Automation

Hooks

Hooks provide an extensible event-driven system for automating actions in response to agent commands and events. Hooks are automatically discovered from directories and can be managed via CLI commands, similar to how skills work in OpenClaw.

Getting Oriented

Hooks are small scripts that run when something happens. There are two kinds:

- **Hooks** (this page): run inside the Gateway when agent events fire, like `/new`, `/reset`, `/stop`, or lifecycle events.
- **Webhooks**: external HTTP webhooks that let other systems trigger work in OpenClaw. See **Webhook Hooks** or use `openclaw webhooks` for Gmail helper commands.

Hooks can also be bundled inside plugins; see **Plugins**.

Common uses:

- Save a memory snapshot when you reset a session
- Keep an audit trail of commands for troubleshooting or compliance
- Trigger follow-up automation when a session starts or ends
- Write files into the agent workspace or call external APIs when events fire

If you can write a small TypeScript function, you can write a hook. Hooks are discovered automatically, and you enable or disable them via the CLI.

›

Overview

The hooks system allows you to:

- Save session context to memory when `/new` is issued
- Log all commands for auditing
- Trigger custom automations on agent lifecycle events
- Extend OpenClaw's behavior without modifying core code

Getting Started

Bundled Hooks

OpenClaw ships with four bundled hooks that are automatically discovered:

- 💾 **session-memory**: Saves session context to your agent workspace (default `~/.openclaw/workspace/memory/`) when you issue `/new`
- 📎 **bootstrap-extra-files**: Injects additional workspace bootstrap files from configured glob/path patterns during `agent:bootstrap`
- 📝 **command-logger**: Logs all command events to `~/.openclaw/logs/commands.log`
- 🚀 **boot-md**: Runs `BOOT.md` when the gateway starts (requires internal hooks enabled)

List available hooks:

```
openclaw hooks list
```

Enable a hook:

```
openclaw hooks enable session-memory
                ›
```

Check hook status:

```
openclaw hooks check
```

Get detailed information:

```
openclaw hooks info session-memory
```

Onboarding

During onboarding (`openclaw onboard`), you'll be prompted to enable recommended hooks. The wizard automatically discovers eligible hooks and presents them for selection.

Hook Discovery

Hooks are automatically discovered from three directories (in order of precedence):

1. **Workspace hooks**: `<workspace>/hooks/` (per-agent, highest precedence)
2. **Managed hooks**: `~/.openclaw/hooks/` (user-installed, shared across workspaces)
3. **Bundled hooks**: `<openclaw>/dist/hooks/bundled/` (shipped with OpenClaw)

Managed hook directories can be either a **single hook** or a **hook pack** (package directory).

Each hook is a directory containing:

```
my-hook/
├── HOOK.md          # Metadata + documentation
└── handler.ts       # Handler implementation
```

Hook Packs (npm/archives)

Hook packs are standard npm packages that export one or more hooks via `openclaw.hooks` in `package.json`. Install them with:

```
openclaw hooks install <path-or-spec>
```

Npm specs are registry-only (package name + optional version/tag). Git/URL/file specs are rejected.

Example `package.json`:

```
{
  "name": "@acme/my-hooks",
  "version": "0.1.0",
  "openclaw": {
    "hooks": ["./hooks/my-hook", "./hooks/other-hook"]
  }
}
```

Each entry points to a hook directory containing `HOOK.md` and `handler.ts` (or `index.ts`). Hook packs can ship dependencies; they will be installed under `~/.openclaw/hooks/<id>`.

Security note: `openclaw hooks install` installs dependencies with `npm install --ignore-scripts` (no lifecycle scripts). Keep hook pack dependency trees “pure JS/TS” and avoid packages that rely on `postinstall` builds.

