

GIT_WIKI_HUB_MANAGER - карта Audion Hub Manager

Содержание

- [1. Главная модель](#)
- [2. Локальная база и основные конфиги](#)
- [3. Реестр проектов](#)
- [4. Профили MIRROR](#)
- [5. Общая раскладка окна](#)
- [6. Header](#)
- [7. Левая панель Control](#)
- [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 и Safety](#)
- [19. Terminal / command dock](#)
- [20. GitHub/GitLab remote workflow](#)
- [21. MIRROR и обслуживание конфигов](#)
- [22. Подробная карта Git-work функций](#)
- [23. Провисающие места и пропущенные частые команды](#)

- [24. Рекомендуемый следующий backlog](#)

Статус: 2026-06-03.

Этот документ - рабочая wiki-карта Audion Hub Manager. Здесь описаны окна, логические группы, команды, поля ввода, обслуживание конфигов/локальной базы, работа с GitHub/GitLab/Git remotes, политика аутентификации и текущие белые пятна программы.

1. Главная модель

Audion Hub Manager связывает три слоя:

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

- **Source** - живой полный проект и источник истины.
- **Hub Data / Mirror** - фильтрованная техническая проекция проекта. Она может иметь собственный **.git** и может быть пересобрана из Source.
- **Docs** - необязательный Markdown/text слой для чтения и внешних docs-tools.

Приложение не является хранилищем паролей. Оно запускает настоящий **git** и полагается на внешнюю аутентификацию: SSH agent, Git Credential Manager, GitHub CLI, GitLab CLI, VS Code, GitKraken или уже настроенный Git.

2. Локальная база и основные конфиги

На текущем этапе у Hub Manager нет большой SQL-базы. Его состояние хранится в JSON/YAML-файлах:

- **config/projects.json** - реестр проектов и активный проект.
- **config/projection_profiles.json** - правила MIRROR и commit allowlist.
- **config/remotes.json** - сохраненные Git remote names и URL.
- **config/auth_profiles.json** - описание стратегии авторизации.
- **config/storage_layout.json** - корни Manager, Hub Data, Docs и Source.
- **config/apps.json** - общие defaults внешних приложений.
- **config/apps.local.json** - machine-local overrides, прежде всего путь к **Code.exe**.
- **config/command_cache.json** - история и pinned manual commands.

- `config/remote_field_cache.json` - recent values для remote form.
- `config/gui_settings.yaml` - язык, тема и UI-настройки.

Отчеты и диагностические payload пишутся в `logs/<project_id>/` как датированные JSON. Workspace helpers пишутся в `workspace/`.

3. Реестр проектов

Один объект `projects.json` описывает один реальный проект:

- `id` - стабильный внутренний ключ.
- `title` - название в dropdown `Project`.
- `source_path` - живой проект. Относительные пути считаются от корня Hub Manager.
- `projection_path` - Hub Data / Mirror папка выбранного проекта.
- `docs_path` - необязательная Docs папка.
- `profile` - id профиля из `projection_profiles.json`.
- `default_branch` - ветка для push/pull/branch defaults.
- `docs_app_name`, `docs_file` - legacy/docs integration поля.
- `vscode_workspace` - workspace, который открывается вместо `source_path`, если поле заполнено.
- `notes` - свободная заметка.

Кнопки выбора папки в `Structure` могут записывать Project/GIT COPY/Docs пути обратно в `projects.json`. Пути внутри дерева менеджера сохраняются относительно, внешние пути остаются абсолютными.

4. Профили MIRROR

`projection_profiles.json` управляет и MIRROR, и проверкой путей перед Hub Manager stage/commit:

- `compare_mode` - `quick`, `safe`, `metadata_then_blake3` или strict BLAKE3.
- `mirror` - разрешено ли удалять projection-only файлы.
- `mirror_scope` - обычно `filtered`.
- `preserve_empty_dirs` - создавать ли `.gitkeep` в пустых разрешенных папках.
- `marker_file` - имя marker-файла.

