Alisa Steensen 6/12/25

Module 3.2 DevOps

**Version Control Guidelines**

Version Control is a tool used for tracking changes, history tracking, and bringing together collaboration amongst developers. It is a essential tool for developers to use in Software Development and enables multiple people to work simultaneously on a project. There are some important guidelines that must be followed to ensure there are no outdated practices and the workflow remains effective. Below are some important guidelines to follow for Version Control best practices.

After researching 3 different articles on guidelines for best Version Control practices, I have seen there are some differences in what they expect and for the majority they are the same.

Michael Ernst, Version Control Advice

These guidelines emphasize individual developer discipline. He focuses primarily on:

* Descriptive commit messages
* Logical commits
* Avoiding generated files
* Avoiding rewriting history on shared branches
* Using merge tools properly.

Perforce, 8 Version Control Best practices

Perforce’s article is essentially geared towards teams working on larger codebases and recommends the following:

* Consistent workflow and branching
* Frequent commits
* Good commit messages
* Avoiding large commits
* Using code reviews
* Ensure commits are traceable
* Keeping a clean repository history
* Protect your assets (security)

GitLab, Version Control Best Practices

GitLab practices are tightly integrated with CI/CD pipelines and collaborative development, their key principles are:

* Making small changes
* Develop using branches
* Obtain feedback through code review
* Atomic and focused commits
* Descriptive commit messages
* Using Git commands like rebase and squash thoughtfully
* Pairing version control with automated testing and deployment

All three of these sources agree on the importance of atomic commits and meaningful commit messages. These practices will help to maintain clarity while working through any project. Ernst is unique in his warning against rewriting history, especially in shared branches, while GitLab allows it within private branch sectors as long as its managed. Perforce and GitLab both put a strong emphasis on collaborative workflows. This is especially so for branching strategies and code reviews, which is less emphasized by Ernst. His guide leans more toward individual contributor practices.

Another thing that I found is a distinction between is the CI/CD integration. GitLab is the only one out of the three that discusses how version control ties to automated pipelines, testing, and deployment. GitLab really does support modern DevOps orientation. While Erst and Perforce focus more on version history and team collaboration.

Additionally, Erst also talks about avoiding generated files and mastering merging tools, which are useful to remember and practice for clean repositories.

Below is a list of my own guidelines that I feel are very important. I think combining different practices from my research has created a well-rounded set of guidelines to follow. I chose these because I feel they are the most important to have effective repositories and clean code.

First, developers need to have well documented and thorough commit messages. It is important to so that others working can see the documentation and understand the history of the project. This helps reduce confusion and give a clear picture as to why the changes were made.

Second, commits to the main branch should have a single purpose and should ultimately implement that purpose. For example, commits should not include new features, typo fixes and a bug fix in one. They should be 3 separate commits. Each branch should also have a single purpose and when merged to the main branch, and should completely implement that purpose.

Third, if you end up committing more changes than you meant to, ensure that you double check before committing to ensure that the commit accurately reflects the intended purpose. There are tools such as git status and git diff that can help you verify what will be committed.

Fourth, when working with a team, it is very important to pull and incorporate others’ changes frequently. This will prevent merge conflicts and ensures your local branch stays up to date. Additionally, you should be pushing your changes often so that others can see your latest updates.

Fifth, you will want to ensure that you and others on your team are on the same page. Version Control system can often have merge changes that different people make at the same time. And when two people edit at the same time it creates a conflict that will need to be manually resolved. To avoid this, strive to avoid conflicts. Coordinate with others before you plan to make significant changes to a file that others may be editing.

Sixth, understand you merging tools and how to use them. When knowing how to properly handle merge conflicts is critical to maintaining a clean codebase and avoiding any accidental overwrites.

Seventh, it's important not to commit generated files such as compiled binaries, temporary files, or environment specific settings. These types of files can clutter the repository and create unnecessary conflicts to happen. Instead of this, you should be using .gitignore file to exclude them.

Last, remember that version control systems are line based. This means that even a small change on the single line will show as a modification. To ensure no unnecessary changes are made, be mindful of the formatting and spacing.

In conclusion, I feel that each source has provided valuable Version Control advice. The emphasis on each source varies depending on the audience and purpose. While Ernst offers a precise and cautious approach for developers, Perforce delivers scalable practices for team-based development, and GitLab incorporates a DevOps perspective with CI/CD integration. And despite the differences, their shared recommendations create a well-formed foundation of best practices to be used in a Version Control environment.

References:

Ernst, M. D. (n.d.). *Version control concepts and best practices*. University of Washington. Retrieved June 10, 2025, from <https://homes.cs.washington.edu/~mernst/advice/version-control.html>

Perforce Software. (n.d.). *8 version control best practices*. Perforce. Retrieved June 10, 2025, from <https://www.perforce.com/blog/vcs/8-version-control-best-practices>

GitLab. (n.d.). *Version control best practices*. GitLab. Retrieved June 10, 2025, from <https://about.gitlab.com/topics/version-control/version-control-best-practices/>