DevOps — Git (25 Questions)

Q1: A feature branch shows hundreds of unrelated changes in `git diff` after merging `main`.

Answer:
This can happen if the branch was created from an outdated base or if a merge was done with incorrect ancestry. Run git fetch --all, then rebase onto the latest main to replay only your commits. For large drift, create a new branch from main and cherry-pick your changes.

Sample Points:

- Outdated base = merge noise.
- Rebase to clean commit history.
- Cherry-pick for precise isolation.

Example Code:

```
git checkout feature
git fetch origin
git rebase origin/main
```

Q2: You accidentally committed secrets to Git and pushed to remote.

Answer:
Use `git filter-repo` or `BFG Repo-Cleaner` to remove them from history, force-push to remote, and rotate credentials immediately.

Sample Points:

- History rewrite required.
- Rotate secrets even if removed.
- Inform collaborators to re-clone.

Example Code:

```
git filter-repo --path secret.txt --invert-paths
git push origin --force
```

Q3: Merge conflicts keep recurring in the same file across multiple sprints.

Answer:
This is usually due to parallel changes in high-churn files. Apply a consistent merge strategy (ours, theirs), or split config into smaller files. Encourage feature toggles to reduce long-lived branches.

Sample Points:

- High-churn files → constant conflicts.
- Modularize files to isolate changes.
- Short-lived branches minimize pain.

Example Code:

```
git merge -X theirs feature
```

Q4: `git pull` shows "unrelated histories" after repo migration.

Answer:

This happens when histories differ. Use `git pull --allow-unrelated-histories` during the first merge, then align branches.

Sample Points:

- Repo migration resets ancestry.
- Merge histories once, then clean.
- Use tags to mark migration point.

Example Code:

```
git pull origin main --allow-unrelated-histories
```

Q5: You need to squash all commits in a feature branch before merging.

Answer:

Use `git rebase -i` to squash commits, or `git merge --squash` when merging into target.

Sample Points:

- Squash for cleaner history.
- Rebase interactive for fine control.
- Squash merge preserves branch diff only.

Example Code:

```
git rebase -i HEAD~5
```

Q6: You cloned a large repo but only need a single folder.

Answer:

Use sparse checkout to pull only that folder.

Sample Points:

- Sparse checkout saves bandwidth.
- Useful for monorepos.
- Avoid full clone if unnecessary.

Example Code:

```
git sparse-checkout init --cone
git sparse-checkout set folder/path
```

Q7: A teammate force-pushed `main` and you have diverging histories.

Answer:

Fetch latest, back up your branch, then hard reset to remote main. Reapply your changes on

top.

Sample Points:

- Back up before reset.
- Divergence due to force push.
- Use reflog if needed.

Example Code:

```
git fetch origin
git checkout main
git reset --hard origin/main
```

Q8: Need to sign all commits with GPG for compliance.

Answer:
Generate GPG key, add to Git config, and enforce signed commits in repo settings.

Sample Points:

- GPG key for identity assurance.
- Repo policy enforces signing.
- Protects against commit spoofing.

Example Code:

```
git config --global user.signingkey <KEY_ID>
git commit -S -m "Signed commit"
```

Q9: Large binary files cause repo size to explode.

Answer:
Use Git LFS for binaries, migrate existing ones with `git lfs migrate`.

Sample Points:

- LFS stores binaries outside Git history.

- Keeps repo lightweight.
- Requires LFS installed on clients.
  Example Code:

```
git lfs install
git lfs track "*.zip"
```

Q10: Need to tag a release and push it to remote.

Answer:
Create annotated tags for release metadata. Push explicitly to remote.

Sample Points:

- Annotated tags store message/author.
- Push tags explicitly.
- Use semantic versioning.
  Example Code:

```
git tag -a v1.2.0 -m "Release 1.2.0"
git push origin v1.2.0
```

Q11: You need to revert a single commit in the middle of history without removing others.

Answer:
Use `git revert` to create a new commit undoing changes.

Sample Points:

- Revert is safe on shared branches.
- Doesn't rewrite history.
- Reapply later if needed.
  Example Code:

```
git revert <commit_hash>
```

Q12: CI pipeline triggers on every commit to any branch, but you only want it on main and dev.

Answer:

Restrict pipeline triggers in `.gitlab-ci.yml` or GitHub Actions workflows with branch filters.

Sample Points:

- Branch filters reduce wasted CI runs.
- Config in pipeline definition.
- Improves build efficiency.