- `max_file_bytes` - лимит размера обычного разрешенного файла.
- `include_globs` - основной allowlist файлов.
- `exclude_globs` - исключения: секреты, бинарники, generated output.
- `small_include_globs`, `small_include_max_file_bytes` - малые license/notice исключения.
- `hide_dirs`, `exclude_dir_contents`, `forbidden_dirs` - классы пропускаемых папок.
- `protected_target_globs` - защищенные target paths, включая `.git/**`.
- `require_include_filter`, `min_include_globs` - защита от copy-all профиля.
- `delete_after_successful_copy` - удаление stale target файлов пропускается, если фаза copy/touch дала ошибки или конфликты.

5. Общая раскладка окна

Главное окно состоит из четырех поверхностей:

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

Splitter меняет ширину левой панели, дерева, Inspector и высоту терминала. Inspector содержит вкладки:

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

Каждый заголовок вкладки начинается с Material-иконки. Tabs и command buttons имеют tooltips с задержкой появления 1200 ms и hide/fade 80 ms.

Safety находится внутри зоны Storage. Текущий выбранный путь показывается в правом верхнем углу Inspector.

6. Header

Header показывает:

- название приложения;
- счетчики Git status: staged, modified, untracked, conflict, changed;

- переключатель вида счетчиков: words/icons/letters;
- выбор темы;
- переключение языка.

Счетчики обновляются после Git status операций.

7. Левая панель Control

Project

Dropdown **Project** выбирает активную запись из **projects.json**. Это переключает сразу весь комплект:

```
source_path
projection_path
docs_path
profile
default_branch
```

Open

- **Open Project** - открыть Source в файловом менеджере.
- **Mirror** - открыть projection root.
- **Docs Folder** - открыть Docs root.
- **Open in VS Code** - открыть **vscode_workspace**, если он задан, иначе Source.
- **Terminal** - открыть внешний терминал в Source и выполнить **git status --short**.
- **Git** - открыть внешний терминал в текущем Git-root и выполнить **git status --short**.

MIRROR

Опции:

- **Dry-run** - **Apply MIRROR** показывает действия без записи файлов.
- **Exact mirror** - включает mirror-поведение активного профиля.
- **.gitkeep dirs** - сохраняет пустые разрешенные папки marker-файлами.
- **BLAKE3 compare** - включает strict BLAKE3 сравнение; без него режим быстрый.

Команды:

- **Preview MIRROR** - строит план Source -> Mirror и пишет report.
- **Apply MIRROR** - применяет текущий план, учитывая **Dry-run**.
- **Refresh** - обновляет Git status и дерево.

SOURCE ACTIONS

Операции по реестру проектов или общему Source-контейнеру:

- **Rebuild dropdown** - выбрать папку, просканировать вложенные project roots и добавить недостающие записи в **projects.json**.
- **Clone Source** - поставить в command area шаблон **git clone <url> <dest>**.
- **Batch Preview MIRROR** - построить MIRROR plan для всех проектов.
- **Batch Safety Scan** - просканировать Source всех проектов на секреты и тяжелые файлы.
- **Batch Verify Mirror** - read-only digest verification Source/Mirror для всех проектов.
- **Batch Git status** - собрать Git status по всем Source и обновить **SOURCE** badge.
- **Combined workspace** - создать общий **.code-workspace** для всех проектов.

PROJECT ACTIONS

Операции над текущим выбранным объектом дерева:

- **Refresh tree** - перестроить дерево.
- **Load selected to Editor** - открыть выбранный **.md**, **.markdown**, **.txt**, **.rst** во встроенном Editor.
- **Open in VS Code** - открыть выбранный файл/папку.
- **Copy relative path** - скопировать путь относительно текущего root.
- **Copy full path** - скопировать абсолютный путь.

Support

- **Auth Doctor** - диагностика Git/auth/remotes.
- **BLAKE3** - проверка hash backend.
- **Verify Mirror** - read-only сверка Source/Mirror.
- **Storage** - проверка корней и разделения ролей.
- **Safety Scan** - проверка текущего root на секреты/тяжелые файлы.
- **Clean projects.json** - удалить из реестра битые Source и дубликаты. Папки не удаляются.

8. Structure tree

Переключатели слоя:

- **PROJECT** - **source_path** активного проекта.
- **MIRROR** - **projection_path**.
- **DOCS** - **docs_path**.

Иконки папок выбирают новое расположение слоя. Clear locations удаляет сохраненные overrides из project config.

