

GIT_WIKI_HUB_MANAGER EN - Audion Hub Manager Map

Contents

- [1. Core Model](#)
- [2. Local Database And Main Configs](#)
- [3. Project Registry](#)
- [4. MIRROR Profiles](#)
- [5. Main Window Layout](#)
- [6. Header](#)
- [7. Left Control Panel](#)
- [8. Structure Tree](#)
- [9. Inspector: Quick](#)
- [10. Inspector: Basket](#)
- [11. Inspector: Branch](#)
- [12. Inspector: Remote](#)
- [13. Inspector: Editor](#)
- [14. Inspector: Reader](#)
- [15. Inspector: Diff](#)
- [16. Inspector: History](#)
- [17. Inspector: Details](#)
- [18. Inspector: Storage And Safety](#)
- [19. Terminal / Command Dock](#)
- [20. GitHub/GitLab Remote Workflow](#)
- [21. MIRROR And Config Maintenance](#)
- [22. Detailed Git-Work Function Map](#)
- [23. Weak Spots And Missing Frequent Commands](#)

- [24. Suggested Next Backlog](#)

Status: 2026-06-03.

This document is the working wiki map for Audion Hub Manager. It describes the application windows, logical groups, commands, input fields, config/local database maintenance, GitHub/GitLab/Git remote work, authentication policy and current workflow gaps.

1. Core Model

Audion Hub Manager connects three layers:

```
Full Project / Source -> Hub Data / Mirror -> Git remotes
                        -> Docs view
```

- **Source** is the live full project and source of truth.
- **Hub Data / Mirror** is a filtered technical projection. It can have its own **.git** and can be rebuilt from Source.
- **Docs** is an optional Markdown/text layer for reading and external docs tools.

The app is not a password vault. It runs real **git** and relies on external authentication: SSH agent, Git Credential Manager, GitHub CLI, GitLab CLI, VS Code, GitKraken, or any already configured Git setup.

2. Local Database And Main Configs

Hub Manager currently stores state in JSON/YAML files, not in a large SQL database:

- **config/projects.json**: project registry and active project id.
- **config/projection_profiles.json**: MIRROR rules and commit allowlist.
- **config/remotes.json**: saved Git remote names and URLs.
- **config/auth_profiles.json**: authentication strategy metadata.
- **config/storage_layout.json**: Manager, Hub Data, Docs and Source roots.
- **config/apps.json**: shared external app defaults.
- **config/apps.local.json**: machine-local overrides, especially **Code.exe**.
- **config/command_cache.json**: manual command history and pinned commands.
- **config/remote_field_cache.json**: recent values for the remote form.
- **config/gui_settings.yaml**: language, theme and UI settings.

Reports and diagnostics are written to `logs/<project_id>/` as timestamped JSON files. Workspace helpers are written to `workspace/`.

3. Project Registry

One `projects.json` entry describes one real project:

- `id`: stable internal key.
- `title`: visible title in the Project dropdown.
- `source_path`: live project root. Relative paths are resolved from the Hub Manager root.
- `projection_path`: Hub Data / Mirror folder for the project.
- `docs_path`: optional Docs folder.
- `profile`: projection profile id from `projection_profiles.json`.
- `default_branch`: branch used by push/pull/branch defaults.
- `docs_app_name`, `docs_file`: legacy/docs integration fields.
- `vscode_workspace`: optional workspace opened instead of `source_path`.
- `notes`: free-form note.

Folder picker buttons in `Structure` can write Project/GIT COPY/Docs paths back to `projects.json`. Paths inside the manager tree are saved as relative paths where possible; external paths stay absolute.

4. MIRROR Profiles

`projection_profiles.json` controls both MIRROR and Hub Manager stage/commit path checks:

- `compare_mode`: `quick`, `safe`, `metadata_then_blake3` or strict BLAKE3.
- `mirror`: whether projection-only files may be deleted.
- `mirror_scope`: usually `filtered`.
- `preserve_empty_dirs`: whether empty allowed directories receive `.gitkeep`.
- `marker_file`: marker filename.
- `max_file_bytes`: normal allowed file size cap.
- `include_globs`: main allowlist.
- `exclude_globs`: exclusions for secrets, binaries and generated output.
- `small_include_globs`, `small_include_max_file_bytes`: small license/notice exceptions.

