The Tidy Clawbot Playbook

File Organization Best Practices for AI Agent Workspaces

ClawGuides by OpenClaw

2026-03-11

The Tidy Clawbot Playbook

File Organization Best Practices for AI Agent Workspaces

Table of Contents

1. Why It Matters

AI agents are productivity multipliers. A well-configured agent can research topics, write code, generate reports, manage projects, and automate workflows — all in a single session. But there's a side effect nobody warns you about: **file sprawl.**

Every task your agent completes produces output. Research gets saved as Markdown. Code gets written to project directories. Session logs accumulate daily. Temporary files pile up. Within a few weeks of active use, your workspace can look like a digital junk drawer.

This isn't just an aesthetic problem. It has real consequences:

- **You can't find things.** Was that competitor analysis in `research/` or `marketing/` or just sitting at the workspace root?
- **Agents get confused too.** When an agent scans your workspace for context, a disorganized structure means it wastes tokens processing irrelevant files and may reference outdated versions.
- **Collaboration breaks down.** If you work with multiple agents (QA, research, UX), they need a shared understanding of where things live.
- **Security risks grow.** Sensitive files (API keys, tokens, credentials) mixed into project folders are easier to accidentally commit or expose.

The fix isn't complicated, but it does require intention. This playbook gives you a proven system for keeping your AI agent workspace organized from day one.

---

2. The Hidden Folder Problem

What Happens by Default

When you install OpenClaw, it creates a configuration directory at:

```
~/.openclaw/
```

That `~/.` prefix means it's a **hidden directory** on macOS and Linux. In Finder on macOS, you won't see it unless you press `Cmd + Shift + Period` to toggle hidden file visibility. Most users never do this — meaning their entire AI workspace is invisible to them.

Inside `~/.openclaw/`, you'll typically find:

```
~/.openclaw/
├── openclaw.json          # Core configuration
├── gateway-token          # API authentication
├── workspace/             # Default working directory
│   ├── MEMORY.md
│   ├── projects/
│   └── ... everything your agent creates
```

Why This Is a Problem

1. **Out of sight, out of mind.** If you can't see your files in Finder, you forget they exist. Output piles up unchecked.
2. **Hard to browse.** Terminal-only access creates friction. You should be able to click through your agent's work like any other folder.
3. **Backup blind spots.** Many backup tools skip hidden directories by default. Your agent's entire body of work could be excluded from Time Machine or cloud sync.
4. **Mixing config with content.** Your gateway token and `openclaw.json` are config files that rarely change. Your projects, research, and memory files change constantly. They have different lifecycles and different security needs.

The Fix: Separate Config from Content

Keep core configuration in the hidden directory where it belongs:

```
~/.openclaw/
├── openclaw.json          # Config — stays hidden
├── gateway-token          # Auth — stays hidden
```

Move your working files to a visible location:

```
~/openclaw/workspace/      # Visible in Finder
├── websites/
├── agents/
├── marketing/
├── memory/
├── FILE-MAP.md
└── ...
```

This gives you the best of both worlds: sensitive config stays tucked away, while your actual work is front and center where you can see it, browse it, and back it up.

How to make the move: Update your OpenClaw configuration to point to the new workspace path. The exact method depends on your setup, but typically you'll update the `workspace` path in `openclaw.json` or set it via the CLI.

---

3. Building a FILE-MAP

What Is a FILE-MAP?

A FILE-MAP is a Markdown file (`FILE-MAP.md`) that lives at the root of your workspace. It serves as the canonical reference for where every type of file belongs. Think of it as a combination of:

- A **directory legend** (what each folder is for)
- A **filing rulebook** (where new files should go)
- A **migration tracker** (what still needs to be moved)

Anatomy of a FILE-MAP

A good FILE-MAP has four sections:

Section 1: Directory Table

```
## Root: ~/openclaw/workspace/

| Directory        | Purpose                                  |
        Examples                    |
|-----------------|------------------------------------------|-----------------------------|

| websites/       | All web properties — apps, sites         | clawguides, roleplay-
        studio  |
| agents/         | Agent configs — one subfolder per agent  | qa/, ux/, research/,
        files/  |
| marketing/      | Campaign assets and copy                 | wafd-money-market-
        campaign   |
| market-research/| Executive briefs, analysis, reports      | fdic-
        brief.html              |
| testing/        | QA reports and audit results             | qa-reports/, file-
        audits/    |
| memory/         | Daily session logs (YYYY-MM-DD.md)       |
        2026-03-11.md               |
| scratch/        | Temp work — cleaned every 30 days        | poc tests, one-off
        scripts   |
```

Section 2: Rules

```
## Rules (enforced by Files Agent)

1. Never use /tmp/ — use scratch/ for anything temporary
2. All websites/apps → websites/
3. Research + executive briefs → market-research/
4. QA and audit reports → testing/
5. Session logs → memory/
6. No loose files at workspace root — everything has a home
7. Agent configs → agents/<name>/
```

Section 3: Personal/System Spaces

```
## System Config: ~/.openclaw/

| File/Dir        | Purpose                    |
|-----------------|----------------------------|
| openclaw.json   | Core configuration         |
| gateway-token   | API authentication         |
```

Document any other workspaces or personal directories separately to prevent cross-contamination.