Example Code:

```
on:
  push:
    branches:
      - main
      - dev
```

Q13: Accidentally committed large log files you don't want in repo at all.

Answer:

Remove from index, add to `.gitignore`, and commit removal. For complete removal, rewrite history.

Sample Points:

- Remove and ignore to prevent recurrence.
- History rewrite for permanent removal.
- Consider LFS for large assets.

Example Code:

```
git rm --cached big.log
echo "big.log" >> .gitignore
```

Q14: Need to apply a fix from `main` to an older release branch.

Answer:

Use `git cherry-pick` for the specific commit.

Sample Points:

- Cherry-pick applies single commit.
- Resolve conflicts manually if needed.
- Preserve commit metadata.

Example Code:

```
git checkout release-1.0
git cherry-pick <commit_hash>
```

Q15: Git merge shows binary files as conflicts with no merge tool option.

Answer:

Binary files can't be merged automatically. Decide which version to keep using `git checkout --ours` or `--theirs`.

Sample Points:

- Binary conflicts require manual choice.
- Ours/theirs for conflict resolution.
- Commit after resolving.

Example Code:

```
git checkout --ours path/file.bin
git add path/file.bin
```

Q16: You need to apply the same commit to multiple repos.

Answer:

Create a patch with `git format-patch` and apply with `git am`.

Sample Points:

- Format-patch preserves metadata.
- Works across unrelated repos.
- Keep patch files in secure storage.

Example Code:

```
git format-patch -1 <commit>
git am < patchfile
```

Q17: Developer needs read-only access to repo.

Answer:

Grant Read role in hosting platform or create a deploy key with read permissions only.

Sample Points:

- Use platform RBAC.
- Deploy keys for automation.
- Never give write if not required.

Example Code:

```
ssh-keygen -t rsa -b 4096 -C "read-only"
```

Q18: A submodule commit is missing in remote after cloning.

Answer:

Ensure you clone with `--recursive` or run `git submodule update --init`

`--recursive`.

SamplePoints:

- Submodules need explicit init.
- Recursive flag fetches nested submodules.
- Keep submodules synced.

Example Code:

```
git clone --recursive repo.git
```

Q19: Need to rename a branch both locally and remotely.

Answer:
Rename locally, push to remote, and delete old remote branch.

Sample Points:

- Push new branch, delete old one.
- Update tracking branch config.
- Inform collaborators.

Example Code:

```
git branch -m old new
git push origin new
git push origin --delete old
```

Q20: Git bisect required to find commit that introduced a bug.

Answer:
Mark known good and bad commits, test each step.

Sample Points:

- Bisect binary search speeds debugging.

- Requires reproducible test.
- Automate with `gitbisect run`.

Example Code:

```
git bisect start
git bisect bad HEAD
git bisect good v1.1
```

Q21: CI build fails because `.gitmodules` points to wrong URL.

Answer:
Update `.gitmodules`, run `git submodule sync`, and commit changes.
Sample Points:

- Sync updates local config.
- Commit .gitmodules change.
- Ensure URL works for CI.

Example Code:

```
git submodule sync --recursive
```

Q22: Need to temporarily stash changes for a quick hotfix.

Answer:
Use `git stash` to save uncommitted work, apply hotfix, then `git stash pop`.
Sample Points:

- Stash to free working dir quickly.
- Pop restores + removes from stash.
- Multiple stashes possible.

Example Code:

```
git  stash  #
hotfix   work
git stash pop
```

Q23: Commit history is cluttered with merge commits from frequent syncs.

Answer:

Use `git pull --rebase` to replay commits on top of fetched branch.

Sample Points:

- Rebasing avoids merge commits.
- Cleaner history for review.
- Conflicts resolved as usual.

Example Code:

```
git pull --rebase origin main
```

Q24: You need to change the author of the last commit.

Answer:

Use `git commit --amend --author`. Force-push if already pushed.

Sample Points:

- Amend for local fix.
- Force push to update remote.
- Inform collaborators.

Example Code:

```
git commit --amend --author="New Name <email>"
```

Q25: Repo size is huge due to unused branches.

Answer:

Prune stale remote branches with `git fetch --prune`, delete local copies.

Sample Points:

- Prune removes tracking refs.
- Delete locals to free space.
- Review before deleting.

Example Code:

```
git fetch --prune
git branch -d old-branch
```