Chapter 1: Installing OpenClaw on Your Mac

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

What You'll Have

- OpenClaw running on your Mac
- 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 Mac running **macOS 13 (Ventura)** or later
- Homebrew installed
- 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

```
brew install node@24
```

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: System Settings > Energy > Prevent automatic sleeping when plugged in.

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 not code plugins — they're markdown documents that tell the AI agent *when* to activate and *what steps to follow*.

A skill might tell OpenClaw:

- "When the user asks about the weather, use the `curl` command to hit this API and format the result like this"
- "When the user says /summarize, read the given file and produce a structured summary"
- "When the user asks to post on Twitter, use the X API with these steps"

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)

A skill in a higher-priority location overrides one with the same name below it.

Automatic vs manual activation

Skills activate in two ways:

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`

You can control this 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

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

Installing Skills

From ClawHub (the community marketplace)

```
# Search for skills
clawhub search "web research"

# Install one
openclaw skills install web-search

# List what's installed
openclaw skills list

# Update all skills
openclaw skills update --all
```

From a GitHub repo

Paste a GitHub link directly into the chat:

> *"Install the skill from https://github.com/someone/their-openclaw-skill"*

Or clone manually:

```
git clone https://github.com/someone/their-skill.git ~/.openclaw/skills/their-skill
```

Uninstall

```
clawhub uninstall skill-name
```

Or just delete the folder:

```
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. **341 malicious skills** were found on ClawHub in early 2026, and the count grew to **824+ malicious skills** across 12 publisher accounts. The campaign (dubbed "ClawHavoc") delivered malware that stole crypto wallets, SSH keys, and browser passwords.

The vetting checklist

Before installing any community skill, run through this:

Check	What to look for	Red flag
Source code	Does it have a public GitHub repo?	No repo = don't install
Commit history	Real development pattern?	Created last week with 2 commits
Publisher	Who made it? Check their GitHub profile	New account, no other repos
GitHub stars	Community validation	<50 stars warrants extra scrutiny
Read the files	Open the skill folder and read everything	Downloads from unknown URLs, obfuscated one-liners
Permissions needed	What does the frontmatter require?	A "timer" skill wanting filesystem + network access
Community discussion	Search GitHub Issues, Discord, Reddit	Reports of suspicious behavior

Practical vetting workflow

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

# 2. DON'T use it yet — read it first
cat ~/.openclaw/skills/suspicious-skill/SKILL.md

# 3. Look for anything that downloads or executes external code
grep -ri "curl\|wget\|npx\|pip install\|eval\|exec" ~/.openclaw/skills/suspicious-skill/

# 4. Check if it tries to access sensitive paths
grep -ri "\.ssh\|\.env\|keychain\|wallet\|password" ~/.openclaw/skills/suspicious-skill/
```

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

Use an allowlist (strictest)

Lock down to only approved skills in `~/.openclaw/openclaw.json`:

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

Security scanning tools

```
# Built-in security audit (also checks skills)
openclaw security audit --deep

# Community scanner (detects ClawHavoc, AMOS malware, etc.)
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. They're maintained by the core team and safe to use.

Bundled (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

You have ~53 bundled skills active by default. List them:

```
openclaw skills list
```

Restricting bundled skills

If you don't want all 53 active (they add to context size), restrict to the ones you use:

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

Enabling and Disabling Skills

Toggle skills in `~/.openclaw/openclaw.json`:

```
{
  "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"
        }
      }
    }
  }
}
```

The env vars are injected only during that skill's execution, then restored afterward.

Anatomy of a Skill

A skill is just 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.
```

Build Your First Custom Skill

```
mkdir -p ~/.openclaw/skills/my-skill
```

Create `~/.openclaw/skills/my-skill/SKILL.md` using this structure:

```
---
name: my_skill_name
description: "One-line description — triggers automatic activation"
user-invocable: true
metadata:
  openclaw:
    os: ["darwin"]
    requires:
      bins: ["curl"]
      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
```

Test it:

```
openclaw gateway restart
```

Then message OpenClaw or use `/my_skill_name`. Edit the SKILL.md and start a new session (`/new`) to iterate.

> ***Context cost:*** *Every active skill adds ~100-200 tokens to the system prompt. Keep your active 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.*

Why Automate?

