GIT REFERENCE

Git Commands for QA Engineers

50+ commands across 11 categories. No fluff. Just the ones you actually use.

BASICS

`git clone <url>`
Download a repository

`git status`
See what changed

`git add .`
Stage all changes

`git commit -m "msg"`
Save with a message

`git push`
Upload to remote

`git pull`
Download latest changes

`git init`
Initialize a new local repository

`git remote -v`
Show remote URLs

BRANCHING

`git checkout -b <name>`
Create + switch branch

`git checkout <branch>`
Switch to branch

`git branch`
List all local branches

`git branch -d <name>`
Delete merged branch

`git merge <branch>`
Merge into current branch

`git branch -a`
List all incl. remote branches

`git switch -c <name>`
Modern way to create + switch branch

`git branch -D <name>`
Force delete unmerged branch

SYNCING

`git fetch`
Download without merging

`git pull origin main`
Pull latest from main

`git push -u origin <br>`
Push + set upstream tracking

`git pull --rebase`
Rebase on top of latest

`git fetch --prune`
Remove stale remote-tracking branches

`git push origin --delete <br>`
Delete a remote branch

UNDO MISTAKES

`git restore <file>`
Discard file changes in working dir

`git reset HEAD <file>`
Unstage a file

`git reset --soft HEAD~1`
Undo commit, keep changes staged

`git revert <commit>`
Undo with a new commit (safe)

`git commit --amend`
Fix last commit message or content

`git clean -fd`
Remove all untracked files and dirs

`git reset --hard HEAD~1`
Undo commit AND discard all changes

`git restore --staged <file>`
Unstage a file (modern syntax)

⚠️ **Warning:** `git reset --hard` permanently discards changes. Never use on shared branches. Use `git revert` instead for pushed commits.

STASHING

`git stash`
Save changes temporarily

`git stash pop`
Restore + delete most recent stash

`git stash list`
Show all saved stashes

`git stash drop`
Delete the most recent stash

`git stash apply stash@{2}`
Apply specific stash without removing

`git stash -u`
Stash including untracked files

PR WORKFLOW

`git checkout -b feature/tests`
Create feature branch for test work

`git push -u origin feature/tests`
Push branch to remote for PR

`git log --oneline`
Compact commit history

`git diff`
Show unstaged changes

`git diff --staged`
Show staged changes before commit

`git cherry-pick <hash>`
Apply a specific commit to current branch

LOG & HISTORY

`git log --oneline --graph`
Visual branch history in terminal

`git log --author="name"`
Filter commits by author

`git log --since="2 weeks ago"`
Commits from last 2 weeks

`git show <hash>`
Show details of a specific commit

`git blame <file>`
See who changed each line of a file

`git shortlog -sn`
Commit count per author

TAGS & RELEASES

git tag
List all tags

git tag v1.0.0
Create a lightweight tag

git tag -a v1.0.0 -m "msg"
Create an annotated tag with message

git push origin v1.0.0
Push a specific tag to remote

git push origin --tags
Push all tags to remote

git tag -d v1.0.0
Delete a local tag

CONFIG & SETUP

git config --global user.name
Set your global username

git config --global user.email
Set your global email

git config --list
View all config settings

git config --global alias.st status
Create alias — git st = git status

.gitignore
File listing paths Git should ignore

git config core.autocrlf true
Fix line endings on Windows

ADVANCED

git rebase <branch>
Rebase current branch onto another

git rebase -i HEAD~3
Interactive rebase — squash last 3 commits

git bisect start
Start binary search for a bad commit

git bisect good / bad
Mark commit as good or bad in bisect

git reflog
View history of HEAD — recover lost commits

git submodule update --init
Initialize and update submodules

git worktree add <path>
Check out branch in a separate directory

git archive --format=zip HEAD
Export current HEAD as a zip file

git bisect is a QA superpower — it uses binary search to find exactly which commit introduced a bug. Run `git bisect start`, mark a known bad commit and a known good commit, and Git will guide you to the culprit automatically.

CI/CD & AUTOMATION

git log --format="%H" -1
Get latest commit hash for pipeline tracking

git describe --tags
Get closest tag — useful for release versioning

git clone --depth 1 <url>
Shallow clone — faster in CI pipelines

git fetch origin <branch>
Fetch a specific branch in CI without full clone

git rev-parse HEAD
Print full SHA of current commit

GIT_SSH_COMMAND="ssh -i key"
Use custom SSH key in pipeline

QA-SPECIFIC GIT WORKFLOWS

🐛 Bug Fix Workflow

```
01 · git checkout main && git pull
02 · git checkout -b fix/login-bug-123
03 · ... make fix + write test ...
04 · git add . && git commit -m "fix: login bug #123"
05 · git push -u origin fix/login-bug-123
06 · → Open Pull Request on GitHub/GitLab
```

🔄 Sync with Main (Before Testing)

```
01 · git checkout feature/my-tests
02 · git fetch origin
03 · git rebase origin/main ← cleaner than merge
04 · git push --force-with-lease ← safer force push
```

🔍 Finding a Bug with Bisect

```
01 · git bisect start
02 · git bisect bad ← current commit is broken
03 · git bisect good v2.0.0 ← this tag was fine
04 · Git checks out the middle commit automatically
05 · Test → run git bisect good or git bisect bad
06 · git bisect reset ← when done, go back to HEAD
```

QUICK REFERENCE CHEAT SHEET

I WANT TO...	COMMAND
Undo last commit but keep my changes	`git reset --soft HEAD~1`
See what I changed before staging	`git diff`
See what is staged	`git diff --staged`
Save work without committing	`git stash`
Find which commit broke something	`git bisect start`
See who changed a specific line	`git blame <file>`
Recover a deleted branch / lost commit	`git reflog`
Apply one commit from another branch	`git cherry-pick <hash>`
Squash last 3 commits into one	`git rebase -i HEAD~3`
Speed up clone in CI pipeline	`git clone --depth 1 <url>`