Hook Structure

HOOK.md Format

The `HOOK.md` file contains metadata in YAML frontmatter plus Markdown documentation:

```
---
name: my-hook
description: "Short description of what this hook does"
homepage: https://docs.openclaw.ai/automation/hooks#my-hook
metadata:
  { "openclaw": { "emoji": "🔗", "events": ["command:new"], "requires": { "bins":
---

# My Hook

Detailed documentation goes here...

## What It Does

- Listens for `/new` commands
- Performs some action
- Logs the result

## Requirements

- Node.js must be installed

## Configuration

No configuration needed.
```

Metadata Fields

The `metadata.openclaw` object supports:

- `emoji` : Display emoji for CLI (e.g., `"💾"`)

- `events` : Array of events to listen for (e.g., `["command:new", "command:reset"]`)
- `export` : Named export to use (defaults to `"default"`)
- `homepage` : Documentation URL
- `requires` : Optional requirements
  - `bins` : Required binaries on PATH (e.g., `["git", "node"]`)
  - `anyBins` : At least one of these binaries must be present
  - `env` : Required environment variables
  - `config` : Required config paths (e.g., `["workspace.dir"]`)
  - `os` : Required platforms (e.g., `["darwin", "linux"]`)
- `always` : Bypass eligibility checks (boolean)
- `install` : Installation methods (for bundled hooks: `[{"id":"bundled","kind":"bundled"}]`)

Handler Implementation

The `handler.ts` file exports a `HookHandler` function:

```typescript
import type { HookHandler } from "../../src/hooks/hooks.js";

const myHandler: HookHandler = async (event) => {
  // Only trigger on 'new' command
  if (event.type !== "command" || event.action !== "new") {
    return;
  }

  console.log(`[my-hook] New command triggered`);
  console.log(`  Session: ${event.sessionKey}`);
  console.log(`  Timestamp: ${event.timestamp.toISOString()}`);

  // Your custom logic here

  // Optionally send message to user
  event.messages.push("✨ My hook executed!");
};

export default myHandler;
```

Event Context

Each event includes:

```
  type: 'command' | 'session' | 'agent' | 'gateway' | 'message',
  action: string,              // e.g., 'new', 'reset', 'stop', 'received', 'sent
  sessionKey: string,          // Session identifier
  timestamp: Date,             // When the event occurred
  messages: string[],          // Push messages here to send to user
  context: {
    // Command events:
    sessionEntry?: SessionEntry,
    sessionId?: string,
    sessionFile?: string,
    commandSource?: string,    // e.g., 'whatsapp', 'telegram'
    senderId?: string,
    workspaceDir?: string,
    bootstrapFiles?: WorkspaceBootstrapFile[],
    cfg?: OpenClawConfig,
    // Message events (see Message Events section for full details):
    from?: string,             // message:received
    to?: string,               // message:sent
    content?: string,
    channelId?: string,
    success?: boolean,         // message:sent
  }
}
```

Event Types

Command Events

Triggered when agent commands are issued:

- **`command`** : All command events (general listener)
- **`command:new`** : When **`/new`** command is issued
- **`command:reset`** : When **`/reset`** command is issued
- **`command:stop`** : When **`/stop`** command is issued

Agent Events

- `agent:bootstrap` : Before workspace bootstrap files are injected (hooks may mutate `context.bootstrapFiles`)

Gateway Events

Triggered when the gateway starts:

- `gateway:startup` : After channels start and hooks are loaded

Message Events

Triggered when messages are received or sent:

- `message` : All message events (general listener)
- `message:received` : When an inbound message is received from any channel
- `message:sent` : When an outbound message is successfully sent

Message Event Context

Message events include rich context about the message:

```
// message:received context
{
  from: string,           // Sender identifier (phone number, user ID, etc.)
  content: string,        // Message content
  timestamp?: number,     // Unix timestamp when received
  channelId: string,      // Channel (e.g., "whatsapp", "telegram", "discord")
  accountId?: string,     // Provider account ID for multi-account setups
  conversationId?: string, // Chat/conversation ID
  messageId?: string,     // Message ID from the provider
  metadata?: {            // Additional provider-specific data
    to?: string,
    provider?: string,
    surface?: string,
    threadId?: string,
    senderId?: string,
    senderName?: string,
    senderUsername?: string,
    senderE164?: string,
  }
}

// message:sent context
{
  to: string,             // Recipient identifier
  content: string,        // Message content that was sent
  success: boolean,       // Whether the send succeeded
  error?: string,         // Error message if sending failed
  channelId: string,      // Channel (e.g., "whatsapp", "telegram", "discord")
  accountId?: string,     // Provider account ID
  conversationId?: string, // Chat/conversation ID
  messageId?: string,     // Message ID returned by the provider
}
```

