Chapter 1: Installing OpenClaw on Your Computer

> ***Goal:*** *Get OpenClaw running on your Windows machine — controllable via Telegram, WhatsApp, and a web dashboard.*

What You'll Have

- OpenClaw running on your computer
- Connected to Telegram and/or WhatsApp
- An AI provider powering it (Anthropic, OpenAI, or others)
- Security configured so it asks before doing anything dangerous

Time: ~30 minutes **Cost:** Depends on your provider — free with an existing ChatGPT/Codex subscription, or ~$1-5/month with Anthropic Haiku (pay-per-use via API key)

Prerequisites

- A computer running **Windows 10 or Windows 11**
- **PowerShell 5.1+** (ships with Windows)
- An AI provider — **one** of the following:
  - An **OpenAI subscription** (ChatGPT Plus/Pro or Codex) — no API key needed
  - An **Anthropic API key** (get one at console.anthropic.com — $5 free credits included)
- A Telegram account

Step 1: Install Node.js

Download and run the installer from nodejs.org. Choose the LTS or Current (v24) version. Make sure **"Add to PATH"** is checked.

Open a **new** PowerShell window and verify:

```
node --version   # Should show v24.x
npm --version
```

Step 2: Install OpenClaw

```
npm install -g openclaw@latest
openclaw --version
```

Step 3: Connect Your AI Provider

Choose **one** of the two options below.

Option A: OpenAI subscription (recommended if you already pay for

ChatGPT/Codex)

This uses your existing subscription — no API key needed. Log in via the CLI:

```
openclaw auth login --provider openai-codex
```

A browser window opens for you to sign in with your OpenAI account. Once authenticated, set the model:

```
openclaw config set `
  agents.defaults.model.primary `
  "openai-codex/gpt-5.4"
```

> ***Note:*** *Subscription-based models use the* `openai-codex/` *prefix. No* `.env` *file or API key required.*

Option B: Anthropic API key (pay-per-use)

Get your API key from console.anthropic.com/settings/keys, then:

```
New-Item -ItemType Directory -Force `
  -Path "$env:USERPROFILE\.openclaw"
notepad "$env:USERPROFILE\.openclaw\.env"
```

Add this line (paste your actual key):

```
ANTHROPIC_API_KEY=sk-ant-api03-YOUR-KEY-HERE
```

Save and close notepad. Then set the model:

```
openclaw config set `
  agents.defaults.model.primary `
  "anthropic/claude-haiku-4-5"
```

> ***Tip:*** *Haiku is fast and cheap (~$1-5/month for personal use). For smarter responses, use* `anthropic/claude-sonnet-4-6` *or* `anthropic/claude-opus-4-6` *(costs more).*

Step 4: Configure OpenClaw

Set the gateway mode

```
openclaw config set gateway.mode local
openclaw config set gateway.bind loopback
openclaw config set gateway.auth.mode token
```

Set up security

```
openclaw config set tools.exec.security ask
openclaw config set tools.exec.ask always
openclaw config set tools.elevated.enabled false
```

> ***What this does:*** *OpenClaw can read/write files, but* ***asks before*** *running any shell command. It cannot use sudo/admin. The gateway only listens locally.*

Step 5: Set Up Telegram

5.1 Create a Telegram bot

1. Open Telegram, search for **@BotFather**
2. Send `/newbot`
3. Choose a name (e.g., "My OpenClaw")
4. Choose a username (e.g., `my_openclaw_bot`)
5. Copy the bot token BotFather gives you

5.2 Add the token to OpenClaw

```
openclaw config set `
  channels.telegram.botToken `
  "YOUR_BOT_TOKEN_HERE"
openclaw config set `
  channels.telegram.dmPolicy pairing
```

Step 6: Set Up WhatsApp (Optional)

```
openclaw config set `
  channels.whatsapp.dmPolicy allowlist
openclaw config set `
  channels.whatsapp.allowFrom `
  '"+YOUR_PHONE_NUMBER"'
```

Then link your phone:

```
openclaw channels add --channel whatsapp
```

A QR code appears. Scan it with your phone (WhatsApp > Settings > Linked Devices > Link a Device). It expires after ~60 seconds — press Enter to regenerate.

