Browser

Browser (OpenClaw-managed)

OpenClaw can run a **dedicated Chrome/Brave/Edge/Chromium profile** that the agent controls. It is isolated from your personal browser and is managed through a small local control service inside the Gateway (loopback only).

Beginner view:

- Think of it as a **separate, agent-only browser**.
- The `openclaw` profile does **not** touch your personal browser profile.
- The agent can **open tabs, read pages, click, and type** in a safe lane.
- The default `chrome` profile uses the **system default Chromium browser** via the extension relay; switch to `openclaw` for the isolated managed browser.

What you get

- A separate browser profile named **openclaw** (orange accent by default).
- Deterministic tab control (list/open/focus/close).
- Agent actions (click/type/drag/select), snapshots, screenshots, PDFs.
- Optional multi-profile support (`openclaw` , `work` , `remote` , …).

This browser is **not** your daily driver. It is a safe, isolated surface for agent automation and verification.

Quick start

```
openclaw browser --browser-profile openclaw status
openclaw browser --browser-profile openclaw start
openclaw browser --browser-profile openclaw open https://example.com
openclaw browser --browser-profile openclaw snapshot
```

If you get “Browser disabled”, enable it in config (see below) and restart the Gateway.

Profiles: `openclaw` vs `chrome`

- `openclaw` : managed, isolated browser (no extension required).
- `chrome` : extension relay to your **system browser** (requires the OpenClaw extension to be attached to a tab).

Set `browser.defaultProfile: "openclaw"` if you want managed mode by default.

Configuration

Browser settings live in `~/.openclaw/openclaw.json` .

```
  browser: {
    enabled: true, // default: true
    // cdpUrl: "http://127.0.0.1:18792", // legacy single-profile override
    remoteCdpTimeoutMs: 1500, // remote CDP HTTP timeout (ms)
    remoteCdpHandshakeTimeoutMs: 3000, // remote CDP WebSocket handshake timeout (ms
    defaultProfile: "chrome",
    color: "#FF4500",
    headless: false,
    noSandbox: false,
    attachOnly: false,
    executablePath: "/Applications/Brave Browser.app/Contents/MacOS/Brave Browser",
    profiles: {
      openclaw: { cdpPort: 18800, color: "#FF4500" },
      work: { cdpPort: 18801, color: "#0066CC" },
      remote: { cdpUrl: "http://10.0.0.42:9222", color: "#00AA00" },
    },
  },
}
```

Notes:

- The browser control service binds to loopback on a port derived from `gateway.port` (default: `18791`, which is gateway + 2). The relay uses the next port (`18792`).
- If you override the Gateway port (`gateway.port` or `OPENCLAW_GATEWAY_PORT`), the derived browser ports shift to stay in the same “family”.
- `cdpUrl` defaults to the relay port when unset.
- `remoteCdpTimeoutMs` applies to remote (non-loopback) CDP reachability checks.
- `remoteCdpHandshakeTimeoutMs` applies to remote CDP WebSocket reachability checks.
- `attachOnly: true` means “never launch a local browser; only attach if it is already running.”

`color` + per-profile `color` tint the browser UI so you can see which profile is active.

Default profile is `chrome` (extension relay). Use `defaultProfile: "openclaw"` for the managed browser.

Auto-detect order: system default browser if Chromium-based; otherwise Chrome → Brave → Edge → Chromium → Chrome Canary.

Local `openclaw` profiles auto-assign `cdpPort` / `cdpUrl` — set those only for remote CDP.

Use Brave (or another Chromium-based browser)

If your **system default** browser is Chromium-based (Chrome/Brave/Edge/etc), OpenClaw uses it automatically. Set `browser.executablePath` to override auto-detection:

CLI example:

```
openclaw config set browser.executablePath "/usr/bin/google-chrome"
```

```
// macOS
{
  browser: {
    executablePath: "/Applications/Brave Browser.app/Contents/MacOS/Brave Browser"
  }
}

// Windows
{
  browser: {
    executablePath: "C:\\Program Files\\BraveSoftware\\Brave-Browser\\Application\\b
  }
}

// Linux
{
  browser: {
    executablePath: "/usr/bin/brave-browser"
  }
}
```

Local vs remote control

- **Local control (default):** the Gateway starts the loopback control service and can launch a local browser.
- **Remote control (node host):** run a node host on the machine that has the browser; the Gateway proxies browser actions to it.
- **Remote CDP:** set `browser.profiles.<name>.cdpUrl` (or `browser.cdpUrl`) to attach to a remote Chromium-based browser. In this case, OpenClaw will not launch a local browser.

