**Version Control Guidelines**

Robert Minkler

College of Science & Tech, Bellevue University

CSD 380 DevOps

Professor John Woods

April 6, 2025

**Version Control Guidelines**

Version Control is a valuable tool in software development, recording a historical record of changes that can be reviewed and reversed as needed. Any tool is only as good as how well it is utilized, and this holds true for version control as well. If commits are inconsistent, too large, or too small, the benefits of version control can be lost. Too many changes in a single commit can make finding a bug and repairing it more challenging. On the other hand, commits that don’t complete a feature can break code others are working on and make tracking a single feature commit more challenging. Following best practices in version control can make working as a team easier and version control systems more effective.

I found three sources regarding Version Control Guidelines. GitLab has an educational blog post discussing best practices. Professor Ernst at Washington University has a site with advice on version control practices. Finally, Perforce Software’s Schiestl shares tips on best practices. All three sources share some of the same concepts. The two topics that resonated across all three sources included using atomic commits that target a single unit of work instead of large commits with multiple changes. If we need to make two unrelated changes, we should do it in two separate commits. Secondly, write descriptive commit messages. This makes everyone’s job easier as the messages clearly indicate what work was done on that specific commit.

GitLab suggests making frequent small commits to a local branch, then merging complete atomic commits to the main branch. Small commits to the main branch would cause additional issues, but working on changes in a separate branch enables the commits to be grouped together and thoroughly tracked while staying separate until the merge. GitLab and Ernst suggest coordinating with coworkers regarding a branching strategy so everyone has a similar process for branching and committing to the project. Some branching strategies include per-feature branching, per-developer branching, GitFlow, and a centralized workflow. When everyone uses a similar workflow, it is easier to understand the progress of other developers on the team, helping avoid conflicts when merging and working more effectively as a team. GitLab and Schiestl recommend code reviews before merging commits to the main branch in an effort to catch conflicts before they occur. Ernst and Schiestl recommend testing the code before committing to ensure nothing breaks when merged with the rest of the project.

Ernst includes a few recommendations that were not mentioned by the other sources. Incorporate others’ changes frequently. Share changes frequently, which aligns with incorporating others’ changes frequently. As everyone frequently shares and incorporates code across the project, bugs and conflicts can be found and fixed faster. He stresses that the tools are line-based, and realigning or justifying text changes the full line of code and messes up the tracking in VCS. He recommends not committing generated files. Finally, have a solid understanding of the merge tool before a merge fails. When a merge fails is not the best time to learn how to use the merge tools.

Schiestl mentions protecting assets. This may include IP or secrets. Protecting the assets in a repository is an important consideration. I have seen cases where secrets were committed to a public repository. Passwords, tokens, confidential data, and legally protected assets should not be posted to a repository. Since repositories track the full history of changes to a file, even removing those secrets can still leave trackable remnants in the VCS.

Overall, all the recommendations from each source have relevance. They all share a single goal, to make the most of version control while working within a team. Making atomic commits to the main branch that fully implements only a single unit of work and writing clear and descriptive commit messages to describe the work completed for future tracking and review. Following these practices are vital to properly utilizing a VCS with a team.

My list of recommendations based on my research:

1. Only make atomic commits that fully implement a single unit of work so each commit can be tracked and reversed if necessary.
2. Write descriptive commit messages so others understand what the commit does.
3. Test the code before merging it with the working branch to avoid bugs.
4. Hold code reviews before merging with the working branch to catch conflicts.
5. Don’t commit code that breaks a build.
6. Coordinate with the team for a consistent branching technique.
7. Don’t commit generated files because they are not part of the codebase.
8. Don’t include protected assets in a public repository to maintain confidentiality and IP rights.

When a team can work together and follow these best practices, it makes future revisions, attribution, bug tracking, and remediation easier. Haphazardly using these tools without a strategy can result in a repository that is more challenging to maintain.

**Resources**

GitLab. (n.d.). *What are git version control best practices?*. The DevSecOps Platform. <https://about.gitlab.com/topics/version-control/version-control-best-practices/>

Ernst, M. (2024, April 8). Version control concepts and best practices. <https://homes.cs.washington.edu/~mernst/advice/version-control.html>

Schiestl, B. (2020, May 21). *8 version control best practices*. Perforce Software. <https://www.perforce.com/blog/vcs/8-version-control-best-practices>