Step 7: Start OpenClaw

```
openclaw gateway
```

You should see output confirming:

- Gateway listening on `127.0.0.1:18789`
- Telegram channel connected

Pair your Telegram account

Send any message to your bot in Telegram. OpenClaw will auto-approve your first device. If it doesn't, approve manually:

```
openclaw pairing list telegram
openclaw pairing approve telegram YOUR_CODE
```

Run on startup (optional)

```
openclaw daemon install
```

Access the web dashboard

Get your auto-generated gateway token:

```
Select-String '"token"' `
  "$env:USERPROFILE\.openclaw\openclaw.json"
```

Copy the token value, open http://localhost:18789 in your browser and paste it to log in.

Step 8: Test It

Send a message to your Telegram bot:

> *"What files are on my desktop?"*

Try a few more:

Message	What it does
"What's my IP address?"	Runs a shell command
"Summarize the PDF on my desktop"	Reads a local file
"What's the weather in Paris?"	Browses the web

If something fails: run `openclaw status` to check what's wrong.

Keeping It Running

To prevent sleep: **Settings > System > Power & battery** > set sleep to **Never** when plugged in.

Update OpenClaw:

```
npm install -g openclaw@latest
openclaw gateway --force
```

Logs:

```
openclaw logs -f
```

Troubleshooting

Problem	Solution
`openclaw : The term 'openclaw' is not recognized`	Restart PowerShell, or run `npm list -g openclaw`
`Invalid bearer token`	If using Anthropic: check your API key in `~\.openclaw\.env`. If using OpenAI subscription: re-run `openclaw auth login --provider openai-codex`
Telegram bot not responding	Check token: `Invoke-RestMethod "https://api.telegram.org/bot<TOKEN>/getMe"`
Gateway won't start	Run `openclaw config get gateway.mode` — must be `local`

Chapter 2: Skills and ClawHub

> ***Goal:*** *Understand how OpenClaw skills work, install useful ones safely, and build your first custom skill.*

What Are Skills?

Skills are **instruction files** that teach OpenClaw how to do specific things. They're not code plugins -- they're markdown documents that tell the AI agent *when* to activate and *what steps to follow*.

Think of skills as **playbooks**. The AI reads them and follows the instructions using the tools it already has (file access, shell commands, browser, etc.).

How Skills Work

The loading chain

When OpenClaw starts, it loads skills from these locations (highest priority first):

1. `<workspace>\skills\` -- project-specific skills
2. `~\.openclaw\skills\` -- your personal skills
3. Bundled skills (~53 that ship with OpenClaw)

Automatic vs manual activation

Mode	How it works	Example
Automatic	OpenClaw reads your message and picks the best matching skill based on its description	"What's the weather?" triggers the weather skill

Slash command	You type `/skill-name` to trigger it directly	`/summarize report.pdf`

Skills don't grant permissions

Skills are just instructions. If your security config blocks shell commands, a skill that needs `exec` will fail when it tries to run anything. Your tool policy (from Chapter 1) is always the gatekeeper.

Installing Skills

From ClawHub (the community marketplace)

```
clawhub search "web research"
openclaw skills install web-search
openclaw skills list
openclaw skills update --all
```

From a GitHub repo

```
git clone https://github.com/someone/their-skill.git
"$env:USERPROFILE\.openclaw\skills\their-skill"
```

Uninstall

```
clawhub uninstall skill-name
```

After any change, start a new session (`/new`) or restart the gateway for skills to reload.

Vetting Skills for Safety

This is not optional. **824+ malicious skills** were found on ClawHub. The campaign ("ClawHavoc") delivered malware that stole crypto wallets, SSH keys, and browser passwords.

Practical vetting workflow

```
# 1. Install the skill
openclaw skills install suspicious-skill

# 2. DON'T use it yet -- read it first
Get-Content ~\.openclaw\skills\suspicious-skill\SKILL.md

# 3. Look for anything that downloads or executes external code
Select-String -Path ~\.openclaw\skills\suspicious-skill\* -Pattern
"curl|wget|npx|pip install|eval|exec" -Recurse

# 4. Check if it tries to access sensitive paths
```

```
Select-String -Path ~\.openclaw\skills\suspicious-skill\* -Pattern
"\.ssh|\.env|keychain|wallet|password" -Recurse

# 5. If anything looks off, remove it
Remove-Item -Recurse -Force ~\.openclaw\skills\suspicious-skill
```

