Git

You know Git. This isn't a tutorial — it's the reference for the commands you reach for under pressure and the recovery moves that turn "I just destroyed three hours of work" into "nothing happened." The headline truth: **almost nothing in Git is actually gone.** `reflog` is your time machine. Learn it and you stop fearing the tool.

Daily driver

```
git status -sb                    # short, with branch info
git add -p                        # stage hunks interactively — review what you commit
git commit -m "msg"
git commit --amend                # fix the last commit (message or staged content)
git commit --amend --no-edit      # tack staged changes onto last commit, keep message
git pull --rebase                 # avoid the noise merge commits on pull
git push
git push -u origin feature/x      # set upstream first push
git log --oneline --graph --all --decorate    # the only log you need
```

Branching

```
git switch -c feature/x           # create + switch (modern; replaces checkout -b)
git switch main                   # switch (replaces checkout <branch>)
git branch -d feature/x           # delete merged branch
git branch -D feature/x           # force-delete unmerged
git branch -vv                    # show tracking + last commit per branch
git fetch -p                      # fetch + prune dead remote branches
```

Stash

```
git stash push -m "wip auth"      # named stash
git stash list
git stash show -p stash@{0}       # diff a stash
git stash pop                     # apply + drop
git stash apply stash@{1}         # apply, keep in list
git stash drop stash@{0}
git stash -u                      # include untracked files
```

Rebase & history surgery

```
git rebase main                  # replay current branch onto main
git rebase -i HEAD~5             # squash/reword/reorder last 5 commits
git rebase --abort               # bail out of a bad rebase
git rebase --continue            # after resolving conflicts
git cherry-pick <sha>            # grab one commit onto current branch
git revert <sha>                 # safe undo: new commit that reverses <sha>
```

The "oh no" recovery section

```
git reflog                       # EVERY HEAD movement — your undo history
git reset --hard HEAD@{2}        # jump back to where HEAD was 2 moves ago

# undo last commit, KEEP the changes staged
git reset --soft HEAD~1
# undo last commit, KEEP changes unstaged (working dir)
git reset --mixed HEAD~1
# nuke last commit AND its changes (careful)
git reset --hard HEAD~1

# recover a "deleted" branch
git reflog | grep feature/x      # find the sha
git switch -c feature/x <sha>

# recover a single file to a previous state
git restore --source=HEAD~3 path/to/file
git checkout <sha> -- path/to/file    # older syntax, still works

# accidentally committed to main instead of a branch
git switch -c feature/x          # branch now points at your commits
git switch main
git reset --hard origin/main     # reset main back to clean

# threw away uncommitted work with reset --hard? if it was ever staged:
git fsck --lost-found            # dangling blobs live here
```

Undo a pushed commit (when you must)

```
# safe (public branch): revert creates a new undo commit
git revert <sha> && git push

# rewriting pushed history (only on branches you own, never shared main)
git reset --hard <good-sha>
git push --force-with-lease       # NOT --force — lease checks nobody else pushed
```

Useful config / aliases

```
git config --global pull.rebase true
git config --global init.defaultBranch main
git config --global alias.lg "log --oneline --graph --all --decorate"
git config --global alias.st "status -sb"
git config --global rerere.enabled true   # remember conflict resolutions
```

Things that bite you

- `--force` **vs** `--force-with-lease`**:** always use the lease version. Plain `--force` will happily overwrite a teammate's pushed work without warning.
- `reset --hard` **discards uncommitted changes permanently** — there's no reflog for unstaged working-dir edits. Stash first if unsure.
- **Detached HEAD** isn't broken — you're just not on a branch. Commits made there get garbage-collected unless you `switch -c` to keep them. `reflog` finds them if you wandered off.
- `git pull` **without** `--rebase` spawns merge commits that clutter history. Set `pull.rebase true` globally.
- **Line-ending wars (CRLF/LF)** across Windows/Mac/Linux — set `core.autocrlf` consistently per platform or use a `.gitattributes` with `* text=auto`.