# Audion Hub Manager — почему устроено так

English · О программе · Руководство · Устройство

**Содержание**


- Решение
- Почему раздельно
- Что из этого следует
- Решение
- Единственный источник правды
- Настоящая запись требует явного намерения
- Фильтр не может молча стать полной копией
- Слово проекта стоит над профилем
- Запись идёт по фазам, удаление — последним
- Три способа сравнения
- Расхождения — это не успех
- Мелочи, которые однажды дорого обошлись
- Проверки, которые должны оставаться зелёными
- Решение
- Что это значит на практике
- Исключение, которое исключением не является
- Решение
- Почему
- Что из этого следует
- Решение
- Почему
- Решение

- Почему

Решения, определившие программу. Каждое принято не из красоты, а потому что иначе было бы хуже.

---

# 1. Три слоя вместо одной большой папки

## Решение

Проект живёт в трёх слоях, которые могут лежать под общим корнем, но остаются раздельными по смыслу.

```
ИСХОДНИК     живой проект целиком
GIT-КОПИЯ    техническая отфильтрованная копия с Git
DOCS         нейтральная папка заметок и описаний
```

## Почему раздельно

**ИСХОДНИК полный.** Со сборками, кэшами, логами и настройками машины — иначе на нём нельзя работать. Такой папке история Git только мешает: её засоряет то, что пересобирается каждый день.

**GIT-КОПИЯ техническая.** Внутри — то, что пропускает профиль: исходный код, документация, настройки без секретов, запускающие файлы. Это слой для сравнения, коммитов, сервера и повторного использования кода.

**DOCS нейтральны.** Слой не сканирует тысячи файлов исходников. Там Markdown, указатели, обзоры, заметки. Его синхронизируют чем угодно — редактором, файловым менеджером, Obsidian, LogSeq, облаком или ничем. GIT-КОПИЯ ведётся через Git и на телефон не нужна.

## Что из этого следует

Слой DOCS отдельно не сверяется: он читающий, а каноническая техническая копия уже в GIT-КОПИИ. Сверка идёт только между ИСХОДНИКОМ и GIT-КОПИЕЙ.

# 2. Зеркало, которое не может испортить исходник

## Решение

Правила ниже собраны по следам настоящих поломок, и повторять их заново не нужно.

## Единственный источник правды

ИСХОДНИК — единственный источник правды. GIT-КОПИЯ производна: зеркало вправе удалять и пересобирать её файлы, но ИСХОДНИК не изменяет **никогда**.

ИСХОДНИК может быть обычным рабочим деревом Git — зеркалу до этого нет дела, оно читает только то, что разрешил профиль. GIT-КОПИЯ тоже дерево Git, и её `.git/**` защищён от обхода, удаления и обслуживания.

## Настоящая запись требует явного намерения

Профиль объявляет предпросмотр поведением по умолчанию, и командная строка не вправе это проигнорировать:

```
hub_cli.cmd mirror apply ПРОЕКТ             предпросмотр, если профиль так велит
hub_cli.cmd mirror apply ПРОЕКТ --apply     настоящая запись
hub_cli.cmd mirror apply ПРОЕКТ --dry-run   принудительный предпросмотр
```

В окне это галочка «сухой прогон» у ПРИМЕНИТЬ.

## Фильтр не может молча стать полной копией

Профиль обязан объявлять, что фильтры требуются и сколько масок минимум. Если масок нет — планирование падает, а не собирает полный дубль проекта. Полную копию разрешает только отдельный признак профиля, и ни один поставляемый профиль его не ставит.

## Слово проекта стоит над профилем

Профиль общий для всех проектов, `.gitignore` — слово самого проекта о том, что не хранить. Что он прячет, зеркало не копирует, а `check-paths` не предлагает в коммит. Иначе ключи в `config\keys\` или дерево сборки в `build-ui\` легли бы в GIT-КОПИЮ лишней копией на диске, даже если в историю они не попадут. Профиль при этом своих запретов не снимает: проект без `.gitignore` защищён им одним.

Догадки профиля о папках сборки уступают тому же слову. `app\` у Next.js — страницы, `build\` у Electron — значки, `bin\` — сценарии, `vendor\` у Go — закоммиченные зависимости. Где у проекта есть `.gitignore`, что сборка — решает он; догадки работают только без него. Запреты безопасности — секреты, двоичные файлы, тяжёлое — действуют всегда.

## Запись идёт по фазам, удаление — последним

```
1  создать разрешённые каталоги
2  скопировать и обновить файлы
3  проставить времена изменения
4  если ошибок и расхождений нет — удалить лишнее в GIT-КОПИИ
5  убрать опустевшие каталоги
6  маркеры пустых папок — только если профиль велит
```

**Удаление не начинается, пока копирование не завершилось успешно.** При сбое любой ранней фазы удаление пропускается, и отчёт это прямо называет. Это ровно тот случай, когда зеркало потеряло бы файлы, существующие только в нём; настройка профиля этого не отключает.