Use an allowlist (strictest)

```
{
  "skills": {
    "allow_list_only": true,
    "allowed_skills": [
      "weather",
      "summarize",
      "github",
      "web-search"
    ]
  }
}
```

Security scanning tools

```
openclaw security audit --deep

npm install -g openclaw-security-monitor
openclaw-security-monitor scan
```

Recommended Safe Skills

These are **bundled skills** (ship with OpenClaw) or from the **openclaw-community** GitHub organization:

Skill	What it does
`github`	Interact with GitHub repos, issues, PRs
`summarize`	Summarize files, articles, URLs
`weather`	Check weather for any location
`obsidian`	Read/write to your Obsidian vault
`web-search`	Search the web and summarize results

Restricting bundled skills

If you don't want all 53 active (they add to context size):

```
{
  "skills": {
    "allowBundled": ["github", "summarize", "weather", "web-search"]
  }
}
```

Enabling and Disabling Skills

```
{
  "skills": {
    "entries": {
      "weather": {
        "enabled": true
      },
      "some-sketchy-skill": {
        "enabled": false
      }
    }
  }
}
```

Configuring skill secrets

Some skills need API keys. Configure them per-skill so they don't leak into chat history:

```
{
  "skills": {
    "entries": {
      "twitter-poster": {
        "enabled": true,
        "env": {
          "TWITTER_API_KEY": "your-key-here",
          "TWITTER_API_SECRET": "your-secret-here"
        }
      }
    }
  }
}
```

Anatomy of a Skill

A skill is a **folder with a `SKILL.md` file**. That's the only required file.

```
my-skill\
  SKILL.md           # Required -- the instructions
  helper-script.ps1  # Optional -- supporting files
  templates\         # Optional -- templates, configs, etc.
```

The SKILL.md format

```
---
name: my_skill_name
description: "One-line description -- this is what triggers automatic activation"
user-invocable: true
metadata:
  openclaw:
    emoji: "wrench"
    os: ["win32"]
    requires:
      bins: ["curl", "jq"]
      env: ["MY_API_KEY"]
---

# My Skill Name

## What it does
Clear explanation of purpose.

## Workflow
1. First, do this
2. Then check that
3. Finally, output the result like this

## Guardrails
- Never do X without asking the user first
- Always validate Y before proceeding

## Failure handling
- If the API fails, report the error and suggest alternatives
```

Frontmatter fields

Field	Required	What it does
`name`	Yes	Unique identifier (snake_case)
`description`	Yes	One-liner -- used for automatic matching
`user-invocable`	No	Show as `/slash-command` (default: true)
`disable-model-invocation`	No	Manual-only, no auto-activation
`metadata.openclaw.os`	No	Restrict to specific OS (`"win32"` for Windows)
`metadata.openclaw.requires.bins`	No	Required binaries on PATH
`metadata.openclaw.requires.env`	No	Required env vars

Build Your First Custom Skill

Step 1: Create the skill directory

```
New-Item -ItemType Directory -Force ~\.openclaw\skills\git-status-check
```

Step 2: Write SKILL.md

Create `~\.openclaw\skills\git-status-check\SKILL.md` using the template above. Write instructions that tell the AI what to do step-by-step, what commands to run, and what guardrails to follow.

Step 3: Test it

```
openclaw gateway restart
```

Then message OpenClaw: "Check my git repos for uncommitted changes" -- automatic matching should pick up the skill.

Step 4: Iterate

Edit `SKILL.md`, start a new session (`/new`), test again. No compilation, no deployment needed.

Context cost

Every eligible skill adds ~100-200 tokens to OpenClaw's system prompt. 50 active skills = 5,000-10,000 tokens before the conversation starts. Keep your active skill set lean.

Chapter 3: Automating Tasks with Cron Jobs and Triggers

> ***Goal:*** *Make OpenClaw do things on its own -- recurring tasks, reminders, and reactions to external events -- without you having to ask every time.*

OpenClaw's Automation Systems

System	What it does	Analogy
Cron jobs	Run tasks on a schedule	Alarm clock
Webhooks	React to external events	Doorbell
Heartbeat	Periodic self-check	Pulse check

Cron Jobs: Scheduled Tasks

> ***Note:*** *In PowerShell, use the backtick (`` ` ``) for line continuation instead of backslash (`\`).*

