` is set, auth header is `Authorization: Bearer <cron.webhookToken>`.

Deprecated fallback: stored legacy jobs with `notify: true` still post to `cron.webhook` (if configured), with a warning so you can migrate to `delivery.mode = "webhook"`.

Model and thinking overrides

Isolated jobs (`agentTurn`) can override the model and thinking level:

`model`: Provider/model string (e.g., `anthropic/claude-sonnet-4-20250514`) or alias (e.g., `opus`)

`thinking`: Thinking level (`off`, `minimal`, `low`, `medium`, `high`, `xhigh`; GPT-5.2 + Codex models only)

Note: You can set `model` on main-session jobs too, but it changes the shared main session model. We recommend model overrides only for isolated jobs to avoid unexpected context shifts.

Resolution priority:

1. Job payload override (highest)
2. Hook-specific defaults (e.g., `hooks.gmail.model`)
3. Agent config default

Delivery (channel + target)

Isolated jobs can deliver output to a channel via the top-level `delivery` config:

- `delivery.mode` : `announce` (channel delivery), `webhook` (HTTP POST), or `none` .
- `delivery.channel` : `whatsapp` / `telegram` / `discord` / `slack` / `mattermost` (plugin) / `signal` / `imessage` / `last` .
- `delivery.to` : channel-specific recipient target.

`announce` delivery is only valid for isolated jobs (`sessionTarget: "isolated"`). `webhook` delivery is valid for both main and isolated jobs.

If `delivery.channel` or `delivery.to` is omitted, cron can fall back to the main session's "last route" (the last place the agent replied).

Target format reminders:

- Slack/Discord/Mattermost (plugin) targets should use explicit prefixes (e.g. `channel:<id>` , `user:<id>`) to avoid ambiguity.
- Telegram topics should use the `:topic:` form (see below).

Telegram delivery targets (topics / forum threads)

Telegram supports forum topics via `message_thread_id` . For cron delivery, you can encode the topic/thread into the `to` field:

- `-1001234567890` (chat id only)
- `-1001234567890:topic:123` (preferred: explicit topic marker)
- `-1001234567890:123` (shorthand: numeric suffix)

Prefixed targets like `telegram:...` / `telegram:group:...` are also accepted:

- `telegram:group:-1001234567890:topic:123`

JSON schema for tool calls

Use these shapes when calling Gateway `cron.*` tools directly (agent tool calls or RPC). CLI flags accept human durations like `20m`, but tool calls should use an ISO 8601 string for `schedule.at` and milliseconds for `schedule.everyMs`.

cron.add params

One-shot, main session job (system event):

```
{
  "name": "Reminder",
  "schedule": { "kind": "at", "at": "2026-02-01T16:00:00Z" },
  "sessionTarget": "main",
  "wakeMode": "now",
  "payload": { "kind": "systemEvent", "text": "Reminder text" },
  "deleteAfterRun": true
}
```

Recurring, isolated job with delivery:

```
{
  "name": "Morning brief",
  "schedule": { "kind": "cron", "expr": "0 7 * * *", "tz": "America/Los_Angeles" }
  "sessionTarget": "isolated",
  "wakeMode": "next-heartbeat",
  "payload": {
    "kind": "agentTurn",
    "message": "Summarize overnight updates."
  },
  "delivery": {
    "mode": "announce",
    "channel": "slack",
    "to": "channel:C1234567890",
    "bestEffort": true
  }
}
```

Notes:

- `schedule.kind`: `at` (`at`), `every` (`everyMs`), or `cron` (`expr`, optional `tz`).
- `schedule.at` accepts ISO 8601 (timezone optional; treated as UTC when omitted).
- `everyMs` is milliseconds.
- `sessionTarget` must be `"main"` or `"isolated"` and must match `payload.kind`.
- Optional fields: `agentId`, `description`, `enabled`, `deleteAfterRun` (defaults to true for `at`), `delivery`.
- `wakeMode` defaults to `"now"` when omitted.

cron.update params

```
{
  "jobId": "job-123",
  "patch": {
    "enabled": false,
    "schedule": { "kind": "every", "everyMs": 3600000 }
  }
}
```

Notes:

- `jobId` is canonical; `id` is accepted for compatibility.
- Use `agentId: null` in the patch to clear an agent binding.

cron.run and cron.remove params

```
{ "jobId": "job-123", "mode": "force" }
```

```
{ "jobId": "job-123" }
```

Storage & history

Job store: `~/.openclaw/cron/jobs.json` (Gateway-managed JSON).

Run history: `~/.openclaw/cron/runs/<jobId>.jsonl` (JSONL, auto-pruned).

Override store path: `cron.store` in config.

Configuration

```
{
  cron: {
    enabled: true, // default true
    store: "~/.openclaw/cron/jobs.json",
    maxConcurrentRuns: 1, // default 1
    webhook: "https://example.invalid/legacy", // deprecated fallback for stored r
    webhookToken: "replace-with-dedicated-webhook-token", // optional bearer toker
  },
}
```

Webhook behavior:

Preferred: set `delivery.mode: "webhook"` with `delivery.to: "https://..."` per job.

