Audion Hub Manager — Git without learning the commands by heart

Русский · About · User guide · Console command

Contents

- Four places where the work lives
- Git words in plain words
- The risk and the mode of a button
- The canon by section
- Common situations and what to do
- The commit message
- The Git lab

Every button of the window runs a real `git` command and shows it whole. Under the command the terminal says in plain words what it did and how to undo it. All the commands of the program are one table — the Git canon (`system_core\core\git_canon.py`); the tables below are written from it and always match what the window does.

The window speaks Russian and English (RU / EN at the top right). This page names its labels as the English window shows them and gives the Russian label in brackets the first time.

Four places where the work lives

```
WORKING FOLDER  ──add──▶  INDEX  ──commit──▶  HISTORY  ──push──▶  SERVER
files on disk            "the desk before     commits on          GitHub, GitLab,
                          a commit"           this computer       Forgejo
      ◀──restore──           ◀──restore --staged──     ◀──pull / fetch──
```

The CHANGES section (ИЗМЕНЕНИЯ) shows these four places as a strip over the list of files: how many are changed, how many lie in the index, whether there are commits, what is not sent. A click on a place

shows only its files or opens its section.

Place	What is there	In	Back
working folder	the files as they lie on disk	edit a file	`git restore` — back to the index
index	what goes into the next commit	`git add` (F5)	`git restore --staged` (F6)
history	the commits of this computer	`git commit` (COMMIT)	`git reset --soft HEAD~1` — take the last one off
server	the commits others see	`git push`	what is sent is undone by `git revert`, not by rewriting

Git words in plain words

Word	In plain words
repository	a folder with a history: a hidden `.git` lies beside the files
commit	an entry in the history: which files became what, who and why
index (staging)	the desk before a commit: what is put on it goes into the next entry
branch	a draft: a separate line of work that does not get in the way of others
`main`	the fair copy: the main branch releases are made from
merge	take a draft into the fair copy
conflict	two branches changed one place differently; Git asks which to keep
remote, `origin`	the server one sends to and takes from; `origin` is the main one
push / pull / fetch	send your commits / take others' and add them on / only learn what is new
tag	a named version mark on a commit: `projection-v0.1.2`
stash	a drawer for edits that stand in the way of going to another branch
HEAD	where you are: the commit the working folder stands on

Word	In plain words
reflog	the log of HEAD's moves: even "lost" commits are found in it
pull request (PR, MR)	a request to take a branch into `main`; made on the site of GitHub, GitLab or Forgejo

The risk and the mode of a button

Every button has a dot of its risk and a tip with its command, its meaning and how to undo it.

Risk	Means
reads	changes nothing
changes	changes only this repository
network	talks to a server
danger	can lose work or rewrite history — the window asks

Mode	Means
at once	the command runs on the click
asks first	a question first: the whole command, what is not committed now and how to undo it; `reset --hard` is confirmed by typing the word RESET
template	the command goes to the terminal's line with its places marked; one finishes and runs it oneself

The EXPLAIN switch (ОБЪЯСНЯТЬ) in the header of the window turns on a line under every command of the terminal: what it did and how to undo it. Once it is familiar, it is turned off.

The canon by section

Places in the commands: `‹path›` and `‹paths›` — files, `‹commit›` — a commit, `‹branch›` — a branch, `‹new_branch›` — the name of a new branch, `‹remote›` — the name of a server, `‹url›` — its address, `‹message›` — a message, `‹tag›` — a version mark, `‹range›` — a chain of commits, `‹dest›` — the folder of a clone, `‹bundle›` — the backup file. The window fills them from what is chosen: a file, a commit, a branch, the fields.

CHANGES (ИЗМЕНЕНИЯ)

Command	What it does	How to undo it	Risk
`git status --short --branch`	What changed, what is in the index and how far the branch is ahead of or behind its remote.	Changes nothing.	reads · at once
`git diff -- <path>`	How the file in the working folder differs from the index: what is not yet prepared for the commit.	Changes nothing.	reads · at once
`git diff --cached -- <path>`	What of the file is already in the index and goes into the next commit.	Changes nothing.	reads · at once
`git diff HEAD -- <path>`	Every change of the file against the last commit, in the index or not.	Changes nothing.	reads · at once
`git blame -- <path>`	Who wrote each line of the file and in which commit.	Changes nothing.	reads · at once
`git add -- <paths>`	Puts the files into the index, "on the table before the commit". The next commit takes exactly what the index holds.	Undo: git restore --staged -- ‹files›	changes · at once
`git add -- <paths>`	Puts every change of the layer into the index, but only the files the project profile allows: logs, builds and secrets stay out.	Undo: git restore --staged -- .	changes · at once
`git restore --staged -- <paths>`	Takes the files off the table: they stay changed in the working folder but leave the commit.	Undo: git add -- ‹files›	changes · at once
`git rm --cached -- <paths>`	Takes the files off the table while the repository has no commit yet: they leave the index and stay in the working folder.	Undo: git add -- ‹files›	changes · at once
`git restore -- <paths>`	Returns the files in the working folder to what the index holds. Their uncommitted edits are lost.	Cannot be undone: those edits are in no commit.	danger · asks first

