Why is master branch is the default branch after git init.

You have two feature branches one for developer A and another for developer B . developer A completed his worked and pushed it to testing . now developer B also completed his work after few hours or next day and pushed it to testing or qa. And now the problem is “merge conflict” and you have to make sure that no unwanted things shall go in production .”How will you handle this merge conflict or resolve this ” dev B is getting merge conflict.

If a centralized server like svn goes down then developers cant share there code , but what if in distributed system like github goes down . how will developer share there code .

how would you implement git branching strategy for your project or How would you do git branching strategy for your project .

Important - master’s branch access is only given to devops architect or client side architect

error - user cant login , user cant upload photo

Dev - test- uat - release(stage env) - prod(master)

Back merge - hot fix = bug fix

Why = · **Back-Merge the Fix**:

After Prod is updated, **back-merge** the fix:

From Prod → Stage Then from Stage → Dev

Ensures all branches are **in sync** with the latest fix. **Result**:**All environments (Dev, Stage, Prod)** have the same, fixed code.Prevents future conflicts or bug reappearance.

/feature/ photo upload

Dev 2

/ feature / ui

Dev 1 - log in ui

We plan release every 2 weeks and the version gets entirely change.

It is tested by management of team / managers / tech architect/

Merge hotfix into master branch with version 0.0.2

Hotfix branch - fix the bug when no connection of user login

- fix the bug when user cant upload the photo

यहाँ पूरा **Git Branching Strategy & Merge Conflict Resolution** वीडियो का **सिंपल पॉइंट्स में पूरा सारांश** है, **कोई पॉइंट मिस नहीं किया गया है**:

## ✅ ****Git Branching Strategy Summary****

### 🔷 1. DevOps सिर्फ टूल्स नहीं, पूरी टीम की समझ जरूरी

DevOps एक बड़ा प्रोसेस है – इसमें टूल्स जैसे Git तो हैं, लेकिन सबसे जरूरी है **टीम का सहयोग**।

हर Dev को process, ticketing system, branching naming conventions समझनी चाहिए।

### 🔷 2. Sprint और Ticket से जुड़ा Git Branch नाम बनाना

Project Jira, ServiceNow जैसे tools से manage होता है।

हर task/ticket का एक unique नंबर होता है, उसी से Git branch name बनाओ:

Example: feature/registration-JIRA-123

इससे architect को पूरी visibility मिलती है कि कौन सा commit किस ticket से जुड़ा है।

### 🔷 3. Epic > Story > Task Structure

**Epic**: High-level requirement जैसे Authentication module।

**Story**: Epic के under छोटे काम, जैसे Registration, Login।

अलग-अलग Dev अलग-अलग story पर काम करते हैं और अपनी branches बनाते हैं।

### 🔷 4. Environment की समझ

Environment नाम सिर्फ physical servers नहीं होते:

Examples: **Dev**, **QA**, **UAT**, **Prod**

हर org का budget और infra अलग होता है।

Environment के हिसाब से branching strategy तय होती है।

### 🔷 5. Branching बेसिक्स (Greenfield Project Scenario)

नए प्रोजेक्ट (greenfield) में:

पहला branch = master

Initial code/boilerplate goes in master by Tech Leads

Features वाले branches master से बनते हैं:

क्योंकि master में **tested**, **client-approved** code होता है।

### 🔷 6. Feature Branches कैसे बनती हैं

Dev A और Dev B अपने-अपने features पर काम करते हैं।

Example branches:

feature/JIRA-123-registration

feature/JIRA-124-login

ये branches master से बनती हैं ताकि base code same रहे।

### 🔷 7. Develop vs Master Confusion

Master में deploy होने के बाद changes develop में back-merge होते हैं।

कभी production में bug आता है, तो hotfix branch **master से** बनाई जाती है।

### 🔷 8. Merge Conflict Scenario Explained

Dev A ने पहले PR raise कर दिया develop में।

Dev B ने late किया और अब develop में और commits आ चुके हैं।

अब Dev B को merge conflict मिलेगा develop के साथ।

### 🔷 9. Common Mistake – Direct Conflict Resolve in Feature Branch

Dev B develop को merge करता है **directly अपनी feature branch में**।

Problem: उसमे और Devs के bug वाले commits भी आ जाते हैं।

जब PR raise करता है, वो सारे buggy commits साथ में push हो जाते हैं।

### 🔷 10. Best Solution – Create Intermediate Merge Branch

Dev B को एक extra branch बनानी चाहिए:

merge/feature-login

यह branch develop से बनाएं, उसमें अपनी feature branch merge करो।

Conflict resolve करो वहीं।

फिर clean PR develop के लिए raise करो।

✅ इससे:

बाकी devs के buggy code avoid होते हैं।

UAT में सिर्फ **tested code** जाता है।

### 🔷 11. Cherry-pick को avoid करें large teams में

Cherry-pick करना OK है छोटे scale पे।

बड़ी टीम में scalable नहीं होता।

Intermediate merge branch is scalable and safe.

### 🔷 12. Master is Golden Branch

Master पर कोई direct commit नहीं करता।

Access सिर्फ lead/architects को होता है।

उस पर **Branch Protection** apply किया जाता है।

### 🔷 13. Interview में Impress करने वाला Point

ये strategy (intermediate merge branch) interview में unique और practical लगेगा।

ऐसा answer लेने वाला सोचता है कि candidate team of 250 devs को handle कर सकता है।

## 🎯 Bonus: Final Takeaways

| **Concept** | **Summary** |
| --- | --- |
| Branch Naming | feature/<task-name>-<ticket-id> |
| Avoid Direct Develop Merge | Create merge/feature branch |
| Conflict Resolve Location | Merge branch > then PR to develop |
| Master = Golden Branch | No direct push, only from fully tested code |
| Focus | Think **team-scale**, **client-ready** |

✅ **All points rechecked** and simplified without losing original meaning.  
Let me know if you want the same in **tree diagram** format!