



## Fundamentals

# OAuth

OpenClaw supports “subscription auth” via OAuth for providers that offer it (notably [OpenAI Codex \(ChatGPT OAuth\)](#)). For Anthropic subscriptions, use the `setup-token` flow. This page explains:

how the OAuth `token exchange` works (PKCE)

where tokens are `stored` (and why)

how to handle `multiple accounts` (profiles + per-session overrides)

OpenClaw also supports `provider plugins` that ship their own OAuth or API-key flows. Run them via:

```
openclaw models auth login --provider <id>
```

## The token sink (why it exists)

OAuth providers commonly mint a `new refresh token` during login/refresh flows. Some providers (or OAuth clients) can invalidate older refresh tokens when a new one is issued for the same user/app.

Practical symptom:

you log in via OpenClaw and via Claude Code / Codex CLI → one of them randomly gets “logged out” later

To reduce that, OpenClaw treats `auth-profiles.json` as a `token sink`:



the runtime reads credentials from **one place**

we can keep multiple profiles and route them deterministically

&gt;

## Storage (where tokens live)

Secrets are stored **per-agent**:

Auth profiles (OAuth + API keys):

```
~/.openclaw/agents/<agentId>/agent/auth-profiles.json
```

Runtime cache (managed automatically; don't edit):

```
~/.openclaw/agents/<agentId>/agent/auth.json
```

Legacy import-only file (still supported, but not the main store):

```
~/.openclaw/credentials/oauth.json (imported into auth-profiles.json on first use)
```

All of the above also respect `$OPENCLAW_STATE_DIR` (state dir override).

Full reference: [/gateway/configuration](#)

## Anthropic setup-token (subscription auth)

Run `claude setup-token` on any machine, then paste it into OpenClaw:

```
openclaw models auth setup-token --provider anthropic
```

If you generated the token elsewhere, paste it manually:

```
openclaw models auth paste-token --provider anthropic
```

Verify:

 openclaw models status

## OAuth exchange (how login works)

OpenClaw's interactive login flows are implemented in `@mariozechner/piai` and wired into the wizards/commands.

### Anthropic (Claude Pro/Max) setup-token

Flow shape:

1. run `claude setup-token`
2. paste the token into OpenClaw
3. store as a token auth profile (no refresh)

The wizard path is `openclaw onboard` → auth choice `setup-token` (Anthropic).

### OpenAI Codex (ChatGPT OAuth)

Flow shape (PKCE):

1. generate PKCE verifier/challenge + random state
2. open `https://auth.openai.com/oauth/authorize?...`
3. try to capture callback on `http://127.0.0.1:1455/auth/callback`
4. if callback can't bind (or you're remote/headless), paste the redirect URL/code
5. exchange at `https://auth.openai.com/oauth/token`
6. extract `accountId` from the access token and store `{ access, refresh, expires, accountId }`

Wizard path is `openclaw onboard` → auth choice `openai-codex`.

## Refresh + expiry

Profiles store an `expires` timestamp.

At runtime:

`if expires` is in the future → use the stored access token

`if expired` → refresh (under a file lock) and overwrite the stored credentials

The refresh flow is automatic; you generally don't need to manage tokens manually.

## Multiple accounts (profiles) + routing

Two patterns:

### 1) Preferred: separate agents

If you want “personal” and “work” to never interact, use isolated agents (separate sessions + credentials + workspace):

```
openclaw agents add work  
openclaw agents add personal
```

Then configure auth per-agent (wizard) and route chats to the right agent.

### 2) Advanced: multiple profiles in one agent

`auth-profiles.json` supports multiple profile IDs for the same provider.

Pick which profile is used:

globally via config ordering (`auth.order`)

per-session via /model ...@<profileId>



Example (session override):

```
>  
/model Opus@anthropic:work
```

How to see what profile IDs exist:

```
openclaw channels list --json (shows auth[] )
```

Related docs:

[/concepts/model-failover](#) (rotation + cooldown rules)

[/tools/slash-commands](#) (command surface)

[< Agent Workspace](#)

[Bootstrapping >](#)

Powered by [mintlify](#)