Command	What it does	How to undo it	Risk
`git commit -m ‹message›`	Writes everything in the index into the history as one snapshot with a message.	Undo the last one: git reset --soft HEAD~1 (the files go back to the index).	changes · at once
`git commit -- only -m ‹message› -- ‹paths›`	A commit of the marked files only, whatever else the index holds.	Undo: git reset --soft HEAD~1	changes · at once
`git commit -- amend -m ‹message›`	Rewrites the last commit: a new message and whatever the index holds now. If the commit was pushed, the history parts from the remote.	Undo: git reset --soft HEAD@{1}	danger · asks first
`git tag -a ‹tag› -m ‹message›`	Puts a named version mark on the current commit. A push with --follow-tags sends it with the branch.	Undo: git tag -d ‹tag›	changes · at once

HISTORY (ИСТОРИЯ)

Command	What it does	How to undo it	Risk
`git log --oneline --decorate -20`	The last 20 commits, one line each.	Changes nothing.	reads · at once
`git log --graph --oneline --decorate -- all -50`	The graph of every branch: where they parted and where they met.	Changes nothing.	reads · at once
`git log --date=short -- pretty=format:%h%x09%ad%x09%an%x09%s -40 -- ‹path›`	The history of one file: which commits changed it.	Changes nothing.	reads · at once
`git reflog --date=local -20`	The journal of HEAD moves, the life buoy: even "lost" commits show here.	Changes nothing.	reads · at once

Command	What it does	How to undo it	Risk
`git show --stat ‹commit›`	What a commit changed: message, author, files.	Changes nothing.	reads · at once
`git tag -n --sort=-creatordate`	Every tag with its note, newest first.	Changes nothing.	reads · at once
`git tag -a ‹tag› ‹commit› -m ‹message›`	Puts a version mark on the chosen commit.	Undo: git tag -d ‹tag›	changes · at once
`git switch -c ‹new_branch› ‹commit›`	Makes a branch at the chosen commit and moves to it: the way to rescue a "lost" commit from the reflog.	Delete: git branch -d ‹branch›	changes · at once
`git revert --no-edit ‹commit›`	Makes a NEW commit that undoes the chosen one. History is not rewritten, so it suits pushed commits too.	Undo: git revert ‹the new commit›	changes · at once
`git cherry-pick ‹commit›`	Copies the changes of the chosen commit onto the current branch as a new commit.	On a conflict: git cherry-pick --abort	changes · at once
`git reset --soft HEAD~1`	Removes the last commit and leaves its files in the index: rewrite the message or split the commit.	Undo: git reset --soft HEAD@{1}	danger · asks first
`git reset --hard ‹commit›`	Moves the branch to the chosen commit and wipes EVERY uncommitted change of the working folder and the index.	The commits stay in the reflog; uncommitted work is gone for good.	danger · asks first

BRANCHES (ВЕТКИ)

Command	What it does	How to undo it	Risk
`git branch -vv`	The branches, their partners on the server and how far ahead or behind they are.	Changes nothing.	reads · at once
`git switch ‹branch›`	Switches the working folder to another branch. Uncommitted edits travel along, or Git refuses if they are in the way.	Go back: git switch -	changes · at once
`git switch -c ‹new_branch›`	Makes a branch at the current commit and moves to it.	Delete: git branch -d ‹branch›	changes · at once
`git switch -`	Goes back to the previous branch.	Once more goes back again.	changes · at once
`git log --left-right --graph --cherry-pick --oneline HEAD...‹branch›`	Which commits only the current branch has (<) and which only the chosen one has (>).	Changes nothing.	reads · at once
`git merge --no-ff --no-edit ‹branch›`	Merges the chosen branch into the current one with a merge commit of its own, so the graph shows where the work came from.	On a conflict: git merge --abort	changes · at once
`git branch -d ‹branch›`	Deletes a merged branch. Git refuses an unmerged one.	Bring back: git switch -c ‹branch› ‹hash from the reflog›	changes · at once
`git stash push -u -m ‹message›`	Puts every uncommitted edit and new file into the "drawer" and leaves the working folder clean.	Bring back: git stash pop	changes · at once
`git stash pop`	Takes the last put-aside edits back into the working folder.	Put aside again: git stash push	changes · at once
`git stash list`	What lies in the drawer.	Changes nothing.	reads · at once