Remote CDP URLs can include auth:

- Query tokens (e.g., `https://provider.example?token=<token>`)
- HTTP Basic auth (e.g., `https://user:pass@provider.example`)

OpenClaw preserves the auth when calling `/json/*` endpoints and when connecting to the CDP WebSocket. Prefer environment variables or secrets managers for tokens instead of committing them to config files.

Node browser proxy (zero-config default)

If you run a **node host** on the machine that has your browser, OpenClaw can auto-route browser tool calls to that node without any extra browser config. This is the default path for remote gateways.

Notes:

- The node host exposes its local browser control server via a **proxy command**.
- Profiles come from the node's own `browser.profiles` config (same as local).
- Disable if you don't want it:
  - On the node: `nodeHost.browserProxy.enabled=false`
  - On the gateway: `gateway.nodes.browser.mode="off"`

Browserless (hosted remote CDP)

Browserless is a hosted Chromium service that exposes CDP endpoints over HTTPS. You can point a OpenClaw browser profile at a Browserless region endpoint and authenticate with your API key.

Example:

```
  browser: {
    enabled: true,
    defaultProfile: "browserless",
    remoteCdpTimeoutMs: 2000,
    remoteCdpHandshakeTimeoutMs: 4000,
    profiles: {
      browserless: {
        cdpUrl: "https://production-sfo.browserless.io?token=<BROWSERLESS_API_KEY>",
        color: "#00AA00",
      },
    },
  },
}
```

Notes:

- Replace `<BROWSERLESS_API_KEY>` with your real Browserless token.
- Choose the region endpoint that matches your Browserless account (see their docs).

Security

Key ideas:

- Browser control is loopback-only; access flows through the Gateway's auth or node pairing.
- If browser control is enabled and no auth is configured, OpenClaw auto-generates `gateway.auth.token` on startup and persists it to config.
- Keep the Gateway and any node hosts on a private network (Tailscale); avoid public exposure.
- Treat remote CDP URLs/tokens as secrets; prefer env vars or a secrets manager.

Remote CDP tips:

Prefer HTTPS endpoints and short-lived tokens where possible.

Avoid embedding long-lived tokens directly in config files.

›

Profiles (multi-browser)

OpenClaw supports multiple named profiles (routing configs). Profiles can be:

- **openclaw-managed**: a dedicated Chromium-based browser instance with its own user data directory + CDP port
- **remote**: an explicit CDP URL (Chromium-based browser running elsewhere)
- **extension relay**: your existing Chrome tab(s) via the local relay + Chrome extension

Defaults:

- The `openclaw` profile is auto-created if missing.
- The `chrome` profile is built-in for the Chrome extension relay (points at `http://127.0.0.1:18792` by default).
- Local CDP ports allocate from **18800–18899** by default.
- Deleting a profile moves its local data directory to Trash.

All control endpoints accept `?profile=<name>` ; the CLI uses `--browser-profile` .

Chrome extension relay (use your existing Chrome)

OpenClaw can also drive **your existing Chrome tabs** (no separate “openclaw” Chrome instance) via a local CDP relay + a Chrome extension.

Full guide: Chrome extension

Flow:

The Gateway runs locally (same machine) or a node host runs on the browser machine.

A local **relay server** listens at a loopback `cdpUrl` (default: `http://127.0.0.1:18792`).

You click the **OpenClaw Browser Relay** extension icon on a tab to attach (it does not auto-attach).

The agent controls that tab via the normal `browser` tool, by selecting the right profile.

If the Gateway runs elsewhere, run a node host on the browser machine so the Gateway can proxy browser actions.

Sandboxed sessions

If the agent session is sandboxed, the `browser` tool may default to `target="sandbox"` (sandbox browser). Chrome extension relay takeover requires host browser control, so either:

run the session unsandboxed, or

set `agents.defaults.sandbox.browser.allowHostControl: true` and use `target="host"` when calling the tool.

Setup

1. Load the extension (dev/unpacked):

```
openclaw browser extension install
```

Chrome → `chrome://extensions` → enable “Developer mode”

“Load unpacked” → select the directory printed by `openclaw browser extension path`

Pin the extension, then click it on the tab you want to control (badge shows `ON`).