Фильтры:

- **View**: **Full Tree**, **Changed Only**, **Staged**, **Untracked**, **Conflicts**.
- **Search**: фильтр по label/path.
- **Hide clean**: скрывает clean nodes, если есть Git status map.
- **Show hidden**: показывает скрытые entries.
- **Top level scan**: оставляет lazy top-level дерево при поиске.

Поведение дерева:

- Single click выбирает путь и обновляет Details/Diff preview.
- Double click открывает поддерживанный текстовый файл в Editor или файл в VS Code.
- Цветовые dots и summary letters идут из **git status --porcelain=v1 -b**.

9. Inspector: Quick

Quick - частые локальные и selected-path команды. Часть кнопок выполняет Python handler, часть кладет точный текст в 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

Поля по возможности получают selected path:

- `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 `.` в активном Mirror root после profile check.
- `basket + selected` - добавить selected path в Basket.
- `Copy relative path` - скопировать selected path.

GIT BACKUP / MAINTENANCE

- `git bundle create` - queued bundle в `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; при запуске через command runner блокируется danger filter.
- `git reset --soft HEAD~1` - soft rewind template.

CI/CD status

Показывается только если найдены CLI:

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

Command cache

- Pinned/history selects копируют exact command в manual command area.
- Pin/unpin/delete/clear обслуживают `config/command_cache.json`.
- `Run` выполняет ровно текст из command area.

Danger filter блокирует команды, содержащие:

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

10. Inspector: Basket

Basket готовит читаемые commits и checkpoint tags.

Поля:

- `type`: `docs`, `code`, `fix`, `ui`, `test`, `chore`, `audit`.
- `Scope`: optional conventional commit scope.
- `Subject`: тема коммита.
- `version_series`: по умолчанию `projection`.
- `version_value`: semver, например `v0.1.0`.
- `bump`: `patch`, `minor`, `major`.
- `tag_head_field`: generated tag, read-only.
- `Commit message`: generated или ручное сообщение.

Правило сообщения:

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

Если `Subject` пустой, используется вручную введенный `Commit message`.

Команды:

- `next version` - ищет tags `<series>-v*` и повышает последнюю semver.

- `git tag HEAD` - создает annotated tag на HEAD с generated message.
- `Update message` - пересобирает commit message из полей.
- `basket clear` - очищает in-memory basket.
- `add active project` - stages active Mirror root.
- `git add -- basket` - stages basket paths.
- `git restore --staged -- basket` - unstages basket paths.
- `git commit --only -- basket` - commits only basket paths.
- `git commit -m` - commits currently staged paths.

Basket привязан к текущему Git root. При смене root он очищается, чтобы не смешивать репозитории. Stage/commit проверяют активный projection profile и блокируют paths вне Hub allowlist.

11. Inspector: Branch

Branch содержит branch switching, compare, integration и 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 из Basket version fields.
- `git tag -n` - tag list no 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`.

Это visible templates: оператор видит точную команду и может ее отредактировать.

12. Inspector: Remote

Remote отвечает за remotes, push/pull/fetch и auth setup.

Поля remote form:

- **Platform**: `GitHub`, `GitLab`, `Codeberg`.
- **Remote name**: имя Git remote. Если пусто при save, станет `hidden_<platform>`.
- **Login / group**: owner/group path.
- **Repository**: репо name. Хвост `.git` снимается при build URL.
- **Remote URL**: явный URL. Если заполнен, он важнее generated URL.
- Recent selects для names, owners, repos, URLs.
- Use buttons применяют recent values явно, без автоматического наложения.

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` - runs fetch/prune и refresh status.

- `git remote -v` - показывает repo remotes.
- `git push all remotes` - Python перечисляет `git remote` и пушит `---follow-tags` в каждый remote последовательно.
- `git apply remotes.json` - добавляет или обновляет enabled remotes из `config/remotes.json`.
- `git origin push URLs` - настраивает `origin` с несколькими push URLs из enabled remotes.
- `build URL` - записывает generated SSH URL в `Remote URL`.
- `save remote` - пишет/обновляет enabled record в `config/remotes.json` и обновляет `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 `git@github.com`.
- `ssh -T GitLab` - non-interactive SSH probe `git@gitlab.com`.
- `gh auth login` - внешний терминал.
- `glab auth login` - внешний терминал.
- `Windows Credentials` - Windows Credential Manager.
- `GitKraken folder` - открыть текущий Git root folder.
- `VS Code` - открыть активный проект в VS Code.