So far, OpenClaw only does things when you message it. But the real power comes when it acts **on its own schedule**:

- Morning briefing delivered to your Telegram every day at 7am
- Reminder 20 minutes before your next meeting
- Nightly summary of what happened today
- Monitoring your email inbox every hour for urgent messages
- Weekly report generated every Monday morning

OpenClaw has three 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

Your first cron job: a 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"
```

Every morning at 7:00 AM Paris time, OpenClaw will:

1. Spin up an isolated session (so it doesn't pollute your main chat)
2. Run the prompt
3. Send the result to your Telegram

Schedule formats

You can schedule jobs three ways:

Recurring (cron expression)

Standard cron format: `minute hour day-of-month month day-of-week`

```
# Every day at 7am
--cron "0 7 * * *"

# Every Monday at 9am
--cron "0 9 * * 1"

# Every hour during work hours (8am-6pm, weekdays)
--cron "0 8-18 * * 1-5"

# Every 15 minutes
--cron "*/15 * * * *"
```

Always set `--tz` to your timezone (IANA format):

```
--tz "Europe/Paris"
--tz "America/New_York"
--tz "Asia/Tokyo"
```

One-shot (specific time)

```
# At a specific time
--at "2026-04-01T14:00:00Z"

# Relative (20 minutes from now)
--at "20m"
```

One-shots run once. Add `--delete-after-run` to auto-cleanup.

Managing cron jobs

```
# List all jobs
openclaw cron list

# Check status
openclaw cron status

# View run history for a job
openclaw cron runs --id <jobId> --limit 20

# Run a job manually right now
openclaw cron run <jobId>

# Edit a job
openclaw cron edit <jobId> --message "Updated prompt" --model opus
```

```
# Delete a job
openclaw cron remove <jobId>
```

Session types

Type	Behavior	Use for
`isolated`	Fresh session each run, no memory of previous runs	Independent tasks (briefings, summaries)
`main`	Runs in your main chat session	Reminders, nudges
`session:<name>`	Named persistent session — context carries between runs	Monitoring tasks that build state over time

```
# Isolated (most common)
--session isolated

# Main session (for reminders)
--session main

# Persistent named session (for ongoing monitoring)
--session "session:project-tracker"
```

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 monitor (context carries over)

```
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. Unlike cron jobs, it doesn't execute a specific task — it reviews a checklist and only speaks up if something needs attention.

Enable the heartbeat

In `~/.openclaw/openclaw.json`:

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

Define what it checks

Create a heartbeat checklist in your AGENTS.md or session bootstrap:

```
# 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 idle for 8+ hours, send a brief check-in via Telegram
```

Every 30 minutes (during active hours), OpenClaw will run through this list. If everything is fine, it silently responds `HEARTBEAT_OK`. If something needs attention, it speaks up.

Heartbeat vs cron

	Heartbeat	Cron
Runs in	Main session	Isolated or named session
Purpose	"Anything need attention?"	"Do this specific thing"
Output	Only speaks if needed	Always produces output
Best for	Ambient monitoring	Scheduled deliverables

Webhooks: Reacting to External Events

Webhooks let external services trigger OpenClaw. Your email service, GitHub, or any app that can send HTTP requests can wake OpenClaw up.

Enable webhooks

In `~/.openclaw/openclaw.json`:

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

Generate a secure token:

```
openssl rand -hex 32
```

Trigger OpenClaw from outside

Wake the main session (e.g., new email arrived):

```
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 (e.g., process a file):

```
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"
  }'
```

Use case: GitHub webhook

Point a GitHub webhook to your OpenClaw instance to get notified about PRs, issues, and CI failures:

1. In your repo: Settings > Webhooks > Add webhook
2. URL: `http://your-machine:18789/hooks/github` (needs port forwarding or Tailscale)
3. Secret: your webhook token

4. Events: Issues, Pull requests, Check runs

OpenClaw can then triage, summarize, and even draft responses.

> ***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 and confuse context
- **Don't over-schedule** — every cron job uses AI tokens. Running something every minute will burn through your usage fast
- **Use** `maxConcurrentRuns: 1` — prevents resource exhaustion if a job takes longer than its interval
- **Webhook tokens are secrets** — treat them like passwords. Don't commit them to git
- **Test manually first** — run `openclaw cron run <jobId>` before trusting the schedule
- **Monitor disk** — run logs grow over time. Check `~/.openclaw/cron/runs/` periodically

Chapter 4: Connecting More Channels

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

How Channels Work

In Chapter 1, you set up Telegram (and maybe WhatsApp). But OpenClaw supports **26+ messaging platforms**, and you can run as many as you want simultaneously.

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)
`disabled`	Nobody	Channel off

> ***Security reminder:*** *`pairing` is the safe default. Never use `open` on a machine with real file access — anyone who finds your bot could control your computer.*

Discord

Discord is 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 the bot token to your environment:

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

Open the config file:

```
nano ~/.openclaw/openclaw.json
```

Add or update with:

```
{
  "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:

```
{
  "channels": {
    "discord": {
      "guilds": {
        "YOUR_SERVER_ID": {
          "requireMention": true
        }
      }
    }
  }
}
```

Set `requireMention: false` if you want the bot to respond to all messages in a server (noisy — use carefully).