- `hide_dirs`, `exclude_dir_contents`, `forbidden_dirs`: skipped directory classes.
- `protected_target_globs`: protected target paths, including `.git/**`.
- `require_include_filter`, `min_include_globs`: protection against copy-all profiles.
- `delete_after_successful_copy`: stale target deletion is skipped if copy/touch had errors or conflicts.

5. Main Window Layout

The main window has four surfaces:

Left Control panel | Structure tree | Inspector tabs
Bottom Terminal / command dock

Splitters resize the left panel, tree, Inspector and terminal height. Inspector tabs:

Quick | Branch | Editor | Diff | Storage
Remote | Basket | Reader | History | Details

Each tab header starts with a Material icon. Tabs and command buttons have tooltips with a 1200 ms show delay and 80 ms hide/fade.

Safety lives inside the Storage area. The selected tree path is shown in the upper-right part of the Inspector.

6. Header

The header shows:

- application title;
- Git status counters: staged, modified, untracked, conflict, changed;
- status counter mode toggle: words/icons/letters;
- theme selector;
- language toggle.

Counters refresh after Git status operations.

7. Left Control Panel

Project

The **Project** dropdown selects the active record from **projects.json**. It switches this whole set together:

```
source_path
projection_path
docs_path
profile
default_branch
```

Open

- **Open Project**: open Source in the OS file manager.
- **Mirror**: open projection root.
- **Docs Folder**: open Docs root.
- **Open in VS Code**: open **vscode_workspace** when configured, otherwise Source.
- **Terminal**: open an external terminal in Source and run **git status --short**.
- **Git**: open an external terminal in the current Git root and run **git status --short**.

MIRROR

Options:

- **Dry-run**: **Apply MIRROR** reports actions without writing files.
- **Exact mirror**: enables mirror behavior for the active profile.
- **.gitkeep dirs**: keeps empty allowed directories via marker files.
- **BLAKE3 compare**: enables strict BLAKE3 comparison; otherwise quick compare is used.

Commands:

- **Preview MIRROR**: build Source -> Mirror plan and write a report.
- **Apply MIRROR**: apply the current plan, respecting **Dry-run**.
- **Refresh**: refresh Git status and tree.

SOURCE ACTIONS

Operations over the project registry or Source collection:

- **Rebuild dropdown**: choose a folder, scan nested project roots and add missing records to **projects.json**.
- **Clone Source**: queue **git clone <url> <dest>** in command area.
- **Batch Preview MIRROR**: build a MIRROR plan for every project.
- **Batch Safety Scan**: scan every Source for secrets and heavy files.
- **Batch Verify Mirror**: read-only Source/Mirror digest verification for all projects.
- **Batch Git status**: collect Git status for all Sources and update the **SOURCE** badge.
- **Combined workspace**: create one **.code-workspace** for all projects.

PROJECT ACTIONS

Operations on the current selected tree item:

- **Refresh tree**: rebuild tree.
- **Load selected to Editor**: open selected **.md**, **.markdown**, **.txt**, **.rst**.
- **Open in VS Code**: open selected file/folder.
- **Copy relative path**: copy path relative to the current root.
- **Copy full path**: copy absolute path.

Support

- **Auth Doctor**: Git/auth/remote diagnostics.
- **BLAKE3**: hash backend probe.
- **Verify Mirror**: read-only Source/Mirror verification.
- **Storage**: root and role separation check.
- **Safety Scan**: scan current root for secrets/heavy files.
- **Clean projects.json**: remove missing Source entries and duplicates. It does not delete folders.

8. Structure Tree

Layer buttons:

- **PROJECT**: active **source_path**.
- **MIRROR**: active **projection_path**.

- `DOCS` : active `docs_path`.

Folder icons choose a new layer location. Clear locations removes saved overrides from project config.

Filters:

- `View` : `Full Tree`, `Changed Only`, `Staged`, `Untracked`, `Conflicts`.
- `Search` : filter by label/path.
- `Hide clean` : hide clean nodes when Git status is available.
- `Show hidden` : show hidden entries.
- `Top level scan` : keep lazy top-level tree behavior during search.

Behavior:

- Single click selects a path and updates Details/Diff preview.
- Double click opens supported text files in Editor, or opens files in VS Code.
- Status dots and summary letters come from `git status --porcelain=v1 -b`.