Example: Message Logger Hook

```
import type { HookHandler } from "../../src/hooks/hooks.js";
import { isMessageReceivedEvent, isMessageSentEvent } from "../../src/hooks/inter

const handler: HookHandler = async (event) => {
  if (isMessageReceivedEvent(event)) {
    console.log(`[message-logger] Received from ${event.context.from}: ${event.co
  } else if (isMessageSentEvent(event)) {
    console.log(`[message-logger] Sent to ${event.context.to}: ${event.context.co
  }
};

export default handler;
```

Tool Result Hooks (Plugin API)

These hooks are not event-stream listeners; they let plugins synchronously adjust tool results before OpenClaw persists them.

- `tool_result_persist` : transform tool results before they are written to the session transcript. Must be synchronous; return the updated tool result payload or `undefined` to keep it as-is. See .

Future Events

Planned event types:

- `session:start` : When a new session begins
- `session:end` : When a session ends
- `agent:error` : When an agent encounters an error

Creating Custom Hooks

1. Choose Location

- **Workspace hooks** (`<workspace>/hooks/`): Per-agent, highest precedence

Managed hooks (`~/.openclaw/hooks/`): Shared across workspaces

2. Create Directory Structure

```
mkdir -p ~/.openclaw/hooks/my-hook
cd ~/.openclaw/hooks/my-hook
```

3. Create HOOK.md

```
---
name: my-hook
description: "Does something useful"
metadata: { "openclaw": { "emoji": "🎯", "events": ["command:new"] } }
---

# My Custom Hook

This hook does something useful when you issue `/new`.
```

4. Create handler.ts

```
import type { HookHandler } from "../../src/hooks/hooks.js";

const handler: HookHandler = async (event) => {
  if (event.type !== "command" || event.action !== "new") {
    return;
  }

  console.log("[my-hook] Running!");
  // Your logic here
};

export default handler;
```

5. Enable and Test

```
# Verify hook is discovered
openclaw hooks list

# Enable it
openclaw hooks enable my-hook

# Restart your gateway process (menu bar app restart on macOS, or restart your dev

# Trigger the event
# Send /new via your messaging channel
```

Configuration

New Config Format (Recommended)

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "entries": {
        "session-memory": { "enabled": true },
        "command-logger": { "enabled": false }
      }
    }
  }
}
```

Per-Hook Configuration

Hooks can have custom configuration:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "entries": {
        "my-hook": {
          "enabled": true,
          "env": {
            "MY_CUSTOM_VAR": "value"
          }
        }
      }
    }
  }
}
```

Extra Directories

Load hooks from additional directories:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "load": {
        "extraDirs": ["/path/to/more/hooks"]
      }
    }
  }
}
```

Legacy Config Format (Still Supported)

The old config format still works for backwards compatibility:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "handlers": [
        {
          "event": "command:new",
          "module": "./hooks/handlers/my-handler.ts",
          "export": "default"
        }
      ]
    }
  }
}
```

Note: `module` must be a workspace-relative path. Absolute paths and traversal outside the workspace are rejected.

Migration: Use the new discovery-based system for new hooks. Legacy handlers are loaded after directory-based hooks.

CLI Commands

List Hooks

```
# List all hooks
openclaw hooks list

# Show only eligible hooks
openclaw hooks list --eligible

# Verbose output (show missing requirements)
openclaw hooks list --verbose

# JSON output
openclaw hooks list --json
```

Hook Information

```
# Show detailed info about a hook
openclaw hooks info session-memory

# JSON output
openclaw hooks info session-memory --json
```

Check Eligibility

```
# Show eligibility summary
openclaw hooks check

# JSON output
openclaw hooks check --json
```

Enable/Disable

```
# Enable a hook
openclaw hooks enable session-memory

# Disable a hook
openclaw hooks disable command-logger
```

Bundled hook reference

session-memory

Saves session context to memory when you issue `/new`.

