Audion Hub Manager — Why It Works This Way

Русский · About · User Guide

Contents

Three decisions that shaped the program. Each was made not for elegance but after a case where the alternative was worse.

1. Three Layers Instead of One Big Folder

The decision

Do not put everything into one giant documentation folder. A project lives in three layers, which may sit under a common root but stay separate in meaning.

```
Manager    an independent portable application
Hub Data   the technical filtered copy, backed by Git
Docs       a neutral folder of notes and descriptions
```

Why separate

The manager is a tool. It can be versioned and mirrored like any other project, but it must not depend on the hub it manages.

Hub Data is technical. Inside are filtered project projections: readmes, specifications, documentation, configuration, source code, launchers. This layer is for the editor, diffs, commits, and code reuse.

Docs are neutral. The docs layer should not scan thousands of source files and scripts. It holds Markdown, indexes, project overviews, daily notes, and links.

The difference is in what opens them. Docs can be synced by anything — an editor, a file manager, Obsidian, LogSeq, a cloud drive, or nothing. Hub Data is Git-backed and does not need to reach a phone.

What follows from it

The docs layer is not hashed separately during verification: it is a consuming layer, and the canonical technical copy is already in Hub Data. Verification runs between source and mirror only.

2. A Mirror That Cannot Damage the Source

The decision

The rules below were carried over from Disk Auditor — they were collected from real failures, and there is no need to learn them twice.

A single source of truth

The full project is the only source of truth. The mirror is derived: it may delete and rebuild its own files, but it **never** modifies the source.

The source may itself be an ordinary Git working tree — the mirror does not care, it only reads what the profile allows. The mirror may be a working tree too, and then its `.git/**` is protected from scanning, deletion, and maintenance.

Separately from the mirror there is the editor: it reads selected files and writes only when you explicitly press save. Automated test runs and window smoke checks never touch the source.

A real write requires explicit intent

A profile may declare preview as the default, and the command line must not ignore that:

```
--mirror-apply PROJECT             preview, if the profile says so
--mirror-apply PROJECT --apply     a real write
--mirror-apply PROJECT --dry-run   forced preview
```

A filter cannot silently become a full copy

A profile must declare that filters are required and how many masks are the minimum. With no masks, planning fails rather than assembling a full duplicate.

The same holds for a future full-mirror mode: if mirroring is on and the mask lists are empty, planning must fail until a profile explicitly allows otherwise through a separate flag. No shipped profile sets that flag.

Writing happens in phases, deletion last

```
1  create the allowed directories
2  copy and update files
3  set modification times
4  if there are no errors or conflicts — delete what only exists in the mirror
5  remove directories left empty
6  place markers in preserved empty folders
```

Deletion does not begin until copying has succeeded. If any earlier phase fails, deletion is skipped and the report says so outright.

The setting that once permitted deletion after errors no longer applies: that is exactly the case where a mirror drops files existing nowhere else.

Three ways to compare

mode	how it compares	when
quick	size and time	rough check
safe	size and time, hashing only when size matches and time differs	ordinary work
strict	hashing for all same-size files, even when times match	checkpoints

The safe mode is fast but deliberately misses "same size, same time, different content". Checkpoints need strict.

Hashes always carry their algorithm: `blake3:…` or `sha256:…`. SHA-256 is a fallback for stripped-down environments, not an equivalent.

Conflicts are not success

Any real apply with errors or conflicts returns a non-zero exit code. A future two-way sync must behave the same.

Details that once cost dearly

Report names carry microseconds — otherwise fast automation overwrites its own reports.

Mirror scope is declared explicitly. "Filtered" means the target is brought into line with the source only for the selected masks; files outside that scope are left alone by scan and plan.

The plan is re-checked before writing. A stale or edited plan could delete along a path overlapping the source — now the profile and the non-overlap rule are verified again.

Tests that must stay green

- a profile requires masks;
- strict mode catches same size, same time, different content;
- safe mode keeps its fast path;
- deletion is skipped when copying fails;
- empty service directories survive mirroring.

Not here

Two-way sync with conflict handling, manifest self-exclusion, release-clean policy. Those belong to Disk Auditor or to future modules; if they are brought here, these rules must come with them.

3. The Program Holds No Passwords

The decision

Hub Manager does not become a token and password vault. The program is portable; Git credentials do not live inside a portable project.

Sign-in is delegated to the ordinary tools:

- SSH keys and ssh-agent;
- the credential manager for HTTPS;
- the GitHub and GitLab command-line tools;
- VS Code and GitKraken for manual sign-in, conflict resolution, and visual work.

The program runs real `git` commands and shows their output in full. If you are already signed in through any of the above, push and pull work while the program knows no secret at all.

What that means in practice

A remote is nothing but a named address:

```
git remote add github git@github.com:audion/Audion_Hub.git
git remote add gitlab git@gitlab.com:audion/Audion_Hub.git
git remote add local_nas file:///Z:/git-mirrors/Audion_Hub.git
```

The sign-in check recognises the address type and tells you how you will be identified — but stays a support panel, not a credential editor.

The exception that isn't one

For self-hosted Forgejo and Gitea the program accepts a personal access token: it verifies the token against the server and hands it to your credential helper. The token never reaches project files, and from then on `git push` picks it up itself.

A typed token works immediately; a separate "remember" button keeps it across restarts — which is exactly what the name promises. Forgetting clears both places: the field and the stored copy.

Known server addresses sit in configuration in the open — with no tokens.

Technical Appendix: Profile Keys

For whoever edits mirroring profiles.

key	effect
`dry_run_default: true`	preview by default; the command line may not override it
`require_include_filter: true`	planning fails when no masks are set
`min_include_globs: 1`	how many masks are the minimum
`allow_unfiltered_full: true`	the only way to permit a full copy; **no shipped profile sets it**
`mirror_scope: "filtered"`	the target matches the source only for selected masks
`mirror_scope: "full"`	not for project mirrors without a separate review

key	effect
`delete_after_successful_copy: false`	still in config, but no longer disables the protection
`delete_phase_skipped: true`	appears in the report when deletion was skipped after a failure

Compare modes: `quick`, `safe`, `strict`. The code-level default is `metadata_then_blake3`, an alias for strict hashing of same-size files, with the older configuration names preserved. Shipped profiles should keep an explicit `strict_blake3` so checkpoint mirrors catch drift at matching size and time.

Report and temporary file names: `YYYYMMDD_HHMMSS_microseconds`.

Any apply with errors or conflicts returns `exit_code: 1`.

Tests that must stay green

```
test_profile_requires_include_masks
test_strict_mode_detects_same_size_same_mtime_different_content
test_safe_mode_keeps_disk_auditor_fast_path_same_size_same_mtime
test_delete_phase_is_skipped_when_copy_phase_fails
test_projection_mirror_preserves_empty_service_dirs
test_projection_preserves_empty_dirs_with_gitkeep
```

Empty-folder markers

After mirroring, maintenance runs over the projection root: a marker is created in every empty directory of the mirrored structure, removed from folders that have gained real files or subdirectories, and never placed in hidden technical directories. The marker counts as generated mirror metadata, not project content.