9. Inspector: Quick

`Quick` contains frequent local and selected-path commands. Some buttons run Python handlers; some queue exact command text into the manual command area.

GIT LOCAL

- `git init` : queued `git init`.
- `git status` : refresh status.
- `git root` : queued `git rev-parse --show-toplevel`.
- `git log --oneline` : queued `git log --oneline --decorate -20`.
- `git reflog` : queued `git reflog --date=local -20`.

GIT DIFF

Fields use the selected path when available:

- `git diff -- <path>`.
- `git diff --cached -- <path>`.
- `git blame -- <path>`.
- `git show --stat <commit>`.

GIT SELECTED PATH

- `git add -- <path>` .
- `git restore --staged -- <path>` .
- `git restore -- <path>` .
- `add active project` : stage `.` in the active Mirror root after profile check.
- `basket + selected` : add selected path to Basket.
- `Copy relative path` : copy selected path.

GIT BACKUP / MAINTENANCE

- `git bundle create` : queued bundle into `backup/checkpoint.bundle` .
- `git clean preview` : queued `git clean -nd` .
- `git gc` : queued `git gc` .
- `git fsck` : queued `git fsck --full` .
- `git config list` : queued `git config --list --show-origin` .

GIT ADVANCED - DANGER

- `git reset --hard <commit>` : visible template; blocked by danger filter when run through the command runner.
- `git reset --soft HEAD~1` : soft rewind template.

CI/CD status

Shown only when CLI tools are found:

- `gh run list` .
- `gh run watch` .
- `glab ci status` .

Command Cache

- Pinned/history selects copy exact commands into manual command area.
- Pin/unpin/delete/clear maintain `config/command_cache.json` .
- `Run` executes exactly the command area text.

Danger filter blocks commands containing:


```
reset --hard
clean -fd
push --force
rm -rf
rmdir /s
del /s
```

10. Inspector: Basket

Basket prepares readable commits and checkpoint tags.

Fields:

- **type**: docs, code, fix, ui, test, chore, audit.
- **Scope**: optional conventional commit scope.
- **Subject**: commit subject.
- **version_series**: defaults to **projection**.
- **version_value**: semver, for example **v0.1.0**.
- **bump**: patch, minor, major.
- **tag_head_field**: generated tag, read-only.
- **Commit message**: generated or manual message.

Message rule:

```
<type>(<scope>): <series> <version> - <subject>
```

If **Subject** is empty, the manually typed **Commit message** is used.

Commands:

- **next version**: scan tags **<series>-v*** and bump the latest semver.
- **git tag HEAD**: create annotated tag on HEAD with generated message.
- **Update message**: rebuild commit message from fields.
- **basket clear**: clear in-memory basket.
- **add active project**: stage active Mirror root.
- **git add -- basket**: stage basket paths.
- **git restore --staged -- basket**: unstage basket paths.

- `git commit --only -- basket` : commit only basket paths.
- `git commit -m` : commit currently staged paths.

Basket is bound to the current Git root. If root changes, it is cleared to avoid cross-repository commits. Stage/commit checks the active projection profile and blocks paths outside the Hub allowlist.

11. Inspector: Branch

Branch contains branch switching, comparison, integration and stash.

branch status

- `git status` : refresh status.
- `git branch -vv` : queued tracking view.
- `git log --graph` : graph log across all refs.
- `git fetch --all --prune` : fetch/prune remote tracking refs.

git branch / tag

- `git switch <branch>` .
- `git switch -c <new branch>` .
- `git tag version` : queued annotated tag command from Basket version fields.
- `git tag -n` : tag list by creator date.

merge

- `git log --left-right --graph --cherry-pick --oneline HEAD...<branch>` .
- `git revert <commit>` .
- `git merge --no-ff <branch>` .
- `git cherry-pick <commit>` .

git stash

- `git stash push -u -m <message>` .
- `git stash pop` .
- `git stash list` .

branch danger

- `git rebase -i <revision range>`.
- `git merge --abort`.
- `git cherry-pick --abort`.

These are visible templates: the operator sees and can edit the exact command.

12. Inspector: Remote

Remote owns remotes, push/pull/fetch and auth setup.

