2. Source control strategy
└ **2.1 Branching and workflows**
└ **2.1.3 Branch protections and merge restrictions**

---

1. What are branch protection rules and why are they critical in DevOps workflows?
2. How do you configure branch protection in GitHub and Azure DevOps?
3. What types of merge restrictions can be enforced using branch protection policies?
4. How can you require status checks before allowing merges?
5. How do you restrict who can push or merge to protected branches?
6. What is the purpose of requiring signed commits or commit signatures?
7. How do you enforce code owner approvals in branch protection?
8. How can you enforce linear history and prevent force-pushes?
9. What is the impact of branch protection on CI/CD automation?
10. What best practices should be followed for branch protections and merge restrictions?

---

1. **What are branch protection rules and why are they critical in DevOps workflows?**

Branch protection rules enforce

- required checks,
- reviews,
- and restrictions on branches,

preventing unauthorized or unsafe changes and ensuring code quality and stability.

---

2. **How do you configure branch protection in GitHub and Azure DevOps?**

In *GitHub*, go to *Repository Settings > Branches > Add rule*. In *Azure DevOps*, navigate to *Project Settings > Repositories > Branches > Add policy* to the desired branch.

---

3. **What types of merge restrictions can be enforced using branch protection policies?**

You can require

- pull request reviews,
- passing status checks,
- code owner approval,
- no unresolved conversations,
- and restrict merges to certain users or teams.

---

4. **How can you require status checks before allowing merges?**

Set required status checks in branch protection so PRs can only merge if defined CI jobs (build, test, security scan) succeed.

---

5. **How do you restrict who can push or merge to protected branches?**

Configure branch protection or security policies to allow pushes or merges only from specified users, teams, or service accounts.

---

6. **What is the purpose of requiring signed commits or commit signatures?**

Requiring signed commits verifies the identity of commit authors, helping prevent unauthorized or untraceable changes.

7. **How do you enforce code owner approvals in branch protection?**
Define a CODEOWNERS file in the repo and require code owner approval via branch protection settings for changes to specific files/paths.

---

8. **How can you enforce linear history and prevent force-pushes?**
Enable "*Require linear history*" and "*Prevent force pushes*" options in branch protection to maintain a clean commit history and prevent data loss.

---

9. **What is the impact of branch protection on CI/CD automation?**
Branch protection ensures all code passing through CI/CD meets quality gates, but may require automation accounts or bots to be explicitly permitted for merges or status check reporting.

---

10. **What best practices should be followed for branch protections and merge restrictions?**
    - Apply protections to main/release branches
    - Keep required checks relevant
    - Regularly review permissions
    - Require at least one reviewer
    - Avoid over-restriction that blocks productivity