Audion Hub Manager — user guide

Русский · About · Git wiki · Console command · History

Contents

- First start
- The window
- CHANGES (ИЗМЕНЕНИЯ)
- HISTORY (ИСТОРИЯ)
- BRANCHES (ВЕТКИ)
- SYNC (СИНХРОНИЗАЦИЯ)
- MIRROR (ЗЕРКАЛО)
- PROJECTS (ПРОЕКТЫ)
- SERVICE (ОБСЛУЖИВАНИЕ)
- An ordinary day
- Keys
- What lies where

The window speaks Russian and English (RU / EN at the top right); this guide names its labels as the English window shows them, with the Russian label in brackets.

First start

`Start.exe` opens the window. If the project registry is empty, the hint line points to PROJECTS (ПРОЕКТЫ): your projects can be found in a folder there, or the Git lab built. If `runtime\` is empty, the window says the console command is not available: `builder_main.cmd` → [01] builds the Python environment.

To practise first: PROJECTS → GIT LAB (УЧЕБНАЯ ЛАБОРАТОРИЯ) → "build anew" (создать заново), then `lab_01` in the list of projects.

The window

```
┌ header: EXPLAIN · language · theme · settings
┐
├ sections: CHANGES · HISTORY · BRANCHES · SYNC · MIRROR · PROJECTS · SERVICE
┤
├ the layer SOURCE / GIT COPY / DOCS of the project, its path
┤
├ the column of projects │ the page of a section: what is looked at
┤
├ the bar of a section: what is done most, with its fields
┤
├ the hint line: the next step
┤
└ terminal: the command, its output, the explanation, the outcome · command line GIT
/ PS / CMD · STOP ┘
```

The column of projects on the left holds the projects of the registry. A click on a project makes it the current one: the Git commands go to it. A project is a record of the registry: it switches the whole set at once — SOURCE, GIT COPY, DOCS, the mirror profile and the default branch. The square by a project marks it for the work done to several at once, ALL and CLEAR mark all or none; the badge in the head says "marked | all".

At the foot of the column is **COMMIT THE MARKED** (КОММИТ ОТМЕЧЕННЫХ): one subject for all, and every marked project in turn goes through a checkpoint in its GIT COPY — the mirror SOURCE → GIT COPY, the check by BLAKE3, the check for secrets, the check of the paths by the profile, add and commit. Every command goes to the terminal as a button would run it, with the name of the project in its head. A project where a step says "no" stops there with the reason, the others go on; a GIT COPY with nothing new is no error. A GIT COPY without a repository of its own is passed over: its first commit is made step by step, in MIRROR and CHANGES. STOP stops the command that goes, and the projects after it are not begun. The outcome is one line: done, without changes, stopped.

To the right of the badge is **the root of the projects**, the folder the live projects lie in: the folder panel and the search for projects look there, and the registry of projects `hub_projects.json` lies in it — each root has its own. The menu of the root:

Item	What
Make it the root	the folder chosen in the folder panel of PROJECTS becomes the root; the column shows its registry

Item	What
Give the default root back	the folder beside the program and its registry
Import the registry from a file…	the projects of the file are added to the registry of the root: their paths are read from where the file lies and written from the root; projects the registry already has stay as they are
Export the registry to a file…	the registry of the root into a file, every path in full: the file reads the same wherever it goes

The choice of the root is kept for this machine in `config\storage_layout.local.json`. When projects of the new root keep their GIT COPY outside the root of the GIT COPIES, the terminal names them in a line under the command. The chevron "‹" folds the column, "›" at the edge of the page opens it. In a window from 1550 points the column stands beside the page and moves it aside, folded or open as the window remembers; in a narrower one it is folded and opens over the page, and a chosen project, the chevron, Esc or a click past it folds it again. A long name or message does not break off in an ellipsis but fades out at its end; the whole of it is in the tip.