Remote form fields:

- **Platform**: `GitHub`, `GitLab`, `Codeberg`.
- **Remote name**: Git remote name. If empty on save, it becomes `hidden_<platform>`.
- **Login / group**: owner/group path.
- **Repository**: repo name. A trailing `.git` is removed for URL building.
- **Remote URL**: explicit URL. If filled, it wins over generated URL.
- Recent selects for names, owners, repos and URLs.
- Use buttons apply recent values explicitly, with no automatic overlay.

Generated SSH URL:

```
git@github.com:<owner>/<repo>.git
git@gitlab.com:<owner>/<repo>.git
git@codeberg.org:<owner>/<repo>.git
```

Remote commands:

- `git push origin`: queued `git push --follow-tags origin <default_branch>`.
- `git pull --ff-only`: queued `git pull --ff-only origin <default_branch>`.
- `git fetch --all --prune`: run fetch/prune and refresh status.
- `git remote -v`: show repo remotes.
- `git push all remotes`: Python enumerates `git remote` and pushes `--follow-tags` to each remote sequentially.
- `git apply remotes.json`: add or update enabled remotes from `config/remotes.json`.
- `git origin push URLs`: configure `origin` with multiple push URLs from enabled remotes.

- `build URL` : write generated SSH URL into `Remote URL` .
- `save remote` : write/update an enabled record in `config/remotes.json` and update `remote_field_cache.json` .

Auth tools:

- `Check Auth` : Auth Doctor.
- `git config user` : queued `git config --show-origin --get-regexp user\.`
- `ssh -T GitHub` : non-interactive SSH probe to `git@github.com` .
- `ssh -T GitLab` : non-interactive SSH probe to `git@gitlab.com` .
- `gh auth login` : external terminal.
- `glab auth login` : external terminal.
- `Windows Credentials` : Windows Credential Manager.
- `GitKraken folder` : open current Git root folder.
- `VS Code` : open active project in VS Code.

Auth Doctor checks Git, global Git identity, `gh` , `glab` , SSH hosts, repo remotes, URL types/providers and optional `git ls-remote` for enabled remotes.

13. Inspector: Editor

Supported extensions:

```
.md
.markdown
.txt
.rst
```

Toolbar:

- Load selected.
- Save.
- Paste from Windows clipboard.
- Copy editor text.
- Clear editor text.
- Open current file in VS Code.
- Expand/collapse panel.

Save writes UTF-8 only by explicit user action and refreshes the tree.

14. Inspector: Reader

Reader is the shared summary area. It can show MIRROR summaries, Git status, Auth Doctor summary, Storage summary and batch command summaries.

15. Inspector: Diff

Commands:

- **Unstaged**: `git diff -- <selected path>`.
- **Staged**: `git diff --cached -- <selected path>`.
- **HEAD**: `git diff HEAD -- <selected path>`.
- **Copy patch**: copy current diff text.

Diff is rendered as a RedLine view: line numbers, hunks, additions, removals and metadata lines.

16. Inspector: History

Commands:

- **Selected path**: `git log --date=short --pretty=... -40 -- <path>`.
- **Repository**: repo log, last 50 commits.
- **Graph**: `git log --graph --oneline --decorate --all -50`.
- **Tags**: `git tag -n --sort=-creatordate`.
- **Copy history**: copy current history text.

17. Inspector: Details

Details shows:

- project;
- tree scope;
- relative/full path;
- Git status;
- file/dir/missing type;

- size;
- modified time;
- Git blob metadata when available.

Commands:

- Refresh.
- Copy JSON.
- Open in VS Code.

18. Inspector: Storage And Safety

Storage commands:

- **Check layout**: check Manager, Hub Data, Docs and full projects roots.
- **Scan projects**: scan selected parent folder and import project entries.
- **Generate workspace**: create **.code-workspace** for current project layers.
- **Copy JSON**: copy storage payload.
- **Open workspace**: open generated workspace.

External tools:

- **VS Code executable**: machine-local **Code.exe** path.
- **Pick**: file picker.
- **Save**: write to **config/apps.local.json**.
- **Test**: check/launch resolved VS Code command.

Safety commands:

- **Scan current root**: find secret-like files, embedded tokens, private keys, heavy extensions and large files.
- **Copy JSON**: copy safety payload.