Example 1: Morning briefing

```
openclaw cron add `
  --name "Morning briefing" `
  --cron "0 7 * * *" `
  --tz "Europe/Paris" `
  --session isolated `
  --message "Generate today's briefing: weather, calendar highlights, and top 3
emails." `
  --announce `
  --channel telegram `
  --to "YOUR_TELEGRAM_CHAT_ID"
```

Example 2: Weekly project summary

```
openclaw cron add `
  --name "Weekly summary" `
  --cron "0 18 * * 5" `
  --tz "Europe/Paris" `
  --session isolated `
  --message "Summarize this week: git commits across my projects, completed tasks,
and draft a short status update I can send to the team." `
  --model opus `
  --thinking high `
  --announce `
  --channel telegram `
  --to "YOUR_CHAT_ID"
```

Example 3: Persistent project monitor

```
openclaw cron add `
  --name "Project monitor" `
  --cron "0 */2 * * *" `
  --tz "Europe/Paris" `
  --session "session:project-monitor" `
  --message "Check the project repo for new issues, PRs, and CI failures. Compare
with your last check and only report what's new."
```

Because this uses a named session, OpenClaw remembers what it reported last time and only flags new items.

Schedule formats

Cron expression: `minute hour day-of-month month day-of-week`

```
--cron "0 7 * * *"       # Every day at 7am
--cron "0 9 * * 1"       # Every Monday at 9am
--cron "0 8-18 * * 1-5"  # Every hour during work hours, weekdays
```

One-shot:

```
--at "2026-04-01T14:00:00Z"   # Specific time
--at "20m"                     # Relative (20 minutes from now)
```

Always set `--tz` to your timezone (IANA format): `"Europe/Paris"`, `"America/New_York"`, etc.

Managing cron jobs

```
openclaw cron list
openclaw cron status
openclaw cron runs --id <jobId> --limit 20
openclaw cron run <jobId>           # Run manually
openclaw cron edit <jobId> --message "Updated prompt"
openclaw cron remove <jobId>
```

Session types

Type	Behavior	Use for
`isolated`	Fresh session each run	Independent tasks (briefings, summaries)
`main`	Runs in your main chat session	Reminders, nudges
`session:<name>`	Named persistent session	Monitoring tasks that build state over time

---

Heartbeat: The Background Pulse

The heartbeat is a periodic self-check that runs in your main session. It reviews a checklist and only speaks up if something needs attention.

In `~\.openclaw\openclaw.json`:

```
{
  "agents": {
    "defaults": {
      "heartbeat": {
        "every": "30m",
        "activeHours": { "start": "08:00", "end": "22:00" }
      }
    }
  }
}
```

Define what it checks in your AGENTS.md:

```
# Heartbeat checklist
- Check email for urgent messages (reply if critical)
- Review calendar for events in next 2 hours
- If a background task finished, summarize results
```

Webhooks: Reacting to External Events

Webhooks let external services trigger OpenClaw.

In `~\.openclaw\openclaw.json`:

```
{
  "hooks": {
    "enabled": true,
    "token": "your-secret-webhook-token",
    "path": "/hooks"
  }
}
```

Generate a token:

```
openssl rand -hex 32
```

Trigger examples

Wake the main session:

```
Invoke-RestMethod -Method Post -Uri "http://127.0.0.1:18789/hooks/wake" `
  -Headers @{ "Authorization" = "Bearer YOUR_WEBHOOK_TOKEN"; "Content-Type" =
"application/json" } `
  -Body '{"text": "New urgent email from boss@company.com", "mode": "now"}'
```

Run an isolated agent task:

```
$body = @{
    message = "A new CSV was uploaded to ~/Downloads/report.csv. Analyze it and send
a summary."
    name = "CSV Analyzer"
    channel = "telegram"
    to = "YOUR_CHAT_ID"
} | ConvertTo-Json

Invoke-RestMethod -Method Post -Uri "http://127.0.0.1:18789/hooks/agent" `
  -Headers @{ "Authorization" = "Bearer YOUR_WEBHOOK_TOKEN"; "Content-Type" =
"application/json" } `
  -Body $body
```

> ***Note:*** *Webhooks require your machine to be reachable from the internet. Use Tailscale Funnel or a reverse proxy if needed.*

Safety Considerations

- **Start with `--session isolated`** -- cron jobs in your main session can clutter your chat
- **Don't over-schedule** -- every cron job uses AI tokens