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

13. Inspector: Editor

Поддержанные расширения:

```
.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 пишет UTF-8 только по явному действию пользователя и обновляет tree.

14. Inspector: Reader

Reader - общая зона summary. Сюда попадают MIRROR summaries, Git status, Auth Doctor summary, Storage summary и batch command summaries.

15. Inspector: Diff

Команды:

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

Diff рисуется как RedLine view: line numbers, hunks, additions, removals, metadata lines.

16. Inspector: History

Команды:

- **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**: копирует current history text.

17. Inspector: Details

Details показывает:

- project;
- tree scope;
- relative/full path;
- Git status;
- file/dir/missing type;
- size;
- modified time;
- Git blob metadata, если доступна.

Команды:

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

18. Inspector: Storage и Safety

Storage commands:

- **Check layout** - проверить Manager, Hub Data, Docs и full projects roots.
- **Scan projects** - просканировать выбранную parent folder и импортировать project entries.
- **Generate workspace** - создать **.code-workspace** для current project layers.
- **Copy JSON** - скопировать storage payload.
- **Open workspace** - открыть generated workspace.

External tools:

- **VS Code executable** - machine-local путь к Code.exe.
- **Pick** - file picker.
- **Save** - записать в **config/apps.local.json**.
- **Test** - проверить/запустить resolved VS Code command.

Safety commands:

- `Scan current root` - поиск secret-like files, embedded tokens, private keys, heavy extensions и large files.
- `Copy JSON` - скопировать safety payload.

Safety пропускает generated/runtime dirs: `.git`, `.venv`, `runtime`, `wheelhouse`, `node_modules`, `logs`, `output`, `backup`, `release`, `report`, `temp`, `tmp` и похожие.

19. Terminal / command dock

Нижний dock содержит:

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

Manual commands запускаются в current Git root, если он есть, иначе в manager root. Output стримится в terminal dock. Команда добавляется в history, если не заблокирована danger filter.

20. GitHub/GitLab remote workflow

Рекомендуемый SSH-flow:

1. Настроить SSH keys вне Hub Manager.
2. Проверить `ssh -T GitHub` и `ssh -T GitLab`.
3. Заполнить `Platform`, `Remote name`, `Login / group`, `Repository`.
4. Нажать `build URL`.
5. Нажать `save remote`.
6. Нажать `git apply remotes.json`.
7. Проверить `git remote -v`.
8. Использовать `git fetch --all --prune` для безопасного remote tracking update.
9. После commit/tag использовать `git push origin` или `git push all remotes`.

Для HTTPS использовать Git Credential Manager, GitHub CLI или GitLab CLI. Remote URL должен быть чистым:

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

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

`origin push URLs` полезен, когда fetch/pull должен идти из одного canonical remote, а push должен публиковать в несколько mirrors.

21. MIRROR и обслуживание конфигов

Повседневное обслуживание:

- `Preview MIRROR` перед реальным apply.
- `Apply MIRROR` с `Dry-run`, если есть сомнения.
- `Verify Mirror` после важных projection changes.
- `Safety Scan` перед public push или release.
- `Batch Git status` для здоровья реестра.
- `Clean projects.json` после переносов/удалений проектов.
- `Storage -> Check layout` после переноса Manager, Hub Data или Docs roots.
- `BLAKE3` после пересборки portable runtime.

Правила registry maintenance:

- `Clean projects.json` редактирует только config.
- Project scanner импортирует недостающие записи и пропускает duplicate sources.
- Scanner игнорирует runtime/build/cache folders и вложенные Hub/Docs roots.
- Storage check сообщает проблемы конфигурации, но не выдумывает новые paths.

22. Подробная карта Git-work функций

Этот раздел фиксирует именно Git-work слой: какие команды выполняются сразу, какие только ставятся в очередь, какие функции кода их обслуживают и какие guards включены.

22.1. Где выполняются Git-команды

Текущий Git root определяется функцией `current_git_root()` и равен текущему root дерева:


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

Поэтому перед запуском Quick/Branch/Remote важно проверить активный слой **Structure**. **add active project** является исключением: он всегда берет **mirror_root()** и stages активную Hub projection.

22.2. Прямые handlers

Прямые handlers сразу вызывают Python-обертки над Git:

- **refresh_git()** -> **git status --porcelain=v1 -b**.
- **stage_selected()** -> **git add -- <selected>**.
- **unstage_selected()** -> **git restore --staged -- <selected>**.
- **stage_active_project()** -> **git add -- .** в Mirror root после 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. Шаблоны в очереди

Шаблоны в очереди вызывают **queue_git_command()**. Они заполняют command area в Inspector и нижний terminal command input, добавляют команду в history и ждут ручного **Run**.

В очередь ставятся:

- `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 terminal dock, writes 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
```

