**Module 3.2 – Assignment: Version Control Guidelines**

Dan Zhu

Software Department, Bellevue University

CSD380-O307 DevOps (2261-DD)

Professor Darren Osier

August 31st, 2025

**Version Control Guidelines**

Gitlab:

1. Make incremental, small changes
2. Keep commits atomic
3. Develop using branches
4. Write descriptive commit messages
5. Obtain feedback through code reviews
6. Identify a branching strategy

Git Tower:

1. Commit Related Changes
2. Commit Often
3. Don’t Commit Half-Done Work
4. Test Before You Commit
5. Write Good Commit Messages
6. Version Control is not a Backup System
7. Use Branches
8. Agree on a Workflow

Perforce

1. Commit Changes Atomically
2. Commit Files with a Single Purpose — Not as a Backup
3. Write Good Commit Messages
4. Don’t Break Builds
5. Do Reviews Before Committing to a Shared Repository
6. Make Sure Every Commit Is Traceable
7. Follow Branching Best Practices
8. Protect Your Assets

Above are the lists of the best version control practices from GitLab, Git Tower, and Perforce. They have some standard best practices mentioned in all three resources.

The first one is that developers should keep commits atomic. When developers commit, they should commit a change with one feature. For example, developers will have two commits if there are refactoring and new features for code changes. One is for refactoring code, and the other is for a new feature. It is easy to roll back a change when you have atomic commits.

The second one is to commit often. GitLab recommends making incremental, small changes. Git Tower mentioned the commit frequently. Committing often with minor modifications can share the latest updates and reduce merge conflicts.

The third one is the branching strategy. According to Git Tower, branching is one of Git’s most powerful features. It is a single-branch strategy in which developers have one main branch to hold the release version code. When there is a new feature, the developer will create a feature branch of the main branch. Once the feature code is done, the feature branch will merge to the main branch. The branching strategy avoids conflicting workflows.

The fourth one is to write descriptive commit messages. GitLab mentioned that descriptive commit messages are as important as the change itself. The message should start with a verb in the present tense in imperative mood to indicate the purpose. Git Tower also states the messages should be summarized in imperative, present tense, with the primary motivation. Perforce says that each commit should explain the why. It is essential to have a descriptive commit message. For example, you made a commit with a “guess what I did” message. It doesn’t give team members any clue about the code change, and it will make the team less efficient because they have to go into the code and check what the change is and why it was made.

The last one is code reviews. GitLab states that requesting feedback from others is an excellent way to ensure code quality. Perforce asks to do reviews before merging the code. Code reviews are a critical process. Doing code reviews will improve code quality and provide an opportunity to learn from each other.

After identifying the standard practices, we also found some unique items in each source. Git Tower recommends only committing completed work, testing before committing, and agreeing on a workflow. It also mentions that version control is a backup system. Perforce has some unique rules as well. Don’t break builds, make sure commit is traceable, and protect your assets.

Based on all three resources, atomic commits, commit often, branching strategy, write descriptive commit messages, code reviews, and commit before testing are essential practices. Atomic commit group-related code in one commit, and easy to roll back. Commit often can share the latest code change with the team. A branching strategy is essential as well. It helps reduce the conflicts. Write a descriptive commit message providing good context about the code change. Code reviews can ensure the code quality and help us learn from each other. The last one is constantly testing before committing. It is good practice to run your code as long as you make a change.

**Reference**

Fournier, B. (n.d.). *Git best practices*. Tower. Retrieved August 30, 2025, from [https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices](https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices?utm_source=chatgpt.com)

GitLab. (n.d.). *Version control best practices*. GitLab. Retrieved August 30, 2025, from [https://about.gitlab.com/topics/version-control/version-control-best-practices/](https://about.gitlab.com/topics/version-control/version-control-best-practices/?utm_source=chatgpt.com)

Perforce. (2023, August 3). *8 version control best practices*. Perforce Software. Retrieved August 30, 2025, from [https://www.perforce.com/blog/vcs/8-version-control-best-