- **Use `maxConcurrentRuns: 1`** -- prevents resource exhaustion
- **Webhook tokens are secrets** -- treat them like passwords
- **Test manually first** -- run `openclaw cron run <jobId>` before trusting the schedule

Chapter 4: Connecting More Channels

> ***Goal:*** *Connect OpenClaw to Discord, Slack, and Signal so you can reach your agent from anywhere.*

How Channels Work

Every channel uses the same pattern:

1. Create a bot/app on the platform
2. Add the credentials to OpenClaw's config
3. Set a DM policy (who can talk to your agent)
4. Pair your account

All channels share these DM policies:

Policy	Who can message	Best for
`pairing` (default)	Anyone can request, you approve	Personal use
`allowlist`	Only listed users	Tight control
`open`	Anyone	Public bots (risky)

Discord

Built-in -- no plugin needed.

1. Create a Discord bot

1. Go to the Discord Developer Portal
2. Click **New Application**, give it a name
3. Go to **Bot** tab:
   - Click **Reset Token** and copy it
   - Enable **Message Content Intent**
   - Enable **Server Members Intent**
4. Go to **OAuth2 > URL Generator**:
   - Scopes: `bot` , `applications.commands`
   - Permissions: View Channels, Send Messages, Read Message History, Embed Links, Attach Files
5. Copy the generated URL, open it, and invite the bot to your server

2. Configure OpenClaw

```
Add-Content -Path "$env:USERPROFILE\.openclaw\.env" -Value
"DISCORD_BOT_TOKEN=YOUR_BOT_TOKEN"
```

```
{
  "channels": {
    "discord": {
      "enabled": true,
      "token": {
        "source": "env",
        "provider": "default",
        "id": "DISCORD_BOT_TOKEN"
      },
      "dmPolicy": "pairing"
    }
  }
}
```

Restart OpenClaw, then DM your bot on Discord. Approve the pairing:

```
openclaw pairing list discord
openclaw pairing approve discord YOUR_CODE
```

3. Group chat behavior

In servers, the bot only responds when **@mentioned** by default. Set `requireMention: false` in the guild config to respond to all messages.

Gotchas

- Without **Message Content Intent**, the bot can't see message text in servers
- Bot-to-bot messages are ignored by default (prevents loops)

Slack

Built-in. Supports **Socket Mode** (recommended) and HTTP mode.

1. Create a Slack app

1. Go to api.slack.com/apps > **Create New App** > **From scratch**
2. **Socket Mode:** Enable it, create an App-Level Token with `connections:write` scope -- copy the `xapp-...` token
3. **OAuth & Permissions:** Add bot scopes: `chat:write`, `channels:history`, `im:history`, `channels:read`, `im:read`, `users:read`, `files:read`
4. **Event Subscriptions:** Subscribe to: `app_mention`, `message.channels`, `message.im`
5. **Install** the app to your workspace -- copy the Bot Token (`xoxb-...`)

2. Configure OpenClaw

```
Add-Content -Path "$env:USERPROFILE\.openclaw\.env" -Value
"SLACK_APP_TOKEN=xapp-..."
Add-Content -Path "$env:USERPROFILE\.openclaw\.env" -Value
"SLACK_BOT_TOKEN=xoxb-..."
```

```
{
  "channels": {
    "slack": {
      "enabled": true,
      "mode": "socket",
      "appToken": "xapp-...",
      "botToken": "xoxb-...",
      "dmPolicy": "pairing"
    }
  }
}
```

Gotchas

- Slack reserves `/status` -- use `/agentstatus` instead
- Multi-workspace requires separate app installs

Signal

Built-in, but requires **signal-cli**.

> ***Important:*** *Registering signal-cli with your phone number will* ***log out your main Signal app****. Use a dedicated number or link as a secondary device.*

1. Install signal-cli

1. Download from signal-cli GitHub releases
2. Extract to `C:\tools\signal-cli\`
3. Add to PATH:

```
[Environment]::SetEnvironmentVariable("Path", $env:PATH + ";C:\tools\signal-cli",
"User")
```

2. Register or link

Link to existing account (won't log you out):

```
signal-cli link -n "OpenClaw"
```

Scan the QR code with Signal (Settings > Linked Devices).

