**1. What is a Git Branch?**

A branch in Git is like a separate “timeline” of your code.

* You can experiment in one branch without affecting the main code.
* The main (or master) branch usually holds the stable production code.
* You can create branches for features, bug fixes, or experiments.

**2. Check Your Current Branch**

git branch

* Shows a list of branches.
* The one with \* is your current branch.

**3. Create a New Branch**

git branch feature/login

* Creates a branch named feature/login.
* Still, you are **not** switched to it yet.

**4. Switch to a Branch**

git checkout feature/login

or (modern way):

git switch feature/login

**5. Create and Switch in One Command**

git checkout -b feature/login

or:

git switch -c feature/login

**6. Rename a Branch**

git branch -m old-name new-name

**7. Delete a Branch**

# Delete locally

git branch -d feature/login # Safe delete (will prevent deletion if unmerged)

git branch -D feature/login # Force delete

# Delete remotely

git push origin --delete feature/login

**8. Push a New Branch to Remote**

git push -u origin feature/login

-u sets upstream tracking, so next time you can just git push.

**9. Merge a Branch into Main**

# Switch to main branch

git checkout main

# Merge

git merge feature/login

If no conflicts → Git will merge automatically.

**10. Handling Merge Conflicts**

When Git says:

CONFLICT (content): Merge conflict in file.txt

You must:

1. Open the file → resolve conflicts manually.
2. Mark as resolved:

git add file.txt

git commit

**11. Rebase (Alternative to Merge)**

Rebasing applies your commits **on top of** another branch’s history:

git checkout feature/login

git rebase main

* Keeps a cleaner history (no merge commits).
* But can be dangerous if used on shared branches.

**12. View Branch History**

git log --oneline --graph --all

This gives a nice visual of branches.

**13. Fetching & Pulling Remote Branches**

# See all remote branches

git branch -r

# Checkout a remote branch locally

git checkout -b feature/login origin/feature/login

**14. Branch Naming Conventions (Best Practice)**

* main → Production-ready code
* develop → Staging / integration branch
* feature/\* → New features (feature/payment)
* bugfix/\* → Bug fixes
* hotfix/\* → Quick fixes for production
* release/\* → Pre-release preparation

**15. Common Branch Workflows**

**Git Flow**

main ← release ← develop ← feature branches

Used for complex projects with multiple releases.

**GitHub Flow**

main ← feature branches → PR → merge into main

Used for simpler continuous deployment setups.

**16. Stash Work Before Switching Branch**

If you have uncommitted changes but need to switch branches:

git stash

git checkout other-branch

git stash pop

**17. Bonus – Clean Up Old Branches**

# List merged branches

git branch --merged

# Delete merged branches

git branch --merged | grep -v "\\*" | grep -v main | xargs git branch -d

✅ **Quick Command Reference**

| **Action** | **Command** |
| --- | --- |
| List branches | git branch |
| Create branch | git branch new-branch |
| Switch branch | git checkout branch or git switch branch |
| Create & switch | git checkout -b branch |
| Rename branch | git branch -m old new |
| Delete branch (local) | git branch -d branch |
| Delete branch (remote) | git push origin --delete branch |
| Push branch | git push -u origin branch |
| Merge branch | git merge branch |
| Rebase branch | git rebase branch |