Section 4: Migration Queue

```
## Migration Queue (pending)

| Current Location              | Should Be                          |
        Notes                  |
|-------------------------------|------------------------------------|-----------------------|

| workspace/my-project/         | workspace/websites/my-project/     | Check git remotes
        first|
| Loose files at workspace root| Various                            | Files Agent to
        audit    |
```

This section tracks files that are in the wrong place but need careful handling (git repos, deployed projects, files with external references).

FILE-MAP Design Principles

1. **Organize by purpose, not by project.** All websites in `websites/`, all testing in `testing/`, all agent configs in `agents/`. This scales better than project-centric organization.

2. **Keep it flat.** 5-8 top-level directories is ideal. More than that and you're recreating the mess at a higher level.
3. **Name directories plainly.** `marketing/` not `mktg/`. `testing/` not `qa-and-audits-v2/`. Future-you will thank present-you.
4. **Include examples.** The Examples column in your directory table helps agents (and humans) pattern-match quickly.
5. **Version it.** Add a "Last updated" date at the top. When the structure evolves, the FILE-MAP evolves with it.

---

4. The Files Agent

What Is a Files Agent?

A Files Agent is a specialized AI agent whose sole responsibility is workspace organization. While your other agents focus on coding, research, QA, or marketing, the Files Agent focuses on *where things go*.

Why a Dedicated Agent?

You might think: "Can't I just tell my main agent to keep things organized?" You can try. But in practice:

- Agents are task-focused. When they're deep in a coding task, they optimize for completing the code, not for filing the output correctly.
- Organization is a different skill. It requires scanning the full workspace, understanding the FILE-MAP, comparing current state to desired state, and proposing changes.
- Separation of concerns works for agents too. Just like you wouldn't have your QA agent also write marketing copy, you shouldn't burden your coding agent with file management.

Files Agent Responsibilities

Responsibility	Description
Own the FILE-MAP	The Files Agent is the authority on FILE-MAP.md. It proposes updates when the structure needs to evolve.
Audit the workspace	Periodically scan for misplaced files, loose files at root, files in /tmp/, and duplicates.
Propose moves	Never auto-move or auto-delete. Always propose changes and wait for approval.
Enforce rules	Flag violations like files written to /tmp/ or loose files at workspace root.
Manage scratch/ cleanup	Flag files in scratch/ that haven't been modified in 30+ days.
Track migrations	Maintain the Migration Queue for files that need to be moved carefully.

Files Agent Configuration

Here's an example agent config (stored in `agents/files/`):

```
# Files Agent

## Identity
You are the Files Agent. You own the workspace FILE-MAP and enforce
file organization conventions.

## Rules
- Never auto-delete files. Always propose and wait for approval.
- Never move files with git repos without checking remotes first.
- The FILE-MAP.md is your source of truth. Consult it before every action.
- When you find a file that doesn't fit any category, propose a new
  category rather than forcing it into the wrong one.

## Triggers
- Run a workspace audit when asked
- Flag any file written to /tmp/
- Flag any loose file at workspace root
- Flag scratch/ files older than 30 days
```

The Golden Rule: Propose, Don't Act

The most important principle for a Files Agent: **it proposes changes, it doesn't execute them without approval.** Moving a file might break a git remote, a deployment path, a symlink, or another agent's reference. The Files Agent should always present its findings and wait for a human "go ahead."