Register a new number:

```
signal-cli -a +YOUR_BOT_NUMBER register
```

3. Configure OpenClaw

```
{
  "channels": {
```

```
    "signal": {
      "enabled": true,
      "account": "+YOUR_BOT_NUMBER",
      "dmPolicy": "pairing",
      "allowFrom": ["+YOUR_PERSONAL_NUMBER"]
    }
  }
}
```

Gotchas

- signal-cli must be kept updated as Signal's server APIs change
- No rich formatting, 8MB media limit

---

iMessage (Not Available on Windows)

iMessage requires a Mac. Use Telegram, WhatsApp, or Signal as alternatives.

Channel Comparison

Channel	Built-in	Rich formatting	Streaming	Media	Setup difficulty
Telegram	Yes	Markdown	Yes	Yes	Easy
WhatsApp	Yes	Limited	No	Yes	Easy (QR scan)
Discord	Yes	Markdown	Yes	Yes	Medium
Slack	Yes	Block Kit	Yes + native	Yes	Medium
Signal	Yes	None	No	8MB limit	Medium (signal-cli)

Chapter 5: Security and Sharing Access

> ***Goal:*** *Lock down your OpenClaw instance properly and optionally share access with trusted people.*

Tool Profiles

Profile	File access	Shell commands	Browser	Automation	Best for
`full`	Everything	Everything	Yes	Yes	Solo power user
`coding`	Yes	Yes	No	Sessions	Development work
`messaging`	No	No	No	No	Chat-only assistant

`minimal`	No	No	No	No	Status checks only

```
{
  "tools": {
    "profile": "full"
  }
}
```

Exec Security: Controlling Shell Commands

Security mode

Mode	Behavior
`deny`	No shell commands allowed
`allowlist`	Only pre-approved commands
`full`	Any command can run

Ask mode

Mode	Behavior
`always`	You approve every command before it runs
`on-miss`	Only asks for new/unknown commands
`off`	No approval needed (dangerous)

Recommended configs

Paranoid:

```
{ "tools": { "exec": { "security": "deny" } } }
```

Cautious (recommended):

```
{ "tools": { "exec": { "security": "allowlist", "ask": "always" } } }
```

Trusting (solo use only):

```
{ "tools": { "exec": { "security": "full", "ask": "on-miss" } } }
```

Filesystem Restrictions

```
{
  "tools": {
    "fs": {
      "workspaceOnly": true
    }
  }
}
```

`workspaceOnly: false` = access any file. `true` = workspace directory only. Set `true` when sharing access.

Elevated Permissions (Administrator)

Keep disabled unless you have a specific use case:

```
{
  "tools": {
    "elevated": {
      "enabled": false
    }
  }
}
```

Network Security

Gateway binding

```
{ "gateway": { "bind": "loopback" } }
```

Bind mode	Listens on	Exposed to
`loopback`	127.0.0.1	Your machine only
`tailnet`	Tailscale IP	Your tailnet only
`lan`	0.0.0.0	Your entire network

Authentication

Use token auth. Generate a strong token:

```
openssl rand -hex 32
```

```
{
  "gateway": {
    "auth": {
      "mode": "token",
```

```
      "rateLimit": {
        "maxAttempts": 10,
        "windowMs": 60000,
        "lockoutMs": 300000
      }
    }
  }
}
```

Session Isolation

```
{
  "session": {
    "dmScope": "per-channel-peer",
    "identityLinks": {
      "laurenz": ["telegram:123456789", "whatsapp:+33612345678"]
    }
  }
}
```

Scope	Behavior
`main`	Everyone shares one session
`per-peer`	Separate session per person
`per-channel-peer`	Separate per person per channel (recommended)

Running the Security Audit

```
openclaw security audit
openclaw security audit --deep
openclaw security audit --fix
openclaw doctor
```

Known Attack Vectors

- **Malicious skills (Critical)** -- 824+ found on ClawHub. Vet every skill, use an allowlist.
- **Prompt injection (High)** -- Hidden instructions in messages. Use `pairing` or `allowlist` DM policies.
- **Data exfiltration via web_fetch (High)** -- Compromised agent POSTs your data externally. Deny `web_fetch` or sandbox with `network: "none"`.
- **Token theft (High)** -- Tokens stored in plaintext at `~\.openclaw\credentials\`. Restrict with `icacls`.