The layer is where Git commands go:

Layer	What it is
SOURCE (ИСХОДНИК)	the whole live project, the source of truth
GIT COPY (GIT-КОПИЯ)	a clean mirror of the project by its profile, its own `.git`: a clean history for the server
DOCS	a layer to read: Markdown and texts

Beside the layer: its path and state — "N changed", "no git", "no folder", "inside another" (the folder lies inside another repository, and the commands would go there). The buttons beside open the layer in Explorer, in VS Code and in an outer terminal with `git status`.

EXPLAIN (ОБЪЯСНЯТЬ) — a line under every command of the terminal: what it did and how to undo it. **Theme** — Audion Vault Dark (the default), Audion Code Dark, Audion Dark, Audion Light. **Settings** — the size of the window's text and of the terminal, confirming `reset --hard` by typing the word RESET, the paths of the settings and the logs. **About** — the number, the folders, the Git canon, the licence.

The terminal shows every command whole, its output in Git's colours and the outcome: "Done" or an error with its exit code. The command line works in three modes: GIT (type `status`, `log --oneline`

or a whole `git …`), PS (the program's PowerShell 7), CMD (`cmd.exe` in UTF-8). All three run in the root of the repository of the chosen layer. The arrows bring back former commands; the list of former commands can pin them. A dangerous command typed by hand is asked about too: a Git command is judged by its arguments, inside a line of PS or CMD as well (`reset --hard`, `push --force`, `restore`, `clean -f`, `branch -D`, `rebase`, `commit --amend`…), and `reset --hard` waits for the word RESET as from the button. While a command runs, the next waits its turn; STOP (СТОП) stops the running one with everything it started. The terminal's buttons: the `logs` folder, all the text to the clipboard, clear, fill the whole window.

Every button has a dot of its risk (reads, changes, network, danger) and a tip: the command, what it does, how to undo it.

CHANGES (ИЗМЕНЕНИЯ)

The strip over the list is the four places of Git: WORKING FOLDER, INDEX, HISTORY, SERVER, with their counts. A click on the working folder or the index leaves only their files in the list; on the history or the server it opens the section.

The list of files has two views:

- **CHANGES (ИЗМЕНЕНИЯ)** — only what differs from the last commit, in groups: CONFLICTS, IN THE INDEX, CHANGED, NEW;
- **ALL FILES (ВСЕ ФАЙЛЫ)** — the folders of the layer whole, with Git's letter at a changed file: Enter — into a folder, Backspace — up.

Space or Insert marks a file; the marking menu takes every changed, every new, every indexed, `*.md` or clears all. The "find" field narrows the list. A file outside the project's profile is marked "outside the profile": it goes neither into the index nor into the GIT COPY.

On the right, the DIFF (РАЗНИЦА) of the chosen file: NOT IN THE INDEX, IN THE INDEX or AGAINST HEAD; what is added in green, what is taken away in coral. The history of the file and VS Code are reached from there.

The **COMMIT (КОММИТ)** block puts the message together: TYPE (docs, code, fix, ui, test, chore, audit), SCOPE, VERSION (series, number, patch / minor / major, "set a tag"), the ticks "only the marked" and "fix the last one (--amend)". The whole message is shown in a frame. The subject is typed in the bar below.

Bar	Key	What
refresh (обновить)	F2	`git status` into the terminal
to the index (в индекс)	F5	the marked files, with none marked the chosen one; files outside the profile are not taken
from the index (из индекса)	F6	off the index, the edits stay; before the first commit `git rm --cached`, there is nothing to compare with yet
restore (вернуть)	F8	bring a file back to the index — danger, the window asks
subject of the commit		what was done and why
COMMIT (КОММИТ)	Ctrl+Enter	write the index into the history; before that the window checks the indexed files by the profile
push		send to the server

HISTORY (ИСТОРИЯ)

The tabs of the page: GRAPH (every branch or only the current one), FILE (the history of the file chosen in CHANGES), TAGS, REFLOG (the log of moves, where "lost" commits are seen too). The field "search the messages" keeps the commits with that text in the message, the author or the hash.

The chosen commit is shown on the right: author, time, parents, files. Its actions are grouped by what they cost:

- CHANGES NOTHING — show the commit;
- A NEW COMMIT ON TOP — revert, cherry-pick into the current branch;
- REWRITES HISTORY — take the last off (soft), reset to here (hard): the window asks.

