2. Source control strategy

└ 2.2 Repository Configuration and Maintenance

└ 2.2.2 Optimize scaling

---

1. What challenges occur as Git repositories grow in size and activity?
2. What is Scalar and how does it help with large repositories?
3. How do partial clones and sparse-checkout help optimize scaling?
4. How can cross-repository sharing reduce duplication and improve efficiency?
5. What are best practices for structuring large monorepos?
6. How do you monitor and maintain repository performance at scale?
7. How do you clean up and archive old branches and tags efficiently?
8. What strategies exist to optimize clone and fetch performance in large repos?
9. How does Azure DevOps or GitHub handle scaling for large enterprise repositories?
10. What tools or commands are used to analyze and reduce repository size?

---

1. **What challenges occur as Git repositories grow in size and activity?**

Large repos can cause slow `clone`, `fetch`, and `checkout` operations, increased storage, network costs, and degraded performance for CI/CD and contributors.

---

2. **What is Scalar and how does it help with large repositories?**

Scalar is a Microsoft tool that optimizes very large monorepos by improving Git's handling of large working trees, accelerating commands like `status` and `checkout`, and enabling virtualized file systems.

---

3. **How do partial clones and sparse-checkout help optimize scaling?**

Partial clones allow downloading only required objects. `Sparse-checkout` enables working with a subset of the repo, reducing disk usage and improving performance.

---

4. **How can cross-repository sharing reduce duplication and improve efficiency?**

Cross-repo sharing (submodules, subtrees, or package feeds) allows teams to reuse code/assets across projects without duplicating large files or code, minimizing storage and bandwidth usage.

---

5. **What are best practices for structuring large monorepos?**

- Organize by logical boundaries
- Use clear folder structures
- Minimize large binaries
- Automate code ownership
- Use sparse-checkout for focused workflows

---

6. **How do you monitor and maintain repository performance at scale?**

Regularly review repository size, commit history, and object count. Use monitoring tools and Git analytics to spot performance bottlenecks and take action early.

---

7. **How do you clean up and archive old branches and tags efficiently?**

Use `git branch -d/-D` for branches, `git tag -d` for tags, and prune remote references. Archive or delete stale branches and tags periodically to keep the repo clean.

8. **What strategies exist to optimize clone and fetch performance in large repos?**
    - Use shallow clones (git clone --depth),
    - partial clones,
    - sparse-checkout,
    - and avoid cloning unnecessary history or content.

---

9. **How does Azure DevOps or GitHub handle scaling for large enterprise repositories?**

Both platforms offer high-availability storage, optimized network delivery (e.g., CDN), *LFS support*, *Scalar*, and monitoring for large repos. *GitHub* also supports monorepo scaling features.

---

10. **What tools or commands are used to analyze and reduce repository size?**
    - *git-sizer*
    - git count-objects
    - git gc
    - git filter-repo

help measure, clean, and reduce repo size. Remove large files or history as needed for optimization.