Command	What it does	How to undo it	Risk
`git rebase -i ‹range›`	Rewrites a chain of commits: squash, reorder, reword. Only for commits not yet pushed.	During it: git rebase --abort; after it: through the reflog.	danger · template
`git merge --abort`	Stops a merge with conflicts and puts everything back as it was before it.	—	changes · at once
`git cherry-pick --abort`	Stops a cherry-pick with conflicts and puts everything back as it was before it.	—	changes · at once

SYNC (СИНХРОНИЗАЦИЯ)

Command	What it does	How to undo it	Risk
`git fetch --all --prune`	Downloads the news of every remote without touching your branches and files. The safest network command.	Changes nothing of yours.	network · at once
`git fetch --prune ‹remote›`	Learns what is new on one server without touching your branches and files.	Changes nothing of yours.	network · at once
`git pull --ff-only ‹remote› ‹branch›`	Takes new commits from the server only when they can simply be added on top, without a merge.	Refuses when the branches have parted: then BRANCHES, merge.	network · at once
`git push --follow-tags ‹remote› ‹branch›`	Sends the commits of the current branch and the tags to the server, into its pair. When the names differ, the command names both: ‹branch›:‹pair›.	What was pushed is undone by a new commit (revert), not by rewriting.	network · at once
`git push -u ‹remote› ‹branch›`	Sends a new branch and ties it to its partner on the server: push and pull need no names after that.	Delete on the server: git push ‹remote› --delete ‹branch›	network · at once
`git push --follow-tags ‹remote› ‹branch›`	Sends the branch and tags to every remote of the repository, one after another.	—	network · at once

Command	What it does	How to undo it	Risk
`git remote -v`	Where the repository can push to and fetch from.	Changes nothing.	reads · at once
`git remote add ‹remote› ‹url›`	Writes a remote of the saved list into the repository.	Remove: git remote remove ‹name›	changes · at once
`git remote set-url ‹remote› ‹url›`	Changes the address of a remote the repository already has.	Back: git remote set-url ‹name› ‹old address›	changes · at once
`git remote set-url --add --push ‹remote› ‹url›`	One remote, several push addresses: fetch from one, push to several mirrors.	Remove: git remote set-url --delete --push ‹remote› ‹address›	changes · at once
`ssh -T -o BatchMode=yes git@github.com`	Checks whether GitHub lets you in by your SSH key.	Changes nothing.	network · at once
`ssh -T -o BatchMode=yes git@gitlab.com`	Checks whether GitLab lets you in by your SSH key.	Changes nothing.	network · at once
`gh run list --limit 15`	The last runs of GitHub Actions.	Changes nothing.	network · at once
`gh run watch`	Follows the running GitHub Actions run to its end.	Changes nothing.	network · at once
`glab ci status`	The state of the GitLab CI pipeline.	Changes nothing.	network · at once

PROJECTS (ПРОЕКТЫ)

Command	What it does	How to undo it	Risk
`git clone ‹url› ‹dest›`	Downloads a repository whole into a new folder.	Delete the folder.	network · at once

SERVICE (ОБСЛУЖИВАНИЕ)

Command	What it does	How to undo it	Risk
`git init -b ‹branch›`	Makes an empty repository (the .git folder) in the chosen layer; its first branch is the project's branch of the registry, main as a rule.	Delete the .git folder.	changes · at once
`git rev-parse --show-toplevel`	The root of the repository the commands run in.	Changes nothing.	reads · at once
`git bundle create ‹bundle› ‹branch› --tags`	The whole history of the branch and its tags in one file: a backup without a server. Restored by git clone ‹file›.	Delete the file.	changes · at once
`git clean -nd`	Shows which untracked files git clean would delete. Deletes nothing itself.	Changes nothing.	reads · at once
`git gc`	Packs the repository database. Safe.	—	changes · at once
`git fsck --full`	Checks the integrity of the repository database.	Changes nothing.	reads · at once
`git config --list --show-origin`	Every Git setting and the file each one comes from.	Changes nothing.	reads · at once
`git config --show-origin --get-regexp user\.`	The name and mail your commits will carry.	Changes nothing.	reads · at once

Common situations and what to do

Save the work. CHANGES: mark the files with Space, F5 — to the index, the subject of the commit at the bottom, COMMIT (КОММИТ, Ctrl+Enter). Files outside the profile of the project do not go into the index — so builds, logs and secrets do not get into the history.

