Chapter 1: Installing OpenClaw on Your Computer

> ***Goal:*** *Get OpenClaw running on your Linux 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 **Linux** (Ubuntu/Debian)
- 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

```
curl -fsSL https://deb.nodesource.com/setup_24.x | sudo -E bash -
sudo apt install -y nodejs
```

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:

```
mkdir -p ~/.openclaw
nano ~/.openclaw/.env
```

Add this line (paste your actual key):

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

Save and exit (Ctrl+X, then Y, then Enter). 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. 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:

```
grep '"token"' ~/.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

Prevent sleep:

```
sudo systemctl mask sleep.target suspend.target
```

Update OpenClaw:

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

Logs:

```
openclaw logs -f
```

Troubleshooting

Problem	Solution
`command not found: openclaw`	Restart your terminal, 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: `curl 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 markdown documents that tell the AI agent *when* to activate and *what steps to follow*, 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 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`

Control per skill:

- `user-invocable: true` — appears as a slash command
- `disable-model-invocation: true` — only activates when you explicitly call it

Skills don't grant permissions

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

Installing Skills

From ClawHub

```
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 ~/.openclaw/skills/their-skill
```

Uninstall

```
rm -rf ~/.openclaw/skills/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 "ClawHavoc" campaign) delivering malware that stole crypto wallets, SSH keys, and browser passwords.

Practical vetting workflow

```
# 1. Install but DON'T use yet — read it first
openclaw skills install suspicious-skill
cat ~/.openclaw/skills/suspicious-skill/SKILL.md

# 2. Look for downloads or external execution
grep -ri "curl\|wget\|npx\|pip install\|eval\|exec" ~/.openclaw/skills/suspicious-
```

```
skill/

# 3. Check for sensitive path access
grep -ri "\.ssh\|\.env\|keychain\|wallet\|password" ~/.openclaw/skills/suspicious-skill/

# 4. If anything looks off, remove it
rm -rf ~/.openclaw/skills/suspicious-skill
```

Use an allowlist (strictest)

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

Security scanning

```
openclaw security audit --deep
```

Recommended Safe Skills

Bundled skills (already installed):

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

Restrict to only the ones you use (reduces context cost):

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

Enabling, Disabling, and Configuring Skills

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

Skill env vars are injected only during that skill's execution.

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.sh   # 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: "🔧"
    os: ["darwin", "linux"]
    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
```

If a skill declares requirements that aren't met (missing binary, missing env var, wrong OS), OpenClaw silently excludes it.

Build Your First Custom Skill

Step 1: Create the skill

```
mkdir -p ~/.openclaw/skills/git-status-check
```

Write `~/.openclaw/skills/git-status-check/SKILL.md` following the template above. Include a clear `description` (triggers automatic matching), a step-by-step `Workflow`, and `Guardrails` for safety.

Step 2: Test it

```
openclaw gateway restart
```

Then message OpenClaw:

> *"Check my git repos for uncommitted changes"*

Step 3: Iterate

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

Context cost

Every eligible skill adds ~100-200 tokens to the 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.*

Cron Jobs: Scheduled Tasks

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"
```

Schedule formats

Recurring (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
--tz "Europe/Paris"       # Always set your timezone
```

One-shot:

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

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	Reminders, nudges
`session:<name>`	Context carries between runs	Monitoring tasks that build state

---

Practical Examples

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"
```

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.

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
```

If everything is fine, it silently responds `HEARTBEAT_OK`. If something needs attention, it speaks up.

Webhooks: Reacting to External Events

Webhooks let external services trigger OpenClaw.

Enable webhooks

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

Generate a secure token:

```
openssl rand -hex 32
```

Trigger examples

Wake the main session:

```
curl -X POST http://127.0.0.1:18789/hooks/wake \
  -H 'Authorization: Bearer YOUR_WEBHOOK_TOKEN' \
  -H 'Content-Type: application/json' \
  -d '{"text": "New urgent email from boss@company.com", "mode": "now"}'
```

Run an isolated agent task:

```
curl -X POST http://127.0.0.1:18789/hooks/agent \
  -H 'Authorization: Bearer YOUR_WEBHOOK_TOKEN' \
  -H 'Content-Type: application/json' \
  -d '{
    "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"
  }'
```

> *Webhooks require your machine to be reachable. Use Tailscale Funnel or a reverse proxy if needed.*

Safety Considerations

- **Start with `--session isolated`** — cron jobs in your main session clutter context
- **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** — `openclaw cron run <jobId>` before trusting the schedule

Troubleshooting

Problem	Solution
Cron job not running	Check `openclaw cron status` and verify timezone with `--tz`
Job runs but no output in chat	Verify `--announce --channel <channel> --to <id>` are set
High token usage	Reduce job frequency or switch to a cheaper model

Chapter 4: Connecting More Channels

> ***Goal:*** *Connect OpenClaw to Discord, Slack, Signal, and iMessage 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

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

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** and **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

```
echo 'DISCORD_BOT_TOKEN=YOUR_BOT_TOKEN' >> ~/.openclaw/.env
```

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

Restart OpenClaw, DM your bot, then approve:

```
openclaw pairing approve discord YOUR_CODE
```

In servers, the bot only responds when @mentioned by default.

Slack

1. Create a Slack app

1. Go to api.slack.com/apps > **Create New App** > **From scratch**
2. Enable **Socket Mode**, 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`, `files:write`
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

```
echo 'SLACK_APP_TOKEN=xapp-...' >> ~/.openclaw/.env
echo 'SLACK_BOT_TOKEN=xoxb-...' >> ~/.openclaw/.env
```

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

In channels, the bot responds only when @mentioned.

Signal

Requires **signal-cli** (a command-line Signal client).

> *Registering signal-cli with your phone number will **log out your main Signal app**. Use a dedicated number or link instead.*

1. Install signal-cli

```
VERSION=$(curl -Ls -o /dev/null -w %{url_effective} \
  https://github.com/AsamK/signal-cli/releases/latest | sed -e 's/^.*\/v//')
curl -L -O "https://github.com/AsamK/signal-
cli/releases/download/v${VERSION}/signal-cli-${VERSION}-Linux-native.tar.gz"
sudo tar xf "signal-cli-${VERSION}-Linux-native.tar.gz" -C /opt
sudo ln -sf /opt/signal-cli /usr/local/bin/
```

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

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

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

3. Configure OpenClaw

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

```
  }
}
```

---

iMessage (macOS only)

Requires a **Mac** running macOS 13+ and the **BlueBubbles** server app.

1. Download from bluebubbles.app and install
2. Enable the **Web API**, set a password, note the server URL

```
{
  "channels": {
    "bluebubbles": {
      "enabled": true,
      "serverUrl": "http://localhost:1234",
      "password": "your-bluebubbles-password",
      "webhookPath": "/bluebubbles-webhook",
      "dmPolicy": "pairing"
    }
  }
}
```

Channel Comparison

Channel	Built-in	Rich formatting	Streaming	Setup difficulty
Telegram	Yes	Markdown	Yes	Easy
WhatsApp	Yes	Limited	No	Easy (QR scan)
Discord	Yes	Markdown	Yes	Medium
Slack	Yes	Block Kit	Yes	Medium
Signal	Yes	None	No	Medium
iMessage	Yes	None	No	Medium (macOS only)

Troubleshooting

Problem	Solution
Bot not responding after setup	Check `openclaw status` and verify the token in `.env`
Pairing code not appearing	Ensure `dmPolicy` is set to `pairing`, not `disabled`
Messages visible but no reply	Check tool permissions — the skill may need `exec` access

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	Best for
`full`	Everything	Everything	Yes	Solo power user
`coding`	Yes	Yes	No	Development work
`messaging`	No	No	No	Chat-only assistant
`minimal`	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`	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" } } }
```

Gateway and Authentication

Use token auth. Generate a strong token:

```
openssl rand -hex 32
```

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

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

Filesystem and Elevated Permissions

```
{
  "tools": {
    "fs": { "workspaceOnly": true },
    "elevated": { "enabled": false }
  }
}
```

`workspaceOnly: false` for personal use (you want file access). Set `true` when sharing.

Keep `elevated.enabled: false` unless you have a specific use case.

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)

Known Attack Vectors

- **Malicious skills** (Critical) — 824+ found on ClawHub stealing SSH keys, crypto wallets, browser passwords. Vet every skill, use an allowlist.
- **Prompt injection** (High) — Hidden instructions in messages. Use `pairing` or `allowlist` DM policies, avoid `open`.
- **Data exfiltration via web_fetch** (High) — Compromised agent POSTs data externally. Deny `web_fetch` or sandbox with `network: "none"`.
- **Token theft** (High) — Gateway tokens stored in plaintext at `~/.openclaw/credentials/`. Set `chmod 600` on config files, `chmod 700` on the directory.

Sharing with Family or a Small Team

OpenClaw is a single-user tool. You *can* share it with trusted people, but shared instances require mutual trust — this is not a multi-tenant security boundary.

Set up separate agents per person

```
{
  "agents": {
    "list": [
      {
        "id": "personal",
        "workspace": "~/.openclaw/workspace-personal",
        "tools": { "profile": "full" }
      },
      {
        "id": "family",
        "workspace": "~/.openclaw/workspace-family",
```

```
        "sandbox": { "mode": "all", "scope": "session" },
        "tools": {
          "profile": "messaging",
          "allow": ["read"],
          "deny": ["write", "edit", "apply_patch", "browser"],
          "exec": { "security": "deny" }
        }
      }
    ]
  },
  "bindings": [
    {
      "agentId": "personal",
      "match": { "channel": "telegram", "peer": { "kind": "direct", "id":
"YOUR_TELEGRAM_ID" } }
    },
    {
      "agentId": "family",
      "match": { "channel": "whatsapp", "peer": { "kind": "direct", "id":
"+PARTNER_NUMBER" } }
    }
  ]
}
```

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: `chmod 600 ~/.openclaw/openclaw.json`
- [ ] Directory permissions: `chmod 700 ~/.openclaw/`

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
- [ ] 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
  }
}
```