Audion Hub Manager

English · Руководство · Git-вики · Консольная команда · Устройство · Решения · История

Содержание

- Зачем это сделано
- Как это выглядит
- Что она умеет
- Запуск
- Что где лежит
- Описание проекта
- Подъём из зеркала
- Правила, которые нельзя нарушать
- Проверка

Портативная программа для Windows, чтобы вести проекты в Git и учиться ему на настоящей работе. Окно запускает настоящий `git`, показывает каждую команду целиком и объясняет простыми словами, что она сделала и как её отменить. Проект при этом живёт в трёх слоях: живой исходник, чистая GIT-КОПИЯ для истории и сервера, папка документов для чтения.

> Исходник остаётся целым. Зеркало остаётся обозримым. Документы остаются читаемыми. Git перестаёт быть страшным.

Зачем это сделано

У живого проекта два несовместимых свойства. Он должен быть полным — со сборками, кэшами, логами и настройками этой машины, иначе на нём нельзя работать. И он должен быть обозримым — чтобы историю можно было читать, изменения сравнивать, а копию поднимать, не разбирая, что здесь исходник, а что след вчерашней сборки.

Поэтому проект живёт в трёх слоях:

```
ИСХОДНИК     полный проект, единственный источник правды
GIT-КОПИЯ    чистое зеркало по профилю: код, документы, конфиги без секретов; свой
.git
DOCS         слой для чтения: Markdown и тексты для Obsidian, LogSeq или просто папки
```

GIT-КОПИЮ можно удалить и собрать заново. DOCS синхронизируют чем угодно. ИСХОДНИК зеркало не меняет никогда.

Вторая половина замысла — Git без заучивания. Все Git-команды программы собраны в одну таблицу (канон, 63 операции): команда, риск, что она делает и как её отменить, на двух языках. Кнопка окна — запись этой таблицы, подпись кнопки — команда так, как её набирают в терминале. Пользоваться и учиться можно одновременно.

Как это выглядит

Сверху — проект из реестра и переключатель слоя: ИСХОДНИК, GIT-КОПИЯ или DOCS. Git-команды идут в репозиторий выбранного слоя. Ниже — семь разделов:

Раздел	Что там
ИЗМЕНЕНИЯ	файлы по стадиям (конфликты, в индексе, изменены, новые), разница выбранного файла, сборка коммита: тип, область, версия, тег
ИСТОРИЯ	граф коммитов всех веток, коммит с его файлами, история файла, теги, reflog; revert, cherry-pick, reset
ВЕТКИ	ветки свои и сервера, сравнение, слияние, отложенные правки (stash), выход из застрявшего слияния
СИНХРОНИЗАЦИЯ	remote репозитория, fetch, pull, push, сборка адреса для GitHub, GitLab, Codeberg, Forgejo, Gitea, проверка входа, токен Forgejo
ЗЕРКАЛО	план ИСХОДНИК → GIT-КОПИЯ по профилю, применение, сверка BLAKE3
ПРОЕКТЫ	панель папок машины, реестр проектов с состоянием каждого слоя, пакетные проверки, поиск проектов в папке, клон, учебная лаборатория
ОБСЛУЖИВАНИЕ	проверка секретов и тяжёлых файлов, резерв bundle, здоровье репозитория, настройки Git, инструменты машины

Под страницей раздела — полоса его частых операций с полями: тема коммита, имя ветки, адрес. Внизу — терминал: в нём каждая команда целиком, её вывод в цвете Git, объяснение и итог с кодом выхода; строка команды принимает `git`, PowerShell программы или `cmd`. Строка подсказки называет следующий шаг: «В индексе 3: напишите тему внизу и нажмите КОММИТ».

Что она умеет

- **Git целиком для повседневной работы** — состояние, разница, индекс, коммит части файлов, теги версий, ветки, слияние, отложенные правки, история и граф, откат и спасение потерянного, remote, fetch, pull, push, резерв без сервера. Редкое и опасное — заготовками, которые дописывают и запускают осознанно.
- **Опасное спрашивает.** Вернуть файл, сбросить ветку, переписать последний коммит — окно показывает команду целиком, что сейчас не закоммичено и как отменить; `reset --hard` подтверждается вводом слова RESET.
- **Коммит по профилю.** Файл вне профиля проекта не попадает ни в индекс, ни в коммит: сборки, логи и секреты в историю не уходят.
- **Зеркало** — план, применение и сверка ИСХОДНИКА с GIT-КОПИЕЙ по суммам BLAKE3; подъём проекта из зеркала.
- **Реестр проектов** — поиск настоящих корней в общей папке, чистка записей без папок и дублей, состояние всех проектов разом.
- **Вход без хранения паролей** — SSH-ключи, Git Credential Manager, `gh` и `glab`; личный токен Forgejo и Gitea проверяется на сервере и уходит в хранилище Git, а не в файлы программы.
- **Учебная лаборатория** — десять репозиториев в известных состояниях: конфликт, отставание от сервера, потерянный коммит и другие. Строятся заново одной кнопкой, с теми же хэшами на любой машине.
- **Консольная команда** `hub_cli.cmd` — всё, что окно делает вне Git, без окна: один JSON на команду.