Committed the wrong thing, not sent yet. HISTORY → "take the last off (soft)" (убрать последний (soft)): the commit goes, the files stay in the index; correct and commit again. Or the tick "fix the last one (--amend)" (исправить прошлый (--amend)) in the COMMIT block.

Spoiled a file, want it as in the last commit. CHANGES → "restore" (вернуть, F8). The window asks: the edits of this file are lost for good.

A bad commit is on the server already. HISTORY → choose the commit → revert. A new commit appears that undoes the changes of the chosen one; the history is not rewritten, nobody loses anything.

Lost a commit after a reset. HISTORY → the REFLOG tab, choose the commit, type a branch name at the bottom → "a branch from here" (ветка отсюда).

A merge stopped on a conflict. In CHANGES the files with the letter U: correct them, F5 — to the index, COMMIT. Changed your mind — BRANCHES → "if stuck" (если застряли) → `git merge --abort`, everything returns as it was before the merge.

pull refused: the branches parted. BRANCHES: choose the server's branch (`origin/main`) → "merge into the current one" (влить в текущую), resolve the conflict if there is one, then PUSH. The program takes only by fast-forward on purpose (`pull --ff-only`): you make the merge yourself and see it.

Edits stand in the way of going to another branch. BRANCHES → "put aside" (отложить, stash), switch, work, come back, "take out" (достать).

Release a version. The COMMIT block: the series (`projection`), the window offers the next number by the tags of the series; patch, minor or major; the tick "set a tag" (поставить тег). After the commit — PUSH: `--follow-tags` sends the tag with the branch.

A backup without a server. SERVICE → `git bundle create`: the whole history of a branch and its tags in one file. `git clone ‹file›` brings it back.

A new branch for a task. BRANCHES: the name at the bottom (`feature/report`) → "create and switch" (создать и перейти). The first push is `push -u`: it ties the branch to its pair on the server, after that push and pull need no names.

The commit message

The COMMIT block puts the message together from its parts and shows it whole in a frame:

```
type(scope): series version - subject
docs(readme): projection v0.1.3 - a section on the mirror
```

Type	When
docs	documents, texts
code	code, a new ability
fix	a fix of an error
ui	the look of the window
test	checks
chore	tidying, the build, dependencies
audit	a review and a check

The scope is a part of the project (`readme`, `mirror`, `ui`), it may be empty. The series and the version are optional: with them the message carries a number, and the tick "set a tag" puts the mark `series-version` on this commit. The window offers the next number by the tags of the series; patch — a fix (0.1.1 → 0.1.2), minor — a new ability (0.1 → 0.2), major — an incompatible change (0 → 1).

The Git lab

Ten small repositories in known states. PROJECTS → GIT LAB (УЧЕБНАЯ ЛАБОРАТОРИЯ) → "build anew" (создать заново) or `hub_cli.cmd lab create --replace --register` builds them: the former ones go to the Recycle Bin, the new ones go into the registry (`lab_01` … `lab_10`). Their authors and dates are fixed, so the commits get the same hashes on any machine. Break them freely.

	Repository	What is in it	What to try
01	first-commit	files, no repository yet	`git init`, add, commit
02	index	changed, new, deleted, renamed, a file half in the index	status, three kinds of diff, add, `restore --staged`, restore, a commit of some files
03	branch-merge	a branch `feature/login` behind `main`, `fix/typo` merged already	switch, `switch -c`, `switch -`, compare, `merge --no-ff`, `branch -d`
04	conflict	a merge stopped on a conflict in `style.css`	resolve the conflict or `merge --abort`

	Repository	What is in it	What to try
05	behind-remote	its own "server" in `_servers`; two of your commits not sent, one new on the server	fetch, `pull --ff-only` (refuses on `main`, goes on `docs`), push, `push -u`
06	stash	an edit of `notes.txt` stands in the way of the branch `other`; one stash lies there already	`stash push`, switch, `stash list`, `stash pop`
07	tags	the tags `projection-v0.1.0` and `v0.1.1`, one more commit after them	the next version, `tag -a`, `tag -n`
08	lost-commit	the commit "step two" wiped out by a reset	reflog, a branch from the lost commit
09	revert-reset	the third commit broke a calculation	revert, `reset --soft`, `reset --hard`
10	source-mirror	a project (`source`) and its GIT COPY (`git-copy`) have parted	the plan of the mirror, apply, verify, a commit of the GIT COPY

The tests of the program run every command of the canon on a fresh copy of the lab and check what came of it; the five network ones (`ssh -T`, `gh`, `glab`) by the shape of the command.