Safety skips generated/runtime dirs: **.git**, **.venv**, **runtime**, **wheelhouse**, **node_modules**, **logs**, **output**, **backup**, **release**, **report**, **temp**, **tmp** and similar directories.

19. Terminal / Command Dock

The bottom dock contains:

- terminal log;
- manual command input;
- `Run`;
- `Clear`;
- terminal toolbar: clear/expand.

Manual commands run in the current Git root when possible, otherwise in the manager root. Output is streamed to the terminal dock. A command is added to history unless it is blocked by the danger filter.

20. GitHub/GitLab Remote Workflow

Recommended SSH flow:

1. Configure SSH keys outside Hub Manager.
2. Check `ssh -T GitHub` and `ssh -T GitLab`.
3. Fill `Platform`, `Remote name`, `Login / group`, `Repository`.
4. Press `build URL`.
5. Press `save remote`.
6. Press `git apply remotes.json`.
7. Check `git remote -v`.
8. Use `git fetch --all --prune` for safe remote tracking updates.
9. After commit/tag, use `git push origin` or `git push all remotes`.

For HTTPS, use Git Credential Manager, GitHub CLI or GitLab CLI. Remote URLs must stay clean:

Good: `https://gitlab.com/user/repo.git`

Bad: `https://username:TOKEN@gitlab.com/user/repo.git`

`origin push URLs` is useful when fetch/pull should use one canonical remote, while push should publish to several mirrors.

21. MIRROR And Config Maintenance

Daily maintenance:

- `Preview MIRROR` before real apply.
- `Apply MIRROR` with `Dry-run` when unsure.
- `Verify Mirror` after important projection changes.
- `Safety Scan` before public push or release.
- `Batch Git status` for registry health.
- `Clean projects.json` after moving/removing projects.
- `Storage -> Check layout` after moving Manager, Hub Data or Docs roots.
- `BLAKE3` after rebuilding portable runtime.

Registry maintenance rules:

- `Clean projects.json` edits config only.
- Project scanner imports missing records and skips duplicate sources.
- Scanner ignores runtime/build/cache folders and nested Hub/Docs roots.
- Storage check reports configuration problems but does not invent new paths.

22. Detailed Git-Work Function Map

This section documents the Git-work layer: which commands run immediately, which are only queued, which code functions handle them and which guards are active.

22.1. Where Git commands run

The current Git root is defined by `current_git_root()` and equals the current tree root:

```
PROJECT -> source_path
GIT COPY -> projection_path
DOCS -> docs_path
```

Before running Quick/Branch/Remote commands, check the active `Structure` layer. `add active project` is the exception: it always uses `mirror_root()` and stages the active Hub projection.

22.2. Direct handlers

Direct handlers call Python wrappers around Git immediately:

- `refresh_git()` -> `git status --porcelain=v1 -b`.
- `stage_selected()` -> `git add -- <selected>`.
- `unstage_selected()` -> `git restore --staged -- <selected>`.
- `stage_active_project()` -> `git add -- .` in Mirror root after allowlist check.
- `stage_basket()` -> `git add -- <basket paths>`.
- `unstage_basket()` -> `git restore --staged -- <basket paths>`.
- `commit_basket()` -> `git commit --only -m <message> -- <basket paths>`.
- `commit_staged()` -> `git commit -m <message>`.
- `tag_head_version()` -> `git tag -a <tag> -m <message>`.
- `show_remotes()` -> `git remote -v`.
- `fetch_all_remotes()` -> `git fetch --all --prune`.
- `apply_configured_remotes()` -> add/set-url enabled remotes from `config/remotes.json`.
- `configure_origin_push_urls()` -> ensure `origin` and add push URLs from enabled remotes.
- `push_all_remotes()` -> `git remote`, then sequential `git push --follow-tags <remote> <default_branch>`.
- `show_selected_diff()` -> `git diff`, `git diff --cached` or `git diff HEAD`.
- `show_history()` -> selected path log, repo log, graph log or tags.

22.3. Queued templates

Queued templates call `queue_git_command()`. They fill both the Inspector command area and bottom terminal command input, add the command to history, and wait for manual `Run`.