Запуск

```
Start.exe
```

Нужны Windows 10 или 11 (x64) и установленный Git (`git` в `PATH`); `gh`, `glab` и VS Code — по желанию. Окно, PowerShell программы и Python движка лежат в папке программы. Если

`runtime\` пуст, окружение Python собирает сборщик: `builder_main.cmd` → [01].

Начать с лаборатории: ПРОЕКТЫ → УЧЕБНАЯ ЛАБОРАТОРИЯ → «создать заново», затем выбрать `lab_01` в списке проектов.

Что где лежит

Путь	Что
`Start.exe`	запуск окна
`App\`	окно `AudionHubManager.exe` и PowerShell 7 программы `AudionPs.exe`
`hub_cli.cmd`, `system_core\`	консольная команда и движок на Python: реестр, зеркало, проверки, Forgejo, лаборатория, канон Git
`config\`	реестр, профили зеркала, известные remote и серверы Forgejo (без токенов), раскладка хранилища, настройки окна
`examples\`	демонстрационный проект и учебная лаборатория `examples\lab`
`logs\`	отчёты зеркала, сверок и проверок: `logs\<проект>\`
`builder_main.cmd`, `install\`	сборщик: окружение Python, проверки, сборка окна
`runtime\`, `wheelhouse\`	Python программы и его пакеты
`docs\`	эти документы

Описание проекта

Запись реестра — `hub_projects.json` в корне проектов, у каждого корня свой:

```
{
  "id": "audion_hub_manager",
  "title": "Audion Hub Manager",
  "source_path": "Audion Hub Manager",
  "projection_path": "S:/Audion/Hub Data/Audion Hub Manager",
  "docs_path": "S:/Audion/Docs/Projects/Audion Hub Manager",
  "profile": "audion_python_project_projection",
  "default_branch": "main"
}
```

Поле	Что
`source_path`	ИСХОДНИК; относительный путь считается от корня проектов
`projection_path`	GIT-КОПИЯ, свой `.git`, пересобирается зеркалом
`docs_path`	DOCS, необязательно
`profile`	что проходит в GIT-КОПИЮ и в коммит: профиль из `config\projection_profiles.json`
`default_branch`	ветка, которую берут pull и push, пока у ветки нет пары на сервере; с ней `git init` начинает историю

Пути внутри корня проектов программа пишет от него (`Hub Data/Проект`): корень переезжает вместе со своим реестром в другую папку или на другой диск. GIT-КОПИЮ можно держать и в другом месте или на другом диске — тогда путь пишется полностью. Корень по умолчанию — папка рядом с программой; другой выбирает кнопка-папка колонки проектов, и её меню импортирует реестр из файла и экспортирует его в файл.

Запись заводит поиск: ПРОЕКТЫ → папка в панели ПАПКИ → «найти в папке» или `hub_cli.cmd projects scan ПАПКА --import`. Папка, которая сама проект (у неё есть `.git`, `README.md`, `pyproject.toml` и подобное), находит себя; общая папка без таких признаков — проекты в своих подпапках. Склонированный проект вносит кнопка «в реестр». Папки и ветку записи правят ПРОЕКТЫ → ЗАПИСЬ или `hub_cli.cmd projects set`.

Подъём из зеркала

ИСХОДНИК потерян, GIT-КОПИЯ жива:

```
robocopy "<GIT-КОПИЯ>\<Проект>" "<корень>\<Проект>" /E /COPY:DAT /DCOPY:DAT /R:1 /W:1
/XJ
```

Поднимется то, что проходило профиль; рантаймы и сборки пересобираются заново.

Правила, которые нельзя нарушать

1. ИСХОДНИК — источник правды; зеркало его не меняет.
2. GIT-КОПИЯ производна: её можно удалить и собрать заново.
3. `.git/**` защищён от сканирования, удаления и обслуживания зеркалом.
4. Настоящая запись зеркала требует явного намерения.
5. В индекс и коммит идёт только то, что пропускает профиль.
6. Токены, пароли, закрытые ключи и пути этой машины не попадают в общие настройки.
7. Машинное — в `*.local.json`, `.env` и прочих исключённых файлах.
8. Отчёты сверки пишутся только в `logs\<проект>\`.
9. Слой DOCS отдельно не сверяется: его техническая копия уже в GIT-КОПИИ.

Проверка

```
runtime\python.exe -m pytest -q tests
install\Build-App.cmd
install\verify_portable_env.cmd
```

Тесты гоняют движок, консольную команду и каждую команду канона на свежей копии лаборатории. Сборка окна заканчивается его самопроверкой; `verify_portable_env.cmd` проверяет окружение, консольную команду, `Start.exe` и собранное окно.