📚 What is a Git Branch

> *A branch is a separate line of development that isolates changes from the main codebase.*

- Think of a branch as a **sandbox** where new code can be written, tested, and refined without affecting the stable version.
- Once the work is verified, the branch can be **merged** back and optionally deleted.

🔀 Branching Strategy Overview

> *A branching strategy defines how a team creates, uses, and merges branches to deliver software reliably and on schedule.*

- **Why it matters:**
  - Guarantees **regular releases** (e.g., every 2–3 months).
  - Keeps the **main line stable** while multiple developers work concurrently.
  - Frequently appears in **DevOps interview questions** (top 20/50 lists).
- **Core idea:**
  - Use **feature branches** for new work, **release branches** for preparing shipments, and keep the **main/master branch** for ongoing development.

🌳 Types of Branches

Branch Type	Purpose	Typical Lifecycle	Example Use-Case
main / master	Stable baseline for the product	Long-living, always present	Production-ready code

Feature branch (feature/*)	Isolate development of a new feature or large change	Created from main, merged back when finished, then deleted	Adding *percentage* operation to a calculator
Release branch (release/*)	Stabilize code for an upcoming release	Forked from main when a set of features is complete; only bug fixes & final tweaks are applied	Preparing Kubernetes v1.30 release
Hotfix branch (hotfix/*)	Emergency fixes on production	Forked from main (or from a release tag), merged back to both main and the active release branch	Critical security patch for Uber's app

1️⃣ Main / Master Branch

- Holds the **latest stable** version of the application.
- Continuous integration pipelines usually run **unit tests** on every push.

2️⃣ Feature Branches

- Created when developers need to **add or modify** functionality.
- Allows multiple teams to work in parallel without stepping on each other's toes.
- Example:
  - Original calculator supports +, -, ×, ÷.
  - A new **V2** branch (feature/percentage) adds % and advanced operations.

3️⃣ Release Branches

- Spawned when the team decides the next **release candidate** is ready for final testing.
- Isolated from ongoing development on main to avoid accidental changes.
- After successful testing, the release branch is **merged** back into main (and often tagged).

4️⃣ Hotfix Branches ***(often discussed alongside branching strategies)***

- Used for **urgent patches** on live systems.
- Quickly merged into both main and the current release branch to keep all lines consistent.

🛠️ Practical Example: Kubernetes Repository

- **Why Kubernetes?**
  - Over **3,300 contributors**—a realistic stress test for any branching model.
  - Delivers a **new version every ~3 months** despite massive parallel work.
- **Observed workflow:**
  1. **main** stays clean, reflecting the latest stable release.
  2. New features (e.g., a scheduler improvement) are developed in **feature branches** (feature/scheduler-v2).
  3. When a set of features is ready, a **release branch** (release/1.28) is cut from main.
  4. Only **bug fixes** and release-specific tweaks land on the release branch until it is tagged and published.
  5. After the release, the branch is merged back to main and **deleted**.
- **Takeaway for personal projects:**
  - Mimic this pattern even on a small repo to demonstrate **professional workflow** on your résumé.

📅 Step-by-Step Workflow (Numbered)

1. **Start** on main.
2. **Create a feature branch** for each new capability:

```
git checkout -b feature/<name>
```

3. **Develop & test** on the feature branch.
4. When ready, **open a pull request** and merge back to main.
5. **Cut a release branch** once a milestone is reached:

```
git checkout -b release/<version> main
```

6. Perform **release-candidate testing**; only apply critical fixes.
7. **Tag** the release and merge the release branch into main (and optionally into develop if you use one).
8. **Delete** the temporary branches to keep the repo tidy.

✅ Key Takeaways

- **Branching isolates work**, protecting the stable product while enabling rapid innovation.
- A **well-defined strategy** (main → feature → release) is essential for predictable, on-time deliveries.
- Real-world projects like **Kubernetes** prove the model scales to thousands of contributors.
- Knowing this strategy is **highly interview-relevant** and can be showcased on your resume. ## 🌳 Master (Main) Branch

> **Definition:** The central, always-up-to-date line of development. All completed work—features, releases, and hot-fixes—must eventually be merged back here so that it reflects the latest stable codebase.

- Serves as the single source of truth for developers.
- Continuous integration pipelines typically run against this branch.
- New **release branches** are created *from* the current tip of master.

✨ Feature Branches

> **Definition:** Short-lived branches used to develop a **specific new capability** (e.g., feature-percentage, feature-bike).

- One branch per feature or large change.
- Multiple developers can work in parallel on different features.
- After feature completion and thorough testing, the branch is **merged into master** and can be deleted.

Typical lifecycle

1. Create feature-<name> from **master**.
2. Implement code, run unit tests.
3. Open a pull request → review.

4. Merge into **master**.
5. Delete the feature branch.

🚀 Release Branches

> **Definition:** Branches that capture the exact code snapshot that will be shipped to customers for a particular version (e.g., release-1.27).

- Created **from master** at the moment a new version is planned.
- All subsequent testing (end-to-end, performance, regression) happens on this branch.
- Only **bug-fixes** and **hot-fixes** are merged back into a release branch; new features continue on master.
- When the release is approved, the release branch is tagged and delivered.

Key properties

- Long-lived only for the duration of the release cycle.
- May be maintained alongside older release branches for patch support.

🛠️ Hotfix Branches

> **Definition:** Extremely short-lived branches created to address **critical production issues** that cannot wait for the next regular release.

- Spawned from the **release branch** that is currently in production (or from master if no release branch exists).
- After the fix is verified, the hotfix is merged **both into the active release branch and back into master** to keep all lines consistent.

Typical hotfix flow

1. Identify urgent bug in version *X*.
2. Create hotfix-<description> from **release-X**.
3. Apply fix, run regression tests.
4. Merge into **release-X** (to ship immediately).
5. Merge into **master** (to keep future releases up-to-date).
6. Delete the hotfix branch.

🔄 Merge Flow & Rules

Source Branch	Destination(s)	When to Merge	Reason
Feature	Master	After feature is complete & reviewed	Integrate new capability into main line
Release	Master	When release is finalized (optional)	Keep master up-to-date with any release-only fixes
Hotfix	Release & Master	Immediately after verification	Deploy urgent fix & propagate to future work
Master	Release (new)	When starting a new version cycle	Snapshot current stable code for testing

Rule of thumb: *Every change that leaves a branch must flow back to master.* This ensures master always reflects the latest functional code.

📚 Real-World Examples

1. Kubernetes Repository

- **master**: Active development of the next Kubernetes version.
- **feature branches**: feature-rate-limiting, feature-server-set, feature-workload-GA, etc.
- When a scheduled release (e.g., release-1.27) is planned, a new release branch is created from master.
- Ongoing development continues on master while testing occurs on release-1.27.

2. Uber Analogy

Phase	Branch Created	Purpose
Initial cab-only app	master	Core product

Add bikes	feature-bikes	Develop bike support in parallel
Add intercity rides	feature-intercity	Separate development stream
Prepare version 3	release-v3 (from master)	Consolidate all merged features, run full testing, ship to customers
Critical bug in v3	hotfix-pricing	Fix immediately, merge into both release-v3 and master

---

📊 Branch Summary Table

Branch Type	Typical Lifespan	Created From	Merged To	Main Goal
master/main	Indefinite	—	— (receives merges)	Central, always-current code
feature	Days–Weeks	master	master	Add new, possibly breaking functionality
release	Weeks–Months (until shipped)	master (at release start)	master (optional)	Stabilize a version for customers
hotfix	Hours–Days	release (or master)	release **and** master	Quickly resolve production-critic bugs

---