The bar: a name of a branch or a tag for the chosen commit, "show" (F3), "log", reflog, "a branch from here", "a tag here", revert.

BRANCHES (ВЕТКИ)

The page: LOCAL, ON THE SERVER, TAGS, PUT ASIDE (STASH). The chosen branch shows where it stands against its pair on the server ("commits only you have: 2", "the branches parted: a merge is needed") and its actions: switch, compare with the current one, merge into the current one, delete,

`push -u`. A server's branch has "merge into the current one"; to work in it, its name without `origin/` is typed below and "create and switch" pressed — Git ties the new branch to the server's.

The block "IF STUCK" (ЕСЛИ ЗАСТРЯЛИ): `git merge --abort` and `git cherry-pick --abort` bring everything back as it was before the merge or the pick; `rebase -i` is a template to rewrite your own commits that are not sent.

The bar: the name of a new branch, "create and switch" (F7), "back", a note for what is put aside, "put aside", "take out"; `git branch -vv`, `git stash list`.

SYNC (СИНХРОНИЗАЦИЯ)

The list of the repository's remotes: where each takes from and sends to. The **NEW REMOTE** block builds an address: platform (GitHub, GitLab, Codeberg, Forgejo, Gitea, your own server), kind (SSH — by a key, HTTPS — through Git Credential Manager or `gh` / `glab`), the remote's name, owner, repository; for your own servers, the address and the SSH port. The address can be "written into the repository", added to `origin` as one more push address, or "saved to remotes.json" — the program's list, from which it can be written into any repository. An address with a password or a token inside is not saved.

The **ACCESS (ДОСТУП)** block: "check all" — Git, the name and e-mail of commits, `gh`, `glab`, SSH to GitHub and GitLab, the remote addresses; who signs the commits; `ssh -T` to GitHub and GitLab; signing in by GitHub CLI and GitLab CLI in an outer terminal; the Windows Credential Manager. The window never asks for passwords and never keeps them.

Your own Forgejo or Gitea server. A personal access token is made on the server: Settings → Applications → Generate New Token. The window checks it by a request to the server and hands it to Git's credential store (Credential Manager); it is written neither into the program's files nor into the log. After that `git push` over HTTPS takes it by itself. Buttons: "is the server there?", "save the token", "who am I", "my repositories", "forget the token" (removes it from this machine; on the server it stays valid until revoked there).

CI: `gh run list`, `gh run watch`, `glab ci status`.

The bar: "fetch all" (F2), pull, "to every remote", PUSH with the count of what is not sent. pull takes by fast-forward only: if the branches parted it refuses — then BRANCHES → merge, then PUSH. PUSH sends the current branch into its pair on the server; when their names differ, the command names both (`topic:docs`).

MIRROR (ЗЕРКАЛО)

The mirror brings the GIT COPY to the SOURCE by the project's profile. The SOURCE does not change, the copy's `.git` is protected.

The PROFILE block shows what the profile lets through and what not, the way of comparing and the kind of mirror ("an exact mirror: what is extra is removed"). "plan" (F2) builds a table: NEW, UPDATE, DELETE EXTRA, UNCHANGED, OUTSIDE THE PROFILE — with the path and the reason; nothing is written. APPLY carries the plan out; with the tick "dry run" it only shows what it would write. "verify BLAKE3" compares SOURCE and GIT COPY by their sums. The reports go to `logs\<project>\`.

After applying: switch the layer to GIT COPY and commit it in CHANGES.

PROJECTS (ПРОЕКТЫ)

On the left, FOLDERS (ПАПКИ) — the file panel of this machine. Over it, the places: DISKS, the root of the projects, the GIT COPIES (`Hub Data`), DOCS when that folder exists. Enter or a double click goes into a folder, Backspace goes up, the field "find" narrows the list. A folder shows what it is to the registry — the SOURCE, the GIT COPY or the DOCS of some project — or that it holds a repository of its own (`git`) the registry does not have. The chosen folder is the row under the cursor, or with `..` under it the folder looked at; its path is in the line under the list.

