Stash

Saving uncommitted changes aside with git stash, the difference between apply and pop, multiple and named stashes, git stash drop/clear, and stashing untracked files. A realistic "interrupted mid-feature, need to switch tasks" scenario.

Intermediate 25 min

Last updated: August 27, 2026
Read online: https://www.learnforgex.com/en/topics/stash

Here's a situation every developer runs into: you're in the middle of a feature, your code doesn't compile yet, and suddenly you need to switch branches — a production bug just came in, or a teammate needs a quick review on a different branch. You're not ready to commit. `git stash` is built exactly for this.

What Is git stash?

`git stash` takes your uncommitted changes — both staged and unstaged — and saves them aside in a separate, temporary storage area, restoring your working tree to match the last commit. It's essentially "commit, but not really, and not on any branch" — a way to get back to a clean working tree without actually finishing what you were doing.

Stashing Changes

```
git stash
```

That's it — every tracked file's uncommitted changes are saved, and your working tree goes back to matching `HEAD`. You can now switch branches freely, since there's nothing uncommitted left to conflict with the switch.

git stash list

Stashes stack up — you can have more than one at a time. See what's saved:

```
$ git stash list
stash@{0}: WIP on add-password-reset: 8a1b2c3 Add PasswordController
stash@{1}: WIP on main: a1b2c3d Fix typo in README
```

Each entry shows an index (`stash@{0}` is the most recent), the branch it was stashed from, and the commit it was based on.

git stash apply

Bring the most recent stash's changes back into your working tree, **without removing it from the stash list**:

```
git stash apply
```

Use `apply` when you might want to apply the same stash again later, or onto a different branch — it's non-destructive.

git stash pop

The more common everyday command — apply the most recent stash **and remove it from the list** in one step:

```
git stash pop
```

Once you're done with a stash's contents, `pop` is almost always what you actually want — it cleans up after itself.

Applying a Specific Stash

If you have more than one stash saved, reference any of them by index:

```
git stash apply stash@{1}
git stash pop stash@{1}
```

Naming Stashes

With several stashes stacked up, `WIP on <branch>: <hash> <message>` can get hard to tell apart. Give a stash a memorable description when you create it:

```
git stash push -m "half-finished password strength validator"
```

(`git stash push` is the more explicit, modern form of plain `git stash` — both do the same thing, but `push` accepts a message and a few other options.)

git stash drop

Delete a specific stash without applying it — useful once you're certain you don't need those changes anymore:

```
git stash drop stash@{0}
```

git stash clear

Delete every stash at once:

```
git stash clear
```

> Both `drop` and `clear` permanently discard the stashed changes — there's no confirmation prompt. Make sure you're certain before running either.

Stashing Untracked Files

By default, `git stash` only stashes changes to files Git is already tracking — brand-new, untracked files are left alone. If you also want to stash new files:

```
git stash -u          # include untracked files
git stash --all       # include untracked AND ignored files
```

A Real-World Stash Workflow

```
#!/bin/sh
# A developer is mid-way through a Java feature (code doesn't even compile yet)
# when an urgent production bug needs attention on a different branch.

git status
# Output:
# On branch add-password-reset
# Changes not staged for commit:
#         modified:   PasswordValidator.java
#         modified:   PasswordController.java

# Not ready to commit -- the code doesn't compile yet. Stash it aside.
git stash push -m "half-finished password strength validator"
# Output:
# Saved working directory and index state On add-password-reset: half-finished password
strength validator

git status
# Output:
# On branch add-password-reset
# nothing to commit, working tree clean

# Switch to main and fix the urgent bug.
git switch main
git switch -c fix-login-crash
# ... fix the bug ...
git commit -am "Fix NullPointerException on login when session expired"
git switch main
git merge fix-login-crash
git branch -d fix-login-crash

# Back to the feature -- restore exactly where we left off.
git switch add-password-reset
git stash list
# Output:
# stash@{0}: On add-password-reset: half-finished password strength validator

git stash pop
# Output:
# On branch add-password-reset
# Changes not staged for commit:
#         modified:   PasswordValidator.java
#         modified:   PasswordController.java
# Dropped stash@{0} (a1b2c3d4e5f6...)

# Exactly the same uncommitted state as before the interruption.
```

Common Mistakes

- **Forgetting a stash exists.** It's easy to `git stash` and move on to something else entirely, only to discover a stack of forgotten stashes weeks later — `git stash list` regularly is a good habit.
- **Using `apply` instead of `pop` out of habit**, then wondering why the stash list keeps growing — if you don't specifically need to reuse the stash, `pop` is usually right.
- **Assuming untracked files are stashed by default** — they aren't, unless you pass `-u` or `--all`.

Best Practices

- Give stashes a message (`git stash push -m "..."`) the moment you have more than one active — future-you will thank you.
- Prefer `pop` for the common case; reach for `apply` specifically when you know you'll want the same stash again.
- Treat the stash as genuinely temporary — if work sits stashed for more than a day or two, it's usually a sign it should have been a proper commit (or a `git commit --amend`) on its own small branch instead.

Summary, Cheat Sheet, and Glossary

Summary

- `git stash` saves uncommitted changes aside and restores a clean working tree, letting you switch tasks without committing unfinished work.
- `apply` restores a stash's changes and keeps it in the list; `pop` restores and removes it — `pop` is the common everyday choice.
- Stashes stack (`stash@{0}`, `stash@{1}`, ...) and can be named, listed, applied by index, dropped individually, or cleared entirely.
- Untracked files are excluded from a stash by default — use `-u` to include them.

Cheat Sheet

```
git stash                       # save uncommitted changes aside
git stash -u                    # also include untracked files
git stash list                  # see all saved stashes
git stash pop                   # restore the most recent stash, and remove it
git stash apply stash@{1}       # restore a specific stash, keep it in the list
git stash drop stash@{0}        # delete a specific stash
git stash clear                 # delete every stash
```

Glossary

- **Stash** — a temporary, off-branch save of uncommitted changes.
- `stash@{N}` — the index used to reference a specific saved stash (0 = most recent).