2. Use it:

CLI: `openclaw browser --browser-profile chrome tabs`

Agent tool: `browser` with `profile="chrome"`

Optional: if you want a different name or 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"
```

Notes:

- This mode relies on Playwright-on-CDP for most operations (screenshots/snapshots/actions).
- Detach by clicking the extension icon again.

Isolation guarantees

- **Dedicated user data dir**: never touches your personal browser profile.
- **Dedicated ports**: avoids `9222` to prevent collisions with dev workflows.
- **Deterministic tab control**: target tabs by `targetId`, not “last tab”.

Browser selection

When launching locally, OpenClaw picks the first available:

1. Chrome
2. Brave
3. Edge

4. Chromium
5. Chrome Canary

You can ovèrride with `browser.executablePath`.

Platforms:

- macOS: checks `/Applications` and `~/Applications`.
- Linux: looks for `google-chrome`, `brave`, `microsoft-edge`, `chromium`, etc.
- Windows: checks common install locations.

Control API (optional)

For local integrations only, the Gateway exposes a small loopback HTTP API:

- Status/start/stop: `GET /`, `POST /start`, `POST /stop`
- Tabs: `GET /tabs`, `POST /tabs/open`, `POST /tabs/focus`, `DELETE /tabs/:targetId`
- Snapshot/screenshot: `GET /snapshot`, `POST /screenshot`
- Actions: `POST /navigate`, `POST /act`
- Hooks: `POST /hooks/file-chooser`, `POST /hooks/dialog`
- Downloads: `POST /download`, `POST /wait/download`
- Debugging: `GET /console`, `POST /pdf`
- Debugging: `GET /errors`, `GET /requests`, `POST /trace/start`, `POST /trace/stop`, `POST /highlight`
- Network: `POST /response/body`
- State: `GET /cookies`, `POST /cookies/set`, `POST /cookies/clear`
- State: `GET /storage/:kind`, `POST /storage/:kind/set`, `POST /storage/:kind/clear`
- Settings: `POST /set/offline`, `POST /set/headers`, `POST /set/credentials`, `POST /set/geolocation`, `POST /set/media`, `POST /set/timezone`, `POST`

`/set/locale` , `POST /set/device`

All endpoints accept `?profile=<name>` .

›

If gateway auth is configured, browser HTTP routes require auth too:

`Authorization: Bearer <gateway token>`

`x-openclaw-password: <gateway password>` or HTTP Basic auth with that password

Playwright requirement

Some features (navigate/act/AI snapshot/role snapshot, element screenshots, PDF) require Playwright. If Playwright isn't installed, those endpoints return a clear 501 error. ARIA snapshots and basic screenshots still work for openclaw-managed Chrome. For the Chrome extension relay driver, ARIA snapshots and screenshots require Playwright.

If you see `Playwright is not available in this gateway build` , install the full Playwright package (not `playwright-core`) and restart the gateway, or reinstall OpenClaw with browser support.

Docker Playwright install

If your Gateway runs in Docker, avoid `npx playwright` (npm override conflicts). Use the bundled CLI instead:

```
docker compose run --rm openclaw-cli \
  node /app/node_modules/playwright-core/cli.js install chromium
```

To persist browser downloads, set `PLAYWRIGHT_BROWSERS_PATH` (for example, `/home/node/.cache/ms-playwright`) and make sure `/home/node` is persisted via `OPENCLAW_HOME_VOLUME` or a bind mount. See .

How it works (internal)

High-level flow:

- A small **control server** accepts HTTP requests.
- It connects to Chromium-based browsers (Chrome/Brave/Edge/Chromium) via **CDP**.
- For advanced actions (click/type/snapshot/PDF), it uses **Playwright** on top of CDP.
- When Playwright is missing, only non-Playwright operations are available.

This design keeps the agent on a stable, deterministic interface while letting you swap local/remote browsers and profiles.

CLI quick reference

All commands accept `--browser-profile <name>` to target a specific profile. All commands also accept `--json` for machine-readable output (stable payloads).

Basics:

```
openclaw browser status
openclaw browser start
openclaw browser stop
openclaw browser tabs
openclaw browser tab
openclaw browser tab new
openclaw browser tab select 2
openclaw browser tab close 2
openclaw browser open https://example.com
openclaw browser focus abcd1234
```

```
openclaw browser close abcd1234
```

Inspection:

```
openclaw browser screenshot

openclaw browser screenshot --full-page

openclaw browser screenshot --ref 12

openclaw browser screenshot --ref e12

openclaw browser snapshot

openclaw browser snapshot --format aria --limit 200

openclaw browser snapshot --interactive --compact --depth 6

openclaw browser snapshot --efficient

openclaw browser snapshot --labels

openclaw browser snapshot --selector "#main" --interactive

openclaw browser snapshot --frame "iframe#main" --interactive

openclaw browser console --level error

openclaw browser errors --clear

openclaw browser requests --filter api --clear

openclaw browser pdf

openclaw browser responsebody "**/api" --max-chars 5000
```

Actions:

```
openclaw browser navigate https://example.com

openclaw browser resize 1280 720

openclaw browser click 12 --double

openclaw browser click e12 --double

openclaw browser type 23 "hello" --submit

openclaw browser press Enter

openclaw browser hover 44
```

```
openclaw browser scrollintoview e12
openclaw browser drag 10 11
openclaw browser select 9 OptionA OptionB
openclaw browser download e12 report.pdf
openclaw browser waitfordownload report.pdf
openclaw browser upload /tmp/openclaw/uploads/file.pdf
openclaw browser fill --fields '[{"ref":"1","type":"text","value":"Ada"}]'
openclaw browser dialog --accept
openclaw browser wait --text "Done"
openclaw browser wait "#main" --url "**/dash" --load networkidle --fn "window.ready===true"
openclaw browser evaluate --fn '(el) => el.textContent' --ref 7
openclaw browser highlight e12
openclaw browser trace start
openclaw browser trace stop
```

State:

```
openclaw browser cookies
openclaw browser cookies set session abc123 --url "https://example.com"
openclaw browser cookies clear
openclaw browser storage local get
openclaw browser storage local set theme dark
openclaw browser storage session clear
openclaw browser set offline on
openclaw browser set headers --headers-json '{"X-Debug":"1"}'
openclaw browser set credentials user pass
openclaw browser set credentials --clear
openclaw browser set geo 37.7749 -122.4194 --origin "https://example.com"
```

```
openclaw browser set geo --clear
openclaw browser set media dark
openclaw browser set timezone America/New_York
openclaw browser set locale en-US
openclaw browser set device "iPhone 14"
```

Notes:

- `upload` and `dialog` are **arming** calls; run them before the click/press that triggers the chooser/dialog.
- Download and trace output paths are constrained to OpenClaw temp roots:
  - traces: `/tmp/openclaw` (fallback: `${os.tmpdir()}/openclaw`)
  - downloads: `/tmp/openclaw/downloads` (fallback: `${os.tmpdir()}/openclaw/downloads`)
- Upload paths are constrained to an OpenClaw temp uploads root:
  - uploads: `/tmp/openclaw/uploads` (fallback: `${os.tmpdir()}/openclaw/uploads`)
- `upload` can also set file inputs directly via `--input-ref` or `--element`.
- `snapshot`:
  - `--format ai` (default when Playwright is installed): returns an AI snapshot with numeric refs (`aria-ref="<n>"`).
  - `--format aria`: returns the accessibility tree (no refs; inspection only).
  - `--efficient` (or `--mode efficient`): compact role snapshot preset (interactive + compact + depth + lower maxChars).
  - Config default (tool/CLI only): set `browser.snapshotDefaults.mode: "efficient"` to use efficient snapshots when the caller does not pass a mode (see **Gateway configuration**).

Role snapshot options (`--interactive` , `--compact` , `--depth` , `--selector`) force a role-based snapshot with refs like `ref=e12` .

`--frame "<iframe selector>"` scopes role snapshots to an iframe (pairs with role refs like `e12`).

`--interactive` outputs a flat, easy-to-pick list of interactive elements (best for driving actions).

`--labels` adds a viewport-only screenshot with overlayed ref labels (prints `MEDIA:<path>`).

`click` / `type` /etc require a `ref` from `snapshot` (either numeric `12` or role ref `e12`). CSS selectors are intentionally not supported for actions.

Snapshots and refs

OpenClaw supports two “snapshot” styles:

AI snapshot (numeric refs): `openclaw browser snapshot` (default; `--format ai`)

Output: a text snapshot that includes numeric refs.

Actions: `openclaw browser click 12` , `openclaw browser type 23 "hello"` .

Internally, the ref is resolved via Playwright’s `aria-ref` .

Role snapshot (role refs like `e12`): `openclaw browser snapshot --interactive` (or `--compact` , `--depth` , `--selector` , `--frame`)

Output: a role-based list/tree with `[ref=e12]` (and optional `[nth=1]`).

Actions: `openclaw browser click e12` , `openclaw browser highlight e12` .

Internally, the ref is resolved via `getByRole(...)` (plus `nth()` for duplicates).

Add `--labels` to include a viewport screenshot with overlayed `e12` labels.

Ref behavior:

Refs are **not stable across navigations**; if something fails, re-run `snapshot` and use a fresh ref.

If the role snapshot was taken with `--frame`, role refs are scoped to that iframe until the next role snapshot.

Wait power-ups

You can wait on more than just time/text:

Wait for URL (globs supported by Playwright):