Gotchas

- Without **Message Content Intent**, the bot can't see message text in servers
- Bot-to-bot messages are ignored by default (prevents loops)
- Discord rate-limits streaming previews quickly with multiple bots

---

Slack

Built-in — no plugin needed. Supports two modes: **Socket Mode** (easier, 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`
4. **Event Subscriptions:** Subscribe to: `app_mention`, `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"
    }
```

```
  }
}
```

3. Group chat behavior

In channels, the bot responds only when @mentioned:

```
{
  "channels": {
    "slack": {
      "channels": {
        "C123456": {
          "requireMention": true
        }
      }
    }
  }
}
```

Gotchas

- Slack reserves `/status` — use `/agentstatus` instead
- Native Slack commands are off by default (enable explicitly)
- Multi-workspace requires separate app installs with distinct webhook paths

---

Signal

Built-in, but requires installing **signal-cli** (a command-line Signal client).

> ***Important:*** *Registering signal-cli with your phone number will* ***log out your main Signal app****.*
> *Use a dedicated phone number for the bot.*

1. Install signal-cli

```
brew install signal-cli
```

2. Register or link

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

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

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

Option B: Register a new number (dedicated bot number):

```
signal-cli -a +YOUR_BOT_NUMBER register
# Follow the verification flow
```

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 (no markdown, no buttons)
- 8MB media size limit
- No read receipts in groups

---

iMessage via BlueBubbles

This is where macOS shines. Since you're running on a Mac, you have native access to iMessage — something no other platform can offer. The **BlueBubbles** server app bridges iMessage to OpenClaw, letting you control your AI agent through the same app you use to text friends and family.

1. Install BlueBubbles

Download from bluebubbles.app and install on your Mac. You can also install via Homebrew:

```
brew install --cask bluebubbles
```

1. Open BlueBubbles, complete setup
2. Enable the **Web API**
3. Set a password
4. Note the server URL (e.g., `http://localhost:1234`)

2. Configure OpenClaw

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

```
  }
}
```

3. What works

- Send/receive iMessages and SMS
- Tapbacks/reactions (requires Private API enabled in BlueBubbles)
- Reply threading
- Message effects (slam, loud, etc.)
- Edit and unsend (macOS 13+)
- Group chats, attachments, voice memos

This means you can message your OpenClaw agent from any Apple device — your iPhone, iPad, or another Mac — using the built-in Messages app. No third-party app needed on the client side.

4. Tips for the best experience

- **Enable the Private API** in BlueBubbles for full feature support (tapbacks, typing indicators, read receipts)
- **Keep BlueBubbles running** — it needs to stay open for the bridge to work. Add it to your Login Items (System Settings > General > Login Items)
- **iMessage on a dedicated Mac** — if you want 24/7 iMessage availability, a Mac mini running as your OpenClaw server is ideal

Gotchas

- Edit is broken on macOS Tahoe (26)
- Requires BlueBubbles to be running at all times
- Group icon updates may silently fail on Tahoe

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)
iMessage	Yes	None	No	Yes	Medium (BlueBubbles)

Recommendation for macOS personal use: Start with **Telegram** (easiest, richest features) and add **iMessage via BlueBubbles** (native to your Mac, no extra app needed on your phone). Add others as needed.

Chapter 5: Security and Sharing Access

> ***Goal:*** *Lock down your OpenClaw instance properly, understand what can go wrong, and optionally share access with trusted people (family, small team).*

Why Security Matters More Here

OpenClaw is not a chatbot — it runs on your Mac with **your permissions**. A misconfigured instance could let someone:

- Read your files (SSH keys, passwords, documents)
- Run commands (install software, delete data, mine crypto)
- Send messages as you (email, social media)
- Exfiltrate data (upload your files to an external server)

This chapter teaches you to control what OpenClaw can do, who can talk to it, and how to share it safely.

Tool Profiles: The Big Picture

Tool profiles control what OpenClaw is **allowed** to do. Think of them as permission levels.

Profile	File access	Shell commands	Browser	Automation	Best for
`full`	Everything	Everything	Yes	Yes	Solo power user who trusts the setup
`coding`	Yes	Yes	No	Sessions	Development work
`messaging`	No	No	No	No	Chat-only assistant
`minimal`	No	No	No	No	Status checks only

Set your profile in `~/.openclaw/openclaw.json`:

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

> ***Recommendation:*** *Start with* `full` *if you're the only user and want the full power. Switch to restrictive profiles when sharing access (see below).*

Exec Security: Controlling Shell Commands

The most powerful (and dangerous) tool is `exec` — it runs shell commands on your Mac. Three settings control it:

Security mode

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

Ask mode (approval prompts)

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 (safest):

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

Cautious (recommended for daily use):

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

Trusting (power user, solo use only):

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