REGISTRY — the table of projects: branch, state, GIT COPY, layers. RECORD (ЗАПИСЬ) — the folders of SOURCE, GIT COPY and DOCS and the default branch of the chosen project; the button by a field takes the folder chosen in the panel into it. "Save" writes them into the registry of the root of the projects (paths inside the root are written from it, and the root moves together with its registry), "remove from the registry" (убрать из реестра) takes the record out and leaves the folders alone.

The bar:

Button	What
status of all (F2)	`git status` of every project — in the GIT COPY when it has a `.git` of its own, otherwise in the SOURCE: clean, changed, errors
mirror plans	the mirror plan of each project
secrets	the scan for secrets and heavy files in every project
verify	the BLAKE3 check of SOURCE and GIT COPY of every project

Button	What
find in a folder	finds projects in the folder chosen in the panel and adds them to the registry; the window names the folder first; the folders are not touched
clone	an address → a clone in the folder chosen in the panel; the window names the address and the place first; "to the registry" enters it
workspace	one `.code-workspace` for VS Code with every project
clean the registry	drops records without a SOURCE folder and duplicates; shows what goes before writing

"Find in a folder": a folder that is a project itself (it has `.git`, `README.md`, `pyproject.toml` or the like) finds itself; a shared folder without such marks finds the projects in its subfolders. The GIT COPY of a found project is offered in the folder `Hub Data` beside the program folder; its path is edited in RECORD.

GIT LAB — ten repositories to learn on in `examples\lab`; "build anew" builds them (the former ones to the Recycle Bin) and enters them into the registry, "open LAB.md" — what to try in each. In detail — the Git wiki.

SERVICE (ОБСЛУЖИВАНИЕ)

Block	What
CHECKS	"secrets" (F2) — files like secrets (`.env`, `*.pem`), keys and tokens in text, binaries over 25 MB, marked whether the file would go out with a push; the mirror check; "storage" — the roots of the program, the GIT COPIES, DOCS and the projects; the hash engine BLAKE3; the folder of reports
REPORTS	the outcome of the last checks
BACKUP	`git bundle create` — the whole history of a branch and its tags in one file
HEALTH OF THE REPOSITORY	`git fsck --full`, `git gc`, what `git clean` would remove
GIT SETTINGS	who signs the commits, every setting with its file, the root of the repository

`git init` is there too — for a layer that is not a repository yet.

An ordinary day

One machine. CHANGES → mark → F5 → subject → COMMIT → push.

A project with a GIT COPY. MIRROR → plan → APPLY → layer GIT COPY → CHANGES → mark every changed and new → F5 → subject and version → COMMIT with a tag → PUSH.

Two machines.

```
machine A: commit → tag → push
machine B: fetch → pull (fast-forward)
```

fetch only learns what is new and touches nothing of yours; pull adds the new commits when you want it.

Keys

Key	Where	What
F2	every section	the main read of the section: status, fetch, plan, status of all, secrets
F3	HISTORY	show the commit
F5 / F6 / F8	CHANGES	to the index / from the index / restore
F7	BRANCHES	create a branch and switch
Ctrl+Enter	CHANGES	COMMIT
Space, Insert	the list of files	mark
Enter, Backspace	ALL FILES	into a folder, up
Enter, ↑ ↓	the command line	run, former commands
Esc		close the settings or About, clear the filter

What lies where

File	What
`<root of the projects>\hub_projects.json`	the project registry of this root; without it the program makes it: for the default root from `config\projects.example.json`, for a root of one's own empty
`config\storage_layout.local.json`	the root of the projects chosen on this machine
`config\projection_profiles.json`	the mirror profiles
`config\remotes.json`	saved remotes, without passwords
`config\forgejo_hosts.json`	the Forgejo and Gitea servers of this machine, without tokens; without the file the list is empty
`config\window_settings.json`	theme, language, fonts and sizes, section and layer, EXPLAIN, type, scope and series of a commit
`config\terminal_commands.json`	former commands of the terminal's line
`logs\`	the window's log; `logs\<project>\` — the reports of the mirror and the checks