ScopeTreasure Courses

GIT & GITHUB COURSES · LESSON G.8 · CODE SCOPE SERIES

Git Stash + Cherry-Pick — Kaam Ko Rok Kar Sambhalna

Park half-finished work safely with stash (list, apply, pop, drop) and lift one single commit onto another branch with cherry-pick.

Introduction

Two situations every developer meets. First: you are halfway through a feature when an urgent bug arrives — your unfinished work is everywhere, but it is not ready to commit. Second: a teammate fixed a bug on THEIR branch, and you need exactly that one commit — not their whole half-finished feature. Git has a precise tool for each: **stash** parks your uncommitted changes in a drawer and hands you a clean desk, and **cherry-pick** copies one chosen commit onto your branch. *(Roman Urdu: Stash adhoora kaam drawer mein rakh deta hai; cherry-pick bherri mein se sirf aik commit utha kar aap ki branch par la deta hai.)*

Key points

- **Stash stores UNcommitted changes:** tracked files you modified (and staged work) are shelved; the working tree returns to the last commit. Untracked brand-new files stay put unless you add `-u`.
- **The stash is a STACK:** each `git stash` pushes a new entry `stash@{0}` on top; older ones slide down (`stash@{1}`, `stash@{2}`...). `git stash list` shows the pile, newest first.
  (Roman Urdu: Stash aik stack hai — sab se naya kaam sab se upar, stash@{0}.)
- **`pop` vs `apply`:** both bring a stash back. `apply` keeps the entry in the list afterwards (a spare copy); `pop` applies AND removes it. Default target is `stash@{0}`; name another to pick an older one.
- **Cherry-pick copies a commit, not the branch:** `git cherry-pick <id>` replays that commit's changes onto your current branch as a NEW commit (new ID, same message). It is how a hotfix travels from `main` to a release branch without dragging everything else along.
- **Both tools can conflict:** a stash pop or a cherry-pick applies changes onto possibly-changed code — conflicts are resolved exactly like merge conflicts (G.3): edit, remove markers, add, commit.

Code examples

Example 1 — Stash: park the feature, fix the urgent bug, resume

```
$ git status
On branch feature-new-homepage
Changes not staged for commit:
          modified:   index.html
          modified:   style.css
$ git stash push -m "Homepage ka adhoora kaam"
Saved working directory and index state On feature-new-homepage: Homepage ka adhoora kaam
$ git status
On branch feature-new-homepage
nothing to commit, working tree clean
$ git switch main
$ git commit -am "Urgent fix: footer ka link theek kiya"
[main j0k1l2m] Urgent fix: footer ka link theek kiya
 1 file changed, 1 insertion(+), 1 deletion(-)
```

What it does: your homepage is half-changed in two files — uncommittable, but too valuable to delete. `git stash push -m` shelves the changes under your message and the desk is instantly clean, so you can switch branches and fix the urgent bug on main like a professional, not a juggler.

What you will see: the moment the stash lands, the modified files return to their last-committed state — it looks as if the afternoon's work vanished. It did not: it is in the drawer, one command away (Example 2). That empty-looking tree is precisely the point.

Example 2 — Bringing the work back: list, pop, verify

```
$ git switch feature-new-homepage
$ git stash list
stash@{0}: On feature-new-homepage: Homepage ka adhoora kaam
$ git stash pop
On branch feature-new-homepage
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
          modified:   index.html
          modified:   style.css

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (k1l2m3n)
$ git stash list
```

What it does: `git stash list` shows the drawer and its labels — your message makes the right entry findable weeks later. `git stash pop` restores both modified files AND removes the entry (see "Dropped

refs/stash@{0}"). A final `git stash list` prints nothing: drawer empty, work resumed exactly where you left it.

What you will see: if the urgent fix had touched the SAME lines you were editing, the pop would stop with a conflict — Git keeps the stash entry in that case (nothing is dropped until the apply succeeds), so resolve the markers, and the entry can be dropped by hand with `git stash drop`.

Example 3 — Cherry-pick: one bug fix, moved alone

```
$ git log --oneline --all
l2m3n4o (main) Urgent fix: footer ka link theek kiya
m3n4o5p (feature-new-homepage) Header ka naya design
$ git switch feature-new-homepage
$ git cherry-pick l2m3n4o
[feature-new-homepage n4o5p6q] Urgent fix: footer ka link theek kiya
 Date: Wed Jan 13 11:02:00 2027
 1 file changed, 1 insertion(+), 1 deletion(-)
```

What it does: the footer fix lives on `main` (commit `l2m3n4o`), and your feature branch needs it too — but NOT the merge of everything else on main. Standing on the feature branch, `git cherry-pick l2m3n4o` replays only that commit's change here, as the brand-new commit `n4o5p6q` with the same message.

What you will see: the fix appears in your branch's log with a DIFFERENT ID — same change, same message, new home. That new ID is the signature of cherry-pick: the commit was copied, not moved.

Quick recap

- Stash = drawer for uncommitted work: `stash push -m` to park, `list` to see the stack, `pop` to resume-and-clear, `apply` to resume-and-keep.
- Cherry-pick = carry ONE commit onto another branch as a new commit with a new ID. Built for hotfixes that must travel alone.
- Both can conflict; both resolve like G.3 merges. A stash only drops after a clean pop — your work is safer than it looks.

Common mistakes

1. **Stashing and forgetting for weeks** — the drawer fills with mystery entries and nobody dares touch them. The fix: stash is a pause button, not storage — always `push -m` with a real message, and pop the same day if you can.
2. **Expecting untracked files in the stash** — brand-new files are NOT stashed by default, so the "clean" tree still shows them and build scripts may pick them up. The fix: `git stash push -u` when new files must go into the drawer too.
3. **Cherry-picking a whole branch one commit at a time, in the wrong order** — ten picks later, the history is a braid. The fix: if you need most of the branch, merge it (G.3); cherry-pick is for ONE or two surgical

commits, oldest first.

Practice

1. In a practice repo, modify two tracked files, stash them with the message "practice pause", and verify the tree is clean. List the stash, then pop it and confirm both modifications returned and the list is empty.
2. Create branch `release` from an older commit. On `main`, make one small fix commit. Switch to `release`, cherry-pick ONLY that fix, and compare `git log --oneline` on both branches — find the fix twice, with two different IDs.

Practice quiz

1. What is the stash, and what is stash@{0}?

- A branch for old work
- A stack of shelved uncommitted changes; each git stash pushes a new entry on top, so stash@{0} is the newest
- A backup on GitHub
- A commit with a special message

Answer: A stack of shelved uncommitted changes; each git stash pushes a new entry on top, so stash@{0} is the newest

2. What is the difference between git stash pop and git stash apply?

- There is none
- Both bring a stash back, but apply keeps the entry in the list afterwards as a spare copy, while pop applies AND removes it
- pop keeps it, apply removes it
- apply only works for the oldest stash

Answer: Both bring a stash back, but apply keeps the entry in the list afterwards as a spare copy, while pop applies AND removes it

3. What does git cherry-pick <id> do?

- It moves the whole branch
- It replays that one commit's changes onto your current branch as a new commit with a new ID and the same message, so a hotfix can travel alone
- It deletes the commit
- It merges two branches

Answer: It replays that one commit's changes onto your current branch as a new commit with a new ID and the same message, so a hotfix can travel alone

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-stash-cherry-pick · More free tools and notes at https://scopetreasure.com