## Три способа сравнения

| Способ | Как сравнивает | Когда |
|---|---|---|
| быстрый | размер и время | черновая сверка |
| безопасный | размер и время, суммы — только при совпавшем размере и разном времени | обычная работа |
| строгий | суммы для всех файлов одного размера, даже при совпавшем времени | важные точки |

Безопасный способ быстр, но намеренно не ловит случай «тот же размер, то же время, другое содержимое». Для контрольных точек нужен строгий — его и держат поставляемые профили.

Суммы всегда помечены алгоритмом: `blake3:…` или `sha256:…`. SHA-256 — запасной вариант, а не равноценная замена.

## Расхождения — это не успех

Любое настоящее применение с ошибками или расхождениями возвращает ненулевой код выхода.

## Мелочи, которые однажды дорого обошлись

**Имена отчётов с микросекундами** — иначе быстрая автоматизация перезаписывает собственные отчёты.

**Область зеркалирования объявляется явно.** «Отфильтрованное» значит, что GIT-КОПИЯ приводится к ИСХОДНИКУ только по выбранным маскам; файлы вне области план не трогает.

**План перепроверяется перед записью.** Устаревший или поправленный план мог бы удалить по пути, пересекающемуся с ИСХОДНИКОМ, — правила и запрет пересечения проверяются заново.

## Проверки, которые должны оставаться зелёными

```
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
```

Профиль может держать пустые папки маркером `.gitkeep`: он ставится в каждую пустую папку, снимается, когда в ней появились настоящие файлы, и не ставится в скрытые технические каталоги. Поставляемые профили маркеров не ставят: служебные папки программа создаёт сама (`install\init_folders.cmd`), а пустые файлы-метки только засоряют историю.

# 3. Программа не хранит пароли

## Решение

Hub Manager не становится хранилищем токенов и паролей. Программа портативна, а учётные данные Git внутри портативной папки не живут.

Вход делегирован обычным средствам:

- ключи SSH и ssh-agent;
- диспетчер учётных данных для HTTPS (Git Credential Manager);
- командные утилиты GitHub и GitLab;
- VS Code и GitKraken — для ручного входа и наглядной работы.

Программа выполняет настоящие команды `git` и показывает их вывод целиком. Если вы уже вошли через любое из перечисленного, отправка и приём работают, а программа не знает ни одного секрета.

## Что это значит на практике

Удалённый репозиторий — это именованный адрес:

```
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
```

Адрес с паролем или токеном внутри программа не сохраняет. Проверка доступа опознаёт вид адреса и подсказывает, чем вы будете опознаны, — но остаётся проверкой, а не редактором учётных данных.

## Исключение, которое исключением не является

Для своих серверов Forgejo и Gitea программа принимает личный токен доступа: проверяет его запросом к серверу и передаёт вашему помощнику учётных данных. В файлы программы и в журнал токен не попадает, а дальше `git push` берёт его из хранилища сам. «Забыть токен» убирает его из хранилища этой машины.

Известные адреса серверов лежат в настройках открыто — без токенов.

# 4. Одна таблица Git-команд

## Решение

Все Git-команды программы — одна таблица, канон: команда с местами, риск, режим, слова на двух языках «что делает» и «как отменить». Окно не пишет Git-команд само: кнопка — запись канона, подпись кнопки — команда, как её набирают в терминале.

## Почему

**Программа учит, пока ей пользуются.** Человек видит, какая команда пошла, и читает, что она сделала и как её отменить. Через месяц он набирает её сам.

**Одно место — одна правда.** Тест гоняет каждую запись канона на свежей копии учебной лаборатории; таблицы в документах пишутся из канона и сверяются тестом. Кнопка, документ и проверка не могут разойтись.

**Риск виден до нажатия.** Читает, меняет, сеть, опасно — точкой у кнопки. Опасное спрашивает с командой целиком, тем, что сейчас не закоммичено, и способом отмены; редкое и переписывающее историю (`rebase -i`) — заготовка, которую дописывают сами.

## Что из этого следует

`pull` — только перемоткой: слияние человек делает сам и видит. В индекс и коммит идёт только то, что пропускает профиль проекта, — никакого `git add .`: иначе смысл фильтрации теряется на первом же коммите.

# 5. Окно — лицо консольной команды

## Решение

Всё, что окно делает вне Git, — реестр, зеркало, проверки, пакеты, лаборатория, Forgejo — идёт через `hub_cli.cmd` с одним JSON на ответ. Git окно зовёт само.

## Почему

То, что умеет окно, умеет и сценарий, и агент, и человек в консоли — без окна и без расхождений. Движок на Python проверен тестами и не переписывается ради окна. Git же окно зовёт напрямую: команда, которую оно показывает, и есть урок, а состояние репозитория читается слишком часто, чтобы ходить за ним через посредника.

# 6. Лаборатория вместо примеров

## Решение

Учебные репозитории не лежат в программе готовыми — их строит `lab.py`: десять репозиториев в известных состояниях, с постоянными авторами, датами и байтами файлов.

## Почему

Учиться надо на том, что можно сломать. Сломанное строится заново одной кнопкой, с теми же хэшами на любой машине, поэтому урок, тест и описание говорят об одних и тех же коммитах. Те же репозитории — испытательный стенд канона.