ScopeTreasure Courses

GIT & GITHUB COURSES · LESSON G.7 · CODE SCOPE SERIES

Git Undo — Galti Sudharna

Every mistake has an undo: restore a file, unstage, amend the last commit, reset vs revert (and which is safe on shared work), and the stash.

Introduction

Beginners fear Git because they think one wrong command destroys work. The truth is the opposite: Git is the most forgiving tool on your machine — almost every mistake has a named undo. This lesson is the map: **restore** a file you ruined, **unstage** a file you added by accident, **amend** the last commit, and the two big ones — **reset** (rewrite history; private work only) versus **revert** (a new commit that cancels an old one; safe anywhere). Plus the **stash**, your pause button for unfinished work. *(Roman Urdu: Git mein taqreeban har ghalti ka undo mojood hai — bas ye jaanna hai ke kaun sa undo kis stage par lagta hai.)*

Key points

- **Undo depends on the stage:** file edited but not staged → `git restore <file>`. Staged but not committed → `git restore --staged <file>`. Committed but not pushed → amend or reset. Pushed → revert. Same mistake, four different doors.
- **`git restore <file>` is the one destructive undo here:** it throws away your unstaged edits and returns the file to the last commit. What was never committed or stashed is gone — read the file once before you restore it.

  (Roman Urdu: Restore wo edits phenk deta hai jo abhi commit nahi hueen — chalane se pehle aik nazar file par lazmi.)
- **Amend rewrites the LAST commit only:** new message or forgotten file folded in, one clean commit comes out — with a new ID. Golden rule: **never amend (or reset) a commit you have already pushed**; teammates may have built on it.
- **Reset vs revert is the series' most important decision:** `reset` MOVES the branch label backwards and erases commits from history (private clean-up). `revert` adds a NEW commit that does the exact opposite of an old one (public, shared, safe).
- **Stash pauses without committing:** `git stash` shelves all uncommitted changes and hands you a clean tree; `git stash pop` brings them back. Perfect for "urgent fix first, my half-done feature later."

Code examples

Example 1 — Restore a ruined file + unstage an accidental add

```
$ git restore index.html
$ git status
On branch main
nothing to commit, working tree clean
$ git add galat-file.txt index.html
$ git restore --staged galat-file.txt
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        galat-file.txt
```

What it does: you broke `index.html` with a bad experiment — `git restore` returns it to the last committed version, and the tree reports clean. Then you staged two files by habit, but one does not belong in this commit: `git restore --staged` takes ONLY that file back out of the staging area (its content on disk is untouched).

What you will see: the difference between the two restores is everything: plain restore edits the FILE (the bad experiment is gone), `--staged` edits only the LIST (the file and its contents survive, it simply will not be in the next commit).

Example 2 — Reset: erasing a private commit

```
$ git log --oneline
h8i9j0k (HEAD -> main) Tajurba jo fail hua
g7h8i9j Setup: project files aur pehla page
$ git reset --soft HEAD~1
$ git log --oneline
g7h8i9j (HEAD -> main) Setup: project files aur pehla page
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   tajurba.html
```

What it does: `HEAD~1` means "one commit before the current one". `--soft` moves the branch label back one commit and leaves ALL the work staged, ready to recommit differently. Its siblings: `--mixed` (the default) also

unstages, keeping the files; `--hard` deletes the work entirely — powerful, and the one command in this lesson to type slowly.

What you will see: the failed commit vanishes from the log as if it never happened, while its file sits staged — history rewritten, work preserved. This commit was never pushed, so rewriting is honest. Had it been pushed, the correct tool is Example 3.

Example 3 — Revert: cancelling a PUSHED commit, safely

```
$ git revert h8i9j0k
[main i9j0k1l] Revert "Tajurba jo fail hua"
 Date: Wed Jan 13 09:15:00 2027
 1 file changed, 1 deletion(-)
$ git log --oneline
i9j0k1l (HEAD -> main) Revert "Tajurba jo fail hua"
h8i9j0k Tajurba jo fail hua
g7h8i9j Setup: project files aur pehla page
$ git push
To https://github.com/ayesha-khan/notes-app.git
   g7h8i9j..i9j0k1l  main -> main
```

What it does: the bad commit is already on GitHub where a teammate may have pulled it. `git revert` does not delete it — it creates a NEW commit that applies the exact inverse (here, deleting the file the bad commit added). You push the revert like any commit; nobody's history breaks.

What you will see: the log keeps BOTH commits — the mistake and its public correction, one above the other. That is not mess; that is an honest audit trail, and it is why revert is the professional's undo for shared work.

Quick recap

- Match the undo to the stage: unstaged edit → `restore`; staged → `restore --staged`; last commit → `amend`; private commits → `reset`; public commits → `revert`.
- `restore` (files) and `reset --hard` destroy uncommitted work — everything else in this lesson is recoverable.
- Stash is the pause button: shelve unfinished changes, get a clean tree, `git stash pop` to resume.

Common mistakes

1. **Resetting a commit that was already pushed** — your local history and GitHub's now disagree, and the next push is rejected (or a force-push breaks a teammate). The fix: pushed = revert; the reset family is for private history only.
2. **`git restore` on a file whose edits were never saved anywhere** — those lines exist nowhere else, and restore is final. The fix: when in doubt, copy the file aside (or `git stash`) first, restore second.

3. **Reverting a revert / reverting a merge blindly** — undoing an undo re-applies the original change, which confuses everyone including you. The fix: read `git log` top to bottom and say the story aloud before reverting anything older than the last commit.

Practice

1. In a practice repo: commit a good file, then ruin it with bad edits. Restore it and verify the good version is back. Then stage a second file by "accident", unstage it with `--staged`, and confirm with `git status` that it left the commit list but not the disk.
2. Make three commits A, B, C. Undo C with `git reset --soft HEAD~1` and recommit its work under a better message. Then make a commit D, "push" it (to GitHub if your repo is connected, or just mark it public), and cancel it with `git revert`. Read the final log and explain — in one line each — why C got reset but D got reverted.

Practice quiz

1. A file is edited but not staged. Which undo throws the edits away?

- git restore --staged <file>
- git restore <file>, back to the last commit, and what was never committed or stashed is gone
- git revert
- git stash pop

Answer: git restore <file>, back to the last commit, and what was never committed or stashed is gone

2. What is the difference between reset and revert?

- They are the same
- reset moves the branch label backwards and erases commits from history (private clean-up); revert adds a new commit that does the exact opposite of an old one (public, shared, safe)
- reset is for public history, revert is for private history
- reset never changes history

Answer: reset moves the branch label backwards and erases commits from history (private clean-up); revert adds a new commit that does the exact opposite of an old one (public, shared, safe)

3. You pushed a commit and now regret it. What is the safe undo?

- Amend it and force-push
- Reset it locally, teammates will catch up
- Revert it, because pushed history is shared; never amend or reset a commit you have already pushed
- Delete the .git folder

Answer: Revert it, because pushed history is shared; never amend or reset a commit you have already pushed

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/git-undo · More free tools and notes at https://scopetreasure.com