```
openclaw browser wait --url "**/dash"
```

Wait for load state:

```
openclaw browser wait --load networkidle
```

Wait for a JS predicate:

```
openclaw browser wait --fn "window.ready===true"
```

Wait for a selector to become visible:

```
openclaw browser wait "#main"
```

These can be combined:

```
openclaw browser wait "#main" \
  --url "**/dash" \
  --load networkidle \
  --fn "window.ready===true" \
  --timeout-ms 15000
```

Debug workflows

When an action fails (e.g. “not visible”, “strict mode violation”, “covered”):

1. `openclaw browser snapshot --interactive`

2. Use `click <ref>` / `type <ref>` (prefer role refs in interactive mode)
3. If it still fails: `openclaw browser highlight <ref>` to see what Playwright is targeting
4. If the page behaves oddly:

```
openclaw browser errors --clear
openclaw browser requests --filter api --clear
```

5. For deep debugging: record a trace:

```
openclaw browser trace start
reproduce the issue
openclaw browser trace stop (prints TRACE:<path>)
```

JSON output

`--json` is for scripting and structured tooling.

Examples:

```
openclaw browser status --json
openclaw browser snapshot --interactive --json
openclaw browser requests --filter api --json
openclaw browser cookies --json
```

Role snapshots in JSON include `refs` plus a small `stats` block (lines/chars/refs/interactive) so tools can reason about payload size and density.

State and environment knobs

These are useful for “make the site behave like X” workflows:

- Cookies: `cookies`, `cookies set`, `cookies clear`
- Storage: `storage local|session get|set|clear`

Offline: `set offline on|off`

Headers: `set headers --headers-json '{"X-Debug":"1"}'` (legacy `set headers --json '{"X-Debug":"1"}'` remains supported)

HTTP basic auth: `set credentials user pass` (or `--clear`)

Geolocation: `set geo <lat> <lon> --origin "https://example.com"` (or `--clear`)

Media: `set media dark|light|no-preference|none`

Timezone / locale: `set timezone ...`, `set locale ...`

Device / viewport:

`set device "iPhone 14"` (Playwright device presets)

`set viewport 1280 720`

Security & privacy

The openclaw browser profile may contain logged-in sessions; treat it as sensitive.

`browser act kind=evaluate` / `openclaw browser evaluate` and `wait --fn` execute arbitrary JavaScript in the page context. Prompt injection can steer this. Disable it with `browser.evaluateEnabled=false` if you do not need it.

For logins and anti-bot notes (X/Twitter, etc.), see **Browser login + X/Twitter posting**.

Keep the Gateway/node host private (loopback or tailnet-only).

Remote CDP endpoints are powerful; tunnel and protect them.

Troubleshooting

For Linux-specific issues (especially snap Chromium), see **Browser troubleshooting**.

Agent tools + how control works

The agent gets **one tool** for browser automation:

- `browser` — status/start/stop/tabs/open/focus/close/snapshot/screenshot/navigate/act

How it maps:

- `browser snapshot` returns a stable UI tree (AI or ARIA).
- `browser act` uses the snapshot `ref` IDs to click/type/drag/select.
- `browser screenshot` captures pixels (full page or element).
- `browser` accepts:
  - `profile` to choose a named browser profile (openclaw, chrome, or remote CDP).
  - `target` (`sandbox` | `host` | `node`) to select where the browser lives.
  - In sandboxed sessions, `target: "host"` requires `agents.defaults.sandbox.browser.allowHostControl=true`.
  - If `target` is omitted: sandboxed sessions default to `sandbox`, non-sandbox sessions default to `host`.
  - If a browser-capable node is connected, the tool may auto-route to it unless you pin `target="host"` or `target="node"`.

This keeps the agent deterministic and avoids brittle selectors.

‹ Reactions Browser Login ›