Webhook URLs must be valid `http://` or `https://` URLs.

When posted, payload is the cron finished event JSON.

If `cron.webhookToken` is set, auth header is `Authorization: Bearer <cron.webhookToken>`.

If `cron.webhookToken` is not set, no `Authorization` header is sent.

Deprecated fallback: stored legacy jobs with `notify: true` still use `cron.webhook` when present.

Disable cron entirely:

`cron.enabled: false` (config)

`OPENCLAW_SKIP_CRON=1` (env)

CLI quickstart

One-shot reminder (UTC ISO, auto-delete after success):

```
openclaw cron add \
  --name "Send reminder" \
  --at "2026-01-12T18:00:00Z" \
  --session main \
  --system-event "Reminder: submit expense report." \
  --wake now \
  --delete-after-run
```

One-shot reminder (main session, wake immediately):

```
openclaw cron add \
  --name "Calendar check" \
  --at "20m" \
  --session main \
  --system-event "Next heartbeat: check calendar." \
  --wake now
```

Recurring isolated job (announce to WhatsApp):

```
openclaw cron add \
  --name "Morning status" \
  --cron "0 7 * * *" \
  --tz "America/Los_Angeles" \
  --session isolated \
  --message "Summarize inbox + calendar for today." \
  --announce \
  --channel whatsapp \
  --to "+15551234567"
```

Recurring cron job with explicit 30-second stagger:

```
openclaw cron add \
  --name "Minute watcher" \
  --cron "0 * * * * *" \
  --tz "UTC" \
  --stagger 30s \
  --session isolated \
  --message "Run minute watcher checks." \
  --announce
```

Recurring isolated job (deliver to a Telegram topic):

```
openclaw cron add \
  --name "Nightly summary (topic)" \
  --cron "0 22 * * *" \
  --tz "America/Los_Angeles" \
  --session isolated \
  --message "Summarize today; send to the nightly topic." \
  --announce \
  --channel telegram \
  --to "-1001234567890:topic:123"
```

Isolated job with model and thinking override:

```
openclaw cron add \
  --name "Deep analysis" \
  --cron "0 6 * * 1" \
  --tz "America/Los_Angeles" \
  --session isolated \
  --message "Weekly deep analysis of project progress." \
  --model "opus" \
  --thinking high \
  --announce \
  --channel whatsapp \
  --to "+15551234567"
```

Agent selection (multi-agent setups):

```
# Pin a job to agent "ops" (falls back to default if that agent is mis
openclaw cron add --name "Ops sweep" --cron "0 6 * * *" --session isolated --messa

# Switch or clear the agent on an existing job
openclaw cron edit <jobId> --agent ops
openclaw cron edit <jobId> --clear-agent
```

Manual run (force is the default, use `--due` to only run when due):

```
openclaw cron run <jobId>
openclaw cron run <jobId> --due
```

Edit an existing job (patch fields):

```
openclaw cron edit <jobId> \
  --message "Updated prompt" \
  --model "opus" \
  --thinking low
```

Force an existing cron job to run exactly on schedule (no stagger):

```
openclaw cron edit <jobId> --exact
```

Run history:

```
openclaw cron runs --id <jobId> --limit 50
```

Immediate system event without creating a job:

```
openclaw system event --mode now --text "Next heartbeat: check battery
```

Gateway API surface

`cron.list`, `cron.status`, `cron.add`, `cron.update`, `cron.remove`

`cron.run` (force or due), `cron.runs` For immediate system events without a job, use .

Troubleshooting

“Nothing runs”

Check cron is enabled: `cron.enabled` and `OPENCLAW_SKIP_CRON`.

Check the Gateway is running continuously (cron runs inside the Gateway process).

For `cron` schedules: confirm timezone (`--tz`) vs the host timezone.

A recurring job keeps delaying after failures

OpenClaw applies exponential retry backoff for recurring jobs after consecutive errors: 30s, 1m, 5m, 15m, then 60m between retries.

Backoff resets automatically after the next successful run.

One-shot (`at`) jobs disable after a terminal run (`ok` , `error` , or `skipped`) and do not retry.

›

Telegram delivers to the wrong place

For forum topics, use `-100…:topic:<id>` so it's explicit and unambiguous.

If you see `telegram:...` prefixes in logs or stored "last route" targets, that's normal; cron delivery accepts them and still parses topic IDs correctly.

Subagent announce delivery retries

When a subagent run completes, the gateway announces the result to the requester session.

If the announce flow returns `false` (e.g. requester session is busy), the gateway retries up to 3 times with tracking via `announceRetryCount` .

Announces older than 5 minutes past `endedAt` are force-expired to prevent stale entries from looping indefinitely.

If you see repeated announce deliveries in logs, check the subagent registry for entries with high `announceRetryCount` values.

‹ Hooks Cron vs Heartbeat ›