```
  }
}
```

When OpenClaw wants to run a command and `ask` is enabled, it sends you a prompt in the chat:

> *"I'd like to run: `git status`. Allow once / Always allow / Deny?"*

Approvals are saved in `~/.openclaw/exec-approvals.json` so you don't have to re-approve commands you've already trusted.

Safe bins

Some basic commands are always safe (they only read stdin, can't do damage):

```
cut, uniq, head, tail, tr, wc
```

Never add interpreters (`python3` , `node` , `bash`) to safe bins — they can execute arbitrary code.

Filesystem Restrictions

Control what OpenClaw can read and write:

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

Setting	Behavior
`workspaceOnly: true`	Can only access the agent's workspace directory
`workspaceOnly: false`	Can access any file your user can access

For personal use, `false` is fine — you want OpenClaw to read your files. Set it to `true` when sharing access.

Elevated Permissions (Sudo)

By default, OpenClaw **cannot** use sudo or run as root:

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

If you enable it, restrict who can trigger it:

```
{
  "tools": {
    "elevated": {
      "enabled": true,
      "allowFrom": {
        "telegram": ["YOUR_TELEGRAM_ID"],
        "whatsapp": ["+YOUR_PHONE_NUMBER"]
      }
    }
  }
}
```

> ***Rule of thumb:*** *Keep elevated disabled unless you have a specific use case. Most things OpenClaw does don't need root.*

Network Security

Gateway binding

Your gateway should **never** be exposed to the internet directly:

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

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

Authentication

Use token auth. Generate a strong token and set it:

```
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"]
    }
  }
}
```

Use `per-channel-peer` for multi-user setups. Use `identityLinks` to link your accounts across channels.

Running the Security Audit

OpenClaw has a built-in audit tool:

```
# Standard audit
openclaw security audit

# Deep audit (probes the running gateway)
openclaw security audit --deep

# Auto-fix issues
openclaw security audit --fix
```

What it checks

- DM policies — are channels too open?
- Tool blast radius — can agents do too much?
- Exec approval drift — are dangerous commands auto-approved?
- Network exposure — is the gateway reachable from outside?
- Disk hygiene — are config files world-readable?
- Skill safety — any suspicious patterns?

Also run the doctor for general health:

```
openclaw doctor
```

Known Attack Vectors

- **Malicious skills (Critical):** Skills run with full privileges. 824+ malicious skills found on ClawHub. Vet every skill, use an allowlist.
- **Prompt injection (High):** Hidden instructions in messages. Use `pairing` / `allowlist` DM policies, avoid `open`.
- **Data exfiltration via web_fetch (High):** Compromised agent POSTs data externally. Deny `web_fetch` for shared instances.
- **Token theft (High):** Tokens stored in plaintext at `~/.openclaw/credentials/`. Set `chmod 600` on config, `chmod 700` on directory. Enable FileVault.

Sharing with Family or a Small Team

OpenClaw is designed as a **single-user tool**. The docs are explicit: "One host per user is the recommended pattern." But you *can* share it with trusted people if you set it up carefully.

> ***Important:*** *Shared instances require mutual trust. OpenClaw is not a multi-tenant security boundary. Don't share with people you wouldn't trust with your computer password.*

Set up separate agents per person

Each person gets their own agent with its own workspace and tool policy:

```
{
  "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

If family members want to use the web dashboard:

Tailnet-only (recommended):

```
{
  "gateway": {
    "bind": "loopback",
    "tailscale": { "mode": "serve" }
  }
}
```

Install Tailscale on their devices, add them to your tailnet. They access the dashboard via `https://your-machine-name/`.

Public access (careful):

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

> ***Funnel*** *exposes your gateway to the public internet over HTTPS. Only do this with password auth and restrictive tool policies.*

Security Checklist

Run through this before going live:

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
- [ ] `openclaw doctor` passes clean
- [ ] Config file permissions: `chmod 600 ~/.openclaw/openclaw.json`
- [ ] Directory permissions: `chmod 700 ~/.openclaw/`
- [ ] No API keys in source control
- [ ] FileVault enabled for full-disk encryption

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"`)

- [ ] `elevated.allowFrom` restricted to your accounts only
- [ ] Shared agents have `write`, `edit`, `browser` denied
- [ ] Tested: shared user cannot access your files or session

---

Hardened Config: Full Example

Here's a complete config for personal use with a shared family agent:

```
{
  "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
  }
}
```