Важный нюанс: опасные команды могут быть поставлены в очередь как видимые templates, чтобы оператор увидел и отредактировал точную команду. Но запуск неизменной опасной команды через `Run` блокируется 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 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")`: difference 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. Провисающие места и пропущенные частые команды

Программа уже закрывает большую часть everyday Git/MIRROR workflow, но видны следующие белые пятна.

Remote status and sync

- Нет компактной таблицы ahead/behind по remote.
- Нет кнопки `git fetch <remote> <branch>`.
- Нет selected remote push/pull: действия в основном `origin` или all remotes.
- Нет `git remote show <name>` inspector.
- Нет remote-группированного отчета fetch/prune.
- Нет UI для disable/remove saved remote из `remotes.json`.

Branch workflow

- Нет branch dropdown из `git branch --all`.

- Нет current branch badge внутри Branch pane.
- Нет `git switch -` для возврата на предыдущую ветку.
- Нет `git branch -d/-D <branch>` template.
- Нет upstream setup: `git push -u origin <branch>` или `git branch --set-upstream-to`.
- Нет визуального merge/rebase state indicator, кроме raw output.

Staging and commit

- В Diff нет явных selected path `stage` / `unstage` кнопок, хотя handlers в коде уже есть.
- В Basket нельзя удалить один path, только clear all.
- Нет amend-flow: `git commit --amend`.
- Нет commit dry-run/status preview именно для текущего basket.
- Commit profile violations уходят в terminal, а не в компактный список Basket.

History and recovery

- Reflog есть в Quick, но нет recovery helper `git checkout <sha> -- <path>`.
- Нет viewer для `git show <commit>:<path>`.
- Нет копирования commit hash из History.
- Нет tag delete templates, local/remote.

Auth

- Auth Doctor знает URL types/providers, но UI не показывает badges рядом с каждым configured remote.
- Нет guided SSH key creation/checklist.
- Нет отдельных `gh auth status` / `glab auth status` кнопок, они только внутри Auth Doctor.
- Нет HTTPS credential cleanup checklist, кроме открытия Windows Credentials.

Safety and release

- Safety Scan не связан gate-ом с push buttons.
- Нет release checklist command group.
- Нет `git archive`.
- Нет signed tag/commit option.

Storage and Docs

- Docs layer открывается, но нет отдельной docs-only projection action.
- `Sync Docs Back` отсутствует намеренно, но UI стоит явно подписать как отсутствующий/небезопасный workflow.
- Storage check показывает paths, но почти не предлагает repair actions кроме picker и scanner.

UI and ergonomics

- Многие commands требуют ручной `Run`: это безопасно, но медленно для частых benign операций.
- Risk commands блокируются token matching, а не typed confirmation dialog.
- Command cache global, не per project.
- Remote recent values global, не per provider/project.
- Small screen layout нужно продолжать проверять скриншотами после каждой правки.

24. Рекомендуемый следующий backlog

Небольшие low-risk улучшения:

1. Branch dropdown из `git branch --all`.
2. Шаблоны `git switch -` и `git push -u origin <branch>`.
3. Selected-path stage/unstage кнопки во вкладке Diff.
4. `basket remove selected` или кликабельные строки Basket.
5. `remote show <name>` и `fetch selected remote`.
6. Таблица Auth/Remote badges: name, provider, URL type, enabled, configured in repo, ls-remote ok.
7. Видимое stale Safety warning перед push.
8. `git commit --amend` как queued command, не direct action.
9. `git show <commit>:<path>` viewer в History/Diff.
10. Docs-only projection action, если Docs станет активным workflow.