Setting file permissions

```
# Restrict .openclaw directory -- your user only
icacls "$env:USERPROFILE\.openclaw" /inheritance:r /grant:r "${env:USERNAME}:(OI)
(CI)(F)"

# Restrict .env file
icacls "$env:USERPROFILE\.openclaw\.env" /inheritance:r /grant:r "${env:USERNAME}:
(R,W)"

# Restrict credentials directory
icacls "$env:USERPROFILE\.openclaw\credentials" /inheritance:r /grant:r
"${env:USERNAME}:(OI)(CI)(F)"
```

Sharing with Family or a Small Team

OpenClaw is a **single-user tool**. You *can* share it with trusted people, but it's not a multi-tenant security boundary.

Set up separate agents per person

```
{
  "agents": {
    "list": [
      {
        "id": "laurenz",
        "workspace": "~\\.openclaw\\workspace-laurenz",
        "sandbox": { "mode": "off" },
        "tools": { "profile": "full" }
      },
      {
        "id": "family",
        "workspace": "~\\.openclaw\\workspace-family",
        "sandbox": { "mode": "all", "scope": "agent" },
        "tools": {
          "profile": "messaging",
          "allow": ["exec", "read"],
          "deny": ["write", "edit", "apply_patch", "browser"]
        }
      }
    ]
  }
}
```

Sharing via Tailscale

Tailnet-only (recommended):

```
{
  "gateway": {
    "bind": "loopback",
```

```
    "tailscale": { "mode": "serve" }
  }
}
```

Public access (careful):

```
{
  "gateway": {
    "bind": "loopback",
    "tailscale": { "mode": "funnel" },
    "auth": { "mode": "password", "password": "strong-shared-password" }
  }
}
```

Security Checklist

Solo use

- [ ] `gateway.bind` set to `loopback`
- [ ] Gateway token is 32+ hex characters
- [ ] `exec.ask` set to `always` or `on-miss`
- [ ] `elevated.enabled` set to `false`
- [ ] DM policy set to `pairing` on all channels
- [ ] `openclaw security audit` passes clean
- [ ] Config file permissions restricted with `icacls`
- [ ] No API keys in source control

Shared use (add these)

- [ ] Separate agents per user with isolated workspaces
- [ ] `session.dmScope` set to `per-channel-peer`
- [ ] Family/shared agents use `messaging` or custom profile
- [ ] Family/shared agents sandboxed (`sandbox.mode: "all"`)
- [ ] Shared agents have `write` , `edit` , `browser` denied

Hardened Config: Full Example

```
{
  "gateway": {
    "bind": "loopback",
    "auth": {
      "mode": "token",
      "rateLimit": {
        "maxAttempts": 10,
        "windowMs": 60000,
        "lockoutMs": 300000
      }
```

```
    }
  },
  "session": {
    "dmScope": "per-channel-peer"
  },
  "agents": {
    "list": [
      {
        "id": "personal",
        "workspace": "~\\.openclaw\\workspace-personal",
        "sandbox": { "mode": "off" },
        "tools": {
          "profile": "full",
          "exec": {
            "security": "full",
            "ask": "on-miss"
          },
          "elevated": { "enabled": false }
        }
      },
      {
        "id": "family",
        "workspace": "~\\.openclaw\\workspace-family",
        "sandbox": { "mode": "all", "scope": "session" },
        "tools": {
          "profile": "messaging",
          "allow": ["read"],
          "deny": ["write", "edit", "apply_patch", "browser", "group:automation"],
          "exec": { "security": "deny" },
          "elevated": { "enabled": false }
        }
      }
    ]
  },
  "bindings": [
    {
      "agentId": "personal",
      "match": { "channel": "telegram", "peer": { "kind": "direct", "id":
"YOUR_TELEGRAM_ID" } }
    },
    {
      "agentId": "personal",
      "match": { "channel": "whatsapp", "peer": { "kind": "direct", "id":
"+YOUR_NUMBER" } }
    },
    {
      "agentId": "family",
      "match": { "channel": "whatsapp", "peer": { "kind": "direct", "id":
"+PARTNER_NUMBER" } }
    }
  ],
  "channels": {
```

```
    "telegram": { "dmPolicy": "pairing" },
    "whatsapp": { "dmPolicy": "pairing" }
  },
  "skills": {
    "allow_list_only": true,
    "allowed_skills": ["weather", "summarize", "github", "web-search",
"git_status_check"]
  },
  "cron": {
    "enabled": true,
    "maxConcurrentRuns": 1
  }
}
```