Queued commands include:

- `git init`.
- `git rev-parse --show-toplevel`.
- `git log --oneline --decorate -20`.
- `git reflog --date=local -20`.
- `git diff -- <path>`.
- `git diff --cached -- <path>`.

- `git blame -- <path>.`
- `git show --stat <commit>.`
- `git add -- <path>.`
- `git restore --staged -- <path>.`
- `git restore -- <path>.`
- `git bundle create "<backup/checkpoint.bundle>" <default_branch> --tags.`
- `git clean -nd.`
- `git gc.`
- `git fsck --full.`
- `git config --list --show-origin.`
- `git reset --hard <commit>.`
- `git reset --soft HEAD~1.`
- `gh run list --limit 15.`
- `gh run watch.`
- `glab ci status.`
- `git switch <branch>.`
- `git switch -c <new branch>.`
- `git tag -a "<tag>" -m "<message>".`
- `git tag -n --sort=creatordate.`
- `git log --left-right --graph --cherry-pick --oneline HEAD...<branch>.`
- `git revert <commit>.`
- `git merge --no-ff <branch>.`
- `git cherry-pick <commit>.`
- `git stash push -u -m "<message>".`
- `git stash pop.`
- `git stash list.`
- `git rebase -i <range>.`
- `git merge --abort.`
- `git cherry-pick --abort.`
- `git push --follow-tags origin <default_branch>.`
- `git pull --ff-only origin <default_branch>.`
- `git config --show-origin --get-regexp user\.`

22.4. Command runner and danger filter

`run_shell_command()` runs manual commands with `shell=True`, streams stdout and stderr into the terminal dock, writes the exit code and updates status.

Before execution it calls `is_dangerous_command()`. Blocked tokens:

```
reset --hard
clean -fd
push --force
rm -rf
rmdir /s
del /s
```

Important nuance: dangerous commands may be queued as visible templates so the operator can inspect/edit them, but unchanged execution through `Run` is blocked by the danger filter.

22.5. Commit profile guard

Hub Manager does not blindly commit everything. Before selected/basket/Mirror stage/commit flows it checks paths through the active projection profile:

- `commit_profile_violations()`.
- `commit_paths_blocked_by_hub_profile()`.

The check expands directories, ignores marker files like `.gitkeep`, applies `include_file()` from the active profile, and blocks files outside the Hub allowlist. This protects Source and Mirror Git workflows from runtime payload, logs, binaries, PDFs, secrets and local config.

22.6. Basket state machine

Basket is in-memory state:

- `state.commit_basket`: set of relative paths.
- `state.commit_basket_root`: root lock.

When Git root changes, `ensure_commit_basket_root()` clears the basket and locks it to the new root. This prevents cross-repository commits.

Basket message generation:

```
type + optional scope + version_series + version_value + subject
```

Tag generation:

```
<slugified version_series>--<normalized semver>
```

`next version` scans existing tags with the same series prefix and bumps `patch`, `minor` or `major`.

22.7. Remote config functions

Remote form flow:

- `remote_form_field_values()` reads platform, owner, repo and remote name.
- `remote_url_from_identity_fields()` builds SSH URL for GitHub/GitLab/Codeberg.
- `remote_url_from_form()` prefers explicit `Remote URL` over generated URL.
- `build_remote_url_into_field()` fills the URL field.
- `save_remote_config_from_fields()` validates remote name, writes `config/remotes.json`, and updates recent cache.

Remote application flow:

- `git_apply_remotes_from_config()` reads enabled remotes and runs `git remote add` or `git remote set-url`.
- `git_configure_origin_push_urls_from_config()` ensures `origin` exists and adds push URLs through `git remote set-url --add --push origin <url>`.
- `git_push_all_remotes()` enumerates remote names with `git remote` and pushes branch/tags sequentially.

Remote names are limited to `[A-Za-z0-9._-]+`. Secrets in remote URLs are not stored by policy.

22.8. Auth and external tools

Auth Doctor uses `system_core/core/auth_doctor.py`:

- `git --version`.
- `git config --global user.name`.
- `git config --global user.email`.
- `gh auth status` when `gh` exists.
- `glab auth status` when `glab` exists.
- SSH probes for configured hosts.

- `git remote -v` for active repo.
- URL type/provider detection.
- optional `git ls-remote --heads <url>` for enabled remotes without embedded credentials.

