**Jose Franco**  
06/10/25  
**Module 3.2 Assignment: Version Control Guidelines**

Version control is an essential component of modern software development and a cornerstone of DevOps culture. It allows teams to track changes, collaborate effectively, and maintain a reliable codebase. As development practices evolve, so too do the guidelines for managing version control systems (VCS). This paper reviews version control guidelines from three sources: Atlassian, GitLab, and The DevOps Handbook by Gene Kim. It compares these guidelines, evaluates their relevance today, and presents a personal list of essential best practices.

### ****Overview of sources and guidelines****

**1. Atlassian: Git workflows and best practices**  
Atlassian recommends well defined Git workflows like feature branching, Git Flow, and Forking.

**Key practices include:**

* Use feature branches for isolated changes
* Write small, frequent commits with clear messages
* Rebase before merging to maintain a clean history
* Protect the main branch and use pull requests for merging
* Regularly sync with the main branch to minimize conflicts

**2. GitLab: Version control recommendations**  
GitLab emphasizes DevOps integration and CI/CD readiness.

**Key practices include:**

* Make incremental, small changes
* Keep commits atomic
* Develop using branches
* Write descriptive commit messages
* Obtain feedback through code reviews
* Identify a branching strategy (e.g., GitLab flow)

**3. The DevOps Handbook by Gene Kim**  
Gene Kim and his co-authors go beyond code to advocate versioning everything, application source code, configurations, infrastructure, environments, and deployment processes. The idea is to treat version control as a **single source of truth** for the entire system.

Key practices include:

* Use version control for all production artifacts: infrastructure-as-code, scripts, test cases, and environment configurations
* Embrace **trunk-based development** for faster integration and fewer merge conflicts
* Automate testing, builds, and deployments triggered by version control events
* Enable easy rollback and recovery through comprehensive version history
* Promote cross functional visibility by having QA, Ops, and Infosec work within the same version control system

The DevOps Handbook stresses that environments typically have more configuration than code, making it even more critical to version everything, not just software. This approach ensures stability, faster recovery, transparency across departments, and reproducibility of the full system, including tools, artifacts, and pre-production infrastructure.

### Comparison of guidelines

While all three sources stress the importance of consistent version control, their emphasis differs based on organizational maturity and development philosophy:

* **Atlassian** supports a flexible, branching heavy model suitable for teams that need isolation and parallel development. However, its reliance on long-lived feature branches can introduce merge debt and slow down continuous delivery.
* **GitLab** provides practical, automation-friendly advice, focusing on small, atomic commits and CI/CD pipelines integrated directly with version control.
* **The DevOps handbook** goes further by incorporating the entire DevOps toolchain into version control, promoting **trunk based development** to maximize speed and reduce complexity, and advocating for versioning of environments, infrastructure, and supporting scripts.

### Outdated or conflicting guidance

* **Atlassian’s promotion of Git Flow**, which relies on long-lived feature branches, is increasingly seen as too rigid and outdated for fast paced DevOps environments. Merge conflicts and deployment delays are more likely under this model.
* **Trunk-based development**, as emphasized in The DevOps Handbook, aligns better with continuous integration and delivery, promoting speed, stability, and reduced merge complexity. Its guidance is highly relevant today, especially for teams practicing CI/CD.

### My version control guidelines

Based on research and current practices, here are the guidelines I believe are most critical:

**• Version control everything**  
Not just source code. Infrastructure, configuration files, scripts, documentation, tests, and environment settings should all be in version control. This ensures full traceability, better collaboration, and quick recovery.

**• Keep the main branch always deployable**  
The main or master branch should always be in a deployable state. Broken builds delay teams and reduce trust in automation.

**• Use trunk based development when possible**  
Short-lived branches merged frequently into the main branch reduce integration issues and support continuous delivery.

**• Automate testing and deployment on every commit**  
CI/CD tools like GitLab CI or Jenkins should automatically run tests and deploy on each commit to validate quality and reduce manual effort.

**• Write clear, frequent, atomic commits**  
Each commit should be focused, meaningful, and well described. This improves reviews, troubleshooting, and rollback precision.

**• Review code with merge/pull requests**  
Peer reviews combined with automated checks improve code quality and foster team knowledge sharing.

**• Use tags to mark releases**  
Tagging specific commits as release versions helps with auditing, rollback, and deployment management.

### Conclusion

Version control remains foundational to software delivery success. Atlassian emphasizes flexible workflows, GitLab prioritizes integration with automation, and The DevOps Handbook urges comprehensive version control of code and environments alike. While older branching models like Git Flow may not suit fast paced delivery teams, modern practices such as trunk based development and environment versioning are increasingly vital.

My recommended guidelines focus on automation, deployability, and clarity, qualities essential for high performing DevOps teams. Version control is no longer just a developer’s tool, it is the backbone of collaboration, reliability, and resilience across the entire software lifecycle.

### References

1. Atlassian. (n.d.). Comparing Workflows. Retrieved from <https://www.atlassian.com/git/tutorials/comparing-workflows>
2. GitLab. (n.d.). What are Git version control best practices?. Retrieved from <https://about.gitlab.com/topics/version-control/version-control-best-practices>
3. Kim, G., Humble, J., Debois, P., & Willis, J. (2021). The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations (2nd ed.). IT Revolution Press.