Browser

Chrome Extension

The OpenClaw Chrome extension lets the agent control your **existing Chrome tabs** (your normal Chrome window) instead of launching a separate openclaw-managed Chrome profile.

Attach/detach happens via a **single Chrome toolbar button**.

What it is (concept)

There are three parts:

- **Browser control service** (Gateway or node): the API the agent/tool calls (via the Gateway)
- **Local relay server** (loopback CDP): bridges between the control server and the extension (`http://127.0.0.1:18792` by default)
- **Chrome MV3 extension**: attaches to the active tab using `chrome.debugger` and pipes CDP messages to the relay

OpenClaw then controls the attached tab through the normal `browser` tool surface (selecting the right profile).

Install / load (unpacked)

1. Install the extension to a stable local path:

```
openclaw browser extension install
```

2. Print the installed extension directory path:

```
openclaw browser extension path
```

3. Chrome → `chrome://extensions`

   Enable “Developer mode”

   “Load unpacked” → select the directory printed above

4. Pin the extension.

Updates (no build step)

The extension ships inside the OpenClaw release (npm package) as static files. There is no separate “build” step.

After upgrading OpenClaw:

- Re-run `openclaw browser extension install` to refresh the installed files under your OpenClaw state directory.
- Chrome → `chrome://extensions` → click “Reload” on the extension.

Use it (no extra config)

OpenClaw ships with a built-in browser profile named `chrome` that targets the extension relay on the default port.

Use it:

- CLI: `openclaw browser --browser-profile chrome tabs`
- Agent tool: `browser` with `profile="chrome"`

If you want a different name or a different relay port, create your own profile:

```
openclaw browser create-profile \
  --name my-chrome \
  --driver extension \
  --cdp-url http://127.0.0.1:18792 \
  --color "#00AA00"
```

Attach / detach (toolbar button)

Open the tab you want OpenClaw to control.

Click the extension icon.

Badge shows `ON` when attached.

Click again to detach.

Which tab does it control?

It does **not** automatically control “whatever tab you’re looking at”.

It controls **only the tab(s) you explicitly attached** by clicking the toolbar button.

To switch: open the other tab and click the extension icon there.

Badge + common errors

`ON` : attached; OpenClaw can drive that tab.

`…` : connecting to the local relay.

`!` : relay not reachable (most common: browser relay server isn’t running on this machine).

If you see `!` :

Make sure the Gateway is running locally (default setup), or run a node host on this machine if the Gateway runs elsewhere.

Open the extension Options page; it shows whether the relay is reachable.

Remote Gateway (use a node host)

Local Gateway (same machine as Chrome) — usually no extra steps

If the Gateway runs on the same machine as Chrome, it starts the browser control service on loopback and auto-starts the relay server. The extension talks to the local relay; the CLI/tool calls go to the Gateway.

Remote Gateway (Gateway runs elsewhere) — run a node host

If your Gateway runs on another machine, start a node host on the machine that runs Chrome. The Gateway will proxy browser actions to that node; the extension + relay stay local to the browser machine.

If multiple nodes are connected, pin one with `gateway.nodes.browser.node` or set `gateway.nodes.browser.mode` .

Sandboxing (tool containers)

If your agent session is sandboxed (`agents.defaults.sandbox.mode != "off"`), the `browser` tool can be restricted:

- By default, sandboxed sessions often target the **sandbox browser** (`target="sandbox"`), not your host Chrome.
- Chrome extension relay takeover requires controlling the **host** browser control server.

Options:

Easiest: use the extension from a **non-sandboxed** session/agent.

Or allow host browser control for sandboxed sessions:

```
{
  agents: {
    defaults: {
      sandbox: {
        browser: {
          allowHostControl: true,
        },
      },
    },
  },
}
```

Then ensure the tool isn't denied by tool policy, and (if needed) call `browser` with `target="host"`.

Debugging: `openclaw sandbox explain`

Remote access tips

- Keep the Gateway and node host on the same tailnet; avoid exposing relay ports to LAN or public Internet.
- Pair nodes intentionally; disable browser proxy routing if you don't want remote control (`gateway.nodes.browser.mode="off"`).

How "extension path" works

`openclaw browser extension path` prints the **installed** on-disk directory containing the extension files.

The CLI intentionally does **not** print a `node_modules` path. Always run `openclaw browser extension install` first to copy the extension to a stable location under your OpenClaw state directory.

If you move or delete that install directory, Chrome will mark the extension as broken until you reload it from a valid path.

Security implications (read this)

This is powerful and risky. Treat it like giving the model "hands on your browser".

- The extension uses Chrome's debugger API (`chrome.debugger`). When attached, the model can:
  - click/type/navigate in that tab
  - read page content
  - access whatever the tab's logged-in session can access
- **This is not isolated** like the dedicated openclaw-managed profile.
  - If you attach to your daily-driver profile/tab, you're granting access to that account state.

Recommendations:

- Prefer a dedicated Chrome profile (separate from your personal browsing) for extension relay usage.
- Keep the Gateway and any node hosts tailnet-only; rely on Gateway auth + node pairing.
- Avoid exposing relay ports over LAN (`0.0.0.0`) and avoid Funnel (public).
- The relay blocks non-extension origins and requires an internal auth token for CDP clients.

Related:

- Browser tool overview: **Browser**

Security audit: **Security**

Tailscale setup: **Tailscale**

‹ Browser Login Browser Troubleshooting ›