Events: `command:new`

Requirements: `workspace.dir` must be configured

Output: `<workspace>/memory/YYYY-MM-DD-slug.md` (defaults to `~/.openclaw/workspace`)

What it does:

1. Uses the pre-reset session entry to locate the correct transcript
2. Extracts the last 15 lines of conversation
3. Uses LLM to generate a descriptive filename slug
4. Saves session metadata to a dated memory file

Example output:

```
# Session: 2026-01-16 14:30:00 UTC

- **Session Key**: agent:main:main
- **Session ID**: abc123def456
- **Source**: telegram
```

Filename examples:

- `2026-01-16-vendor-pitch.md`
- `2026-01-16-api-design.md`
- `2026-01-16-1430.md` (fallback timestamp if slug generation fails)

Enable:

```
openclaw hooks enable session-memory
```

bootstrap-extra-files

Injects additional bootstrap files (for example monorepo-local `AGENTS.md` / `TOOLS.md`) during `agent:bootstrap`.

Events: `agent:bootstrap`

Requirements: `workspace.dir` must be configured

Output: No files written; bootstrap context is modified in-memory only.

Config:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "entries": {
        "bootstrap-extra-files": {
          "enabled": true,
          "paths": ["packages/*/AGENTS.md", "packages/*/TOOLS.md"]
        }
      }
    }
  }
}
```

Notes:

- Paths are resolved relative to workspace.
- Files must stay inside workspace (realpath-checked).
- Only recognized bootstrap basenames are loaded.
- Subagent allowlist is preserved (`AGENTS.md` and `TOOLS.md` only).

Enable:

```
openclaw hooks enable bootstrap-extra-files
```

command-logger

Logs all command events to a centralized audit file.

Events: command

Requirements: None

Output: ~/.openclaw/logs/commands.log

What it does:

1. Captures event details (command action, timestamp, session key, sender ID, source)
2. Appends to log file in JSONL format
3. Runs silently in the background

Example log entries:

```
{"timestamp":"2026-01-16T14:30:00.000Z","action":"new","sessionKey":"a
{"timestamp":"2026-01-16T15:45:22.000Z","action":"stop","sessionKey":"agent:main:m
```

View logs:

```
# View recent commands
tail -n 20 ~/.openclaw/logs/commands.log

# Pretty-print with jq
cat ~/.openclaw/logs/commands.log | jq .

# Filter by action
grep '"action":"new"' ~/.openclaw/logs/commands.log | jq .
```

Enable:

```
openclaw hooks enable command-logger
```

boot-md

Runs `BOOT.md` when the gateway starts (after channels start). Internal hooks must be enabled for this to run.

Events: `gateway:startup`

Requirements: `workspace.dir` must be configured

What it does:

1. Reads `BOOT.md` from your workspace
2. Runs the instructions via the agent runner
3. Sends any requested outbound messages via the message tool

Enable:

```
openclaw hooks enable boot-md
```

Best Practices

Keep Handlers Fast

Hooks run during command processing. Keep them lightweight:

```
// ✓ Good - async work, returns immediately
const handler: HookHandler = async (event) => {
  void processInBackground(event); // Fire and forget
};

// ✗ Bad - blocks command processing
const handler: HookHandler = async (event) => {
  await slowDatabaseQuery(event);
  await evenSlowerAPICall(event);
};
```

Handle Errors Gracefully

Always wrap risky operations:

```
const handler: HookHandler = async (event) => {
  try {
    await riskyOperation(event);
  } catch (err) {
    console.error("[my-handler] Failed:", err instanceof Error ? err.message : Str
    // Don't throw - let other handlers run
  }
};
```

Filter Events Early

Return early if the event isn't relevant:

```
const handler: HookHandler = async (event) => {
  // Only handle 'new' commands
  if (event.type !== "command" || event.action !== "new") {
    return;
  }

  // Your logic here
};
```

Use Specific Event Keys

Specify exact events in metadata when possible:

```
metadata: { "openclaw": { "events": ["command:new"] } } # Specific
```

Rather than:

```
metadata: { "openclaw": { "events": ["command"] } } # General - more c
```

›

Debugging

Enable Hook Logging

The gateway logs hook loading at startup:

```
Registered hook: session-memory -> command:new
Registered hook: bootstrap-extra-files -> agent:bootstrap
Registered hook: command-logger -> command
Registered hook: boot-md -> gateway:startup
```

Check Discovery

List all discovered hooks:

```
openclaw hooks list --verbose
```

Check Registration

In your handler, log when it’s called:

```
const handler: HookHandler = async (event) => {
  console.log("[my-handler] Triggered:", event.type, event.action);
  // Your logic
};
```

Verify Eligibility

Check why a hook isn’t eligible:

```
openclaw hooks info my-hook
```

Look for missing requirements in the output.

Testing

Gateway Logs

Monitor gateway logs to see hook execution:

```
# macOS
./scripts/clawlog.sh -f

# Other platforms
tail -f ~/.openclaw/gateway.log
```

Test Hooks Directly

Test your handlers in isolation:

```
import { test } from "vitest";
import { createHookEvent } from "./src/hooks/hooks.js";
import myHandler from "./hooks/my-hook/handler.js";

test("my handler works", async () => {
  const event = createHookEvent("command", "new", "test-session", {
    foo: "bar",
  });

  await myHandler(event);

  // Assert side effects
});
```

Architecture

Core Components

- `src/hooks/types.ts` : Type definitions
- `src/hooks/workspace.ts` : Directory scanning and loading
- `src/hooks/frontmatter.ts` : HOOK.md metadata parsing
- `src/hooks/config.ts` : Eligibility checking
- `src/hooks/hooks-status.ts` : Status reporting
- `src/hooks/loader.ts` : Dynamic module loader
- `src/cli/hooks-cli.ts` : CLI commands
- `src/gateway/server-startup.ts` : Loads hooks at gateway start
- `src/auto-reply/reply/commands-core.ts` : Triggers command events

Discovery Flow

```
Gateway startup
    ↓
Scan directories (workspace → managed → bundled)
    ↓
Parse HOOK.md files
    ↓
Check eligibility (bins, env, config, os)
    ↓
Load handlers from eligible hooks
    ↓
Register handlers for events
```

Event Flow

```
User sends /new
    ↓
Command validation
    ↓
Create hook event
    ↓
Trigger hook (all registered handlers)
    ↓
Command processing continues
    ↓
Session reset
```

Troubleshooting

Hook Not Discovered

1. Check directory structure:

```
ls -la ~/.openclaw/hooks/my-hook/
# Should show: HOOK.md, handler.ts
```

2. Verify HOOK.md format:

```
cat ~/.openclaw/hooks/my-hook/HOOK.md
# Should have YAML frontmatter with name and metadata
```

3. List all discovered hooks:

```
openclaw hooks list
```

Hook Not Eligible

Check requirements:

```
openclaw hooks info my-hook
```

Look for missing:

- Binaries (check PATH)
- Environment variables
- Config values
- OS compatibility

Hook Not Executing

1. Verify hook is enabled:

```
openclaw hooks list
# Should show ✓ next to enabled hooks
```

2. Restart your gateway process so hooks reload.
3. Check gateway logs for errors:

```
./scripts/clawlog.sh | grep hook
```

Handler Errors

Check for TypeScript/import errors:

```
# Test import directly
node -e "import('./path/to/handler.ts').then(console.log)"
```

Migration Guide

From Legacy Config to Discovery

Before:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "handlers": [
        {
          "event": "command:new",
          "module": "./hooks/handlers/my-handler.ts"
        }
      ]
    }
  }
}
```

After:

1. Create hook directory:

```
mkdir -p ~/.openclaw/hooks/my-hook
mv ./hooks/handlers/my-handler.ts ~/.openclaw/hooks/my-hook/handler.ts
```

2. Create HOOK.md:

```
---
name: my-hook
description: "My custom hook"
metadata: { "openclaw": { "emoji": "🎯", "events": ["command:new"] } }
---

# My Hook

Does something useful.
```

3. Update config:

```
{
  "hooks": {
    "internal": {
      "enabled": true,
      "entries": {
        "my-hook": { "enabled": true }
      }
    }
  }
}
```

4. Verify and restart your gateway process:

```
openclaw hooks list
# Should show: 🎯 my-hook ✓
```

Benefits of migration:

- Automatic discovery
- CLI management
- Eligibility checking
- Better documentation
- Consistent structure

See Also

‹ Zalo Personal Plugin Cron Jobs ›