External setup buttons:

- `gh auth login` and `glab auth login` open a visible terminal.
- `Windows Credentials` opens Windows Credential Manager.
- `GitKraken folder` opens the current Git root.
- `VS Code` opens the active project.

Background probes set non-interactive behavior to avoid freezing the GUI on password prompts.

22.9. Diff and History functions

Diff:

- `show_selected_diff("unstaged")` : unstaged patch.
- `show_selected_diff("staged")` : staged patch.
- `show_selected_diff("head")` : diff against HEAD.
- `copy_diff_patch()` : clipboard copy.

History:

- `show_history("selected")` : selected path log.
- `show_history("repo")` : repository log.
- `show_history("graph")` : graph across refs.
- `show_history("tags")` : sorted annotated tag list.
- `copy_history_text()` : clipboard copy.

Selecting a tree node also triggers Details refresh and lightweight diff preview for changed files.

22.10. Git-work reports and logs

Git command output is streamed to the terminal dock. Structured diagnostics are written as JSON reports:

- projection plan;
- projection apply;
- mirror verify;

- BLAKE3 probe;
- project scan;
- projects config clean;
- batch MIRROR/status/safety/verify payloads.

Reports use `write_report(kind, payload, project_id=...)` and live under `logs/<project_id>/`.

23. Weak Spots And Missing Frequent Commands

The app already covers most everyday Git/MIRROR workflows, but these gaps remain.

Remote status and sync

- No compact ahead/behind table per remote.
- No `git fetch <remote> <branch>` button.
- No selected remote push/pull; actions mostly use `origin` or all remotes.
- No `git remote show <name>` inspector.
- No remote-grouped fetch/prune report.
- No UI to disable/remove a saved remote from `remotes.json`.

Branch workflow

- No branch dropdown from `git branch --all`.
- No current branch badge inside Branch pane.
- No `git switch -` for previous branch.
- No `git branch -d/-D <branch>` template.
- No upstream setup: `git push -u origin <branch>` or `git branch --set-upstream-to`.
- No visual merge/rebase state indicator beyond raw output.

Staging and commit

- Diff has no explicit selected-path `stage` / `unstage` buttons, although handlers already exist in code.
- Basket cannot remove one path, only clear all.
- No amend flow: `git commit --amend`.
- No commit dry-run/status preview for the exact basket.

- Commit profile violations go to terminal, not a compact Basket list.

History and recovery

- Reflog exists in Quick, but there is no recovery helper: `git checkout <sha> -- <path>`.
- No viewer for `git show <commit>:<path>`.
- No commit hash copy from History.
- No tag delete templates, local or remote.

Auth

- Auth Doctor knows URL types/providers, but UI does not show badges next to each configured remote.
- No guided SSH key creation/checklist.
- No separate `gh auth status` / `glab auth status` buttons; they are only inside Auth Doctor.
- No HTTPS credential cleanup checklist beyond opening Windows Credentials.

Safety and release

- Safety Scan is not yet a gate for push buttons.
- No release checklist command group.
- No `git archive`.
- No signed tag/commit option.

Storage and Docs

- Docs layer opens, but there is no dedicated docs-only projection action.
- `Sync Docs Back` is intentionally absent, but UI should label this workflow as absent/unsafe.
- Storage check shows paths, but offers few repair actions besides picker and scanner.

UI and ergonomics

- Many commands require manual `Run`: safe, but slow for frequent benign operations.
- Risk commands are blocked by token matching, not typed confirmation dialogs.
- Command cache is global, not per project.
- Remote recent values are global, not per provider/project.
- Small-screen layout should continue to be tuned by screenshots after every layout change.

24. Suggested Next Backlog

Small low-risk improvements:

1. Branch dropdown from `git branch --all`.
2. Templates for `git switch -` and `git push -u origin <branch>`.
3. Selected-path stage/unstage buttons in Diff.
4. `basket remove selected` or clickable Basket rows.
5. `remote show <name>` and `fetch selected remote`.
6. Auth/Remote badge table: name, provider, URL type, enabled, configured in repo, ls-remote ok.
7. Visible stale Safety warning before push.
8. `git commit --amend` as queued command, not direct action.
9. `git show <commit>:<path>` viewer in History/Diff.
10. Docs-only projection action if Docs becomes an active workflow.