---

5. Directory Structure Template

Copy this template and adapt it to your needs:

```
~/openclaw/workspace/
│
├── FILE-MAP.md              # Directory convention (this file)
├── MEMORY.md                # Cross-session persistent memory
│
├── websites/                # All web properties
│   ├── my-app/              #   Deployed web application
│   ├── my-site/             #   Marketing or docs site
│   └── installer/           #   Installer/distribution builds
│
├── agents/                  # Agent configurations
│   ├── qa/                  #   QA agent config + memory
│   ├── ux/                  #   UX/UI agent config
│   ├── research/            #   Research agent config
│   └── files/               #   Files agent config
│
├── marketing/               # Campaign assets and copy
│   └── campaign-name/       #   One subfolder per campaign
│
├── market-research/         # Executive briefs and analysis
│   ├── competitor-brief.md
│   └── industry-report.html
│
├── testing/                 # QA reports and audit results
│   ├── qa-reports/
│   └── file-audit-reports/
│
├── memory/                  # Daily session logs
│   ├── 2026-03-10.md
│   └── 2026-03-11.md
│
├── scratch/                 # Temporary work (30-day cleanup)
│   └── poc-experiment/
│
└── docs/                    # Internal documentation
    └── onboarding.md
```

Adapting the Template

- **If you don't do marketing:** Remove `marketing/`. Don't create empty folders "just in case."
- **If you have multiple clients:** Consider adding a `clients/` directory, or prefix project folders (e.g., `websites/client-a-site/`).
- **If you work solo:** You might not need `agents/` with multiple subfolders. Start with what you actually use.
- **If you generate lots of data:** Add a `data/` directory for datasets, CSVs, and database exports.

The template is a starting point. The FILE-MAP makes it yours.

---

6. Tips from the Community

From the AGENTS.md Movement

The `AGENTS.md` standard — now adopted by over 40,000 open-source projects — establishes a pattern of putting agent instructions in a single discoverable file at the project root. The same principle applies to FILE-MAP: one file, one location, one source of truth. As the `AGENTS.md` spec puts it: agents should have a "front door" — a single place to look for guidance.

From LangChain and LangGraph Users

The LangChain community recommends a modular directory structure that separates concerns: tools in one folder, state in another, agent logic in a third. This mirrors our approach of separating agent configs from their outputs. LangGraph specifically recommends keeping a configuration file (`langgraph.json`) at the project root — similar to how FILE-MAP.md sits at the workspace root.

From Reddit and Discord

Common advice from AI agent practitioners:

- **"Start organized or suffer later."** Multiple Reddit threads emphasize that retroactive organization is 10x harder than setting up structure from the start.
- **"Your agent's mess is your mess."** Users report that disorganized workspaces lead to degraded agent performance — the agent wastes context window scanning irrelevant files.
- **"Separate your secrets."** Keep API keys, tokens, and credentials isolated from working files. The hidden `~/.openclaw/` directory is actually perfect for this — just don't put your working files there too.
- **"Use scratch/, not /tmp/."** Files in `/tmp/` get wiped on reboot (or sooner). If your agent writes something to `/tmp/`, it's gone. A `scratch/` folder inside your workspace survives reboots and is included in backups.
- **"Version your structure."** As your needs change, your directory structure should evolve. The FILE-MAP tracks this evolution so nothing gets lost in the shuffle.

From Claude Cowork Users

Early adopters of Claude Cowork (Anthropic's desktop agent, launched January 2026) report that the agent is surprisingly good at proposing organizational structures for messy folders. The lesson: AI agents *can* organize files effectively — they just need clear rules and boundaries. That's exactly what a FILE-MAP provides.

---

7. Quick-Start Checklist

Use this checklist to set up your organized workspace in under 15 minutes:

- ☐ **Locate your current workspace.** Check `~/.openclaw/workspace/` — is that where your files are?
- ☐ **Create a visible workspace.** Make `~/openclaw/workspace/` (or your preferred visible path).
- ☐ **Move working files.** Copy projects, memory, and agent configs to the new location. Keep `openclaw.json` and auth tokens in `~/.openclaw/`.
- ☐ **Update your config.** Point OpenClaw to the new workspace path.
- ☐ **Create FILE-MAP.md.** Use the template in Section 5 as a starting point.
- ☐ **Create your top-level folders.** Only create the ones you'll actually use right now.
- ☐ **Add a scratch/ folder.** With a 30-day cleanup policy noted in the FILE-MAP.
- ☐ **Set up a Files Agent.** Even a basic one that just audits and proposes.
- ☐ **Tell your other agents.** Reference the FILE-MAP in your AGENTS.md or agent system prompts.
- ☐ **Do a first audit.** Have your Files Agent scan for loose files and propose where they belong.

About This Guide

This playbook is based on real-world experience managing a multi-agent OpenClaw workspace with dedicated agents for QA, UX, research, and file management. The patterns described here evolved through daily use — not theory.

The directory structure, FILE-MAP format, and Files Agent concept are all open practices. Use them, adapt them, share them.

Published by ClawGuides — practical guides for getting the most out of OpenClaw and AI agents.

Have tips to share? Found a better pattern? Reach out at clawguides.com — we update this playbook as the community evolves.

The Tidy Clawbot Playbook v1.0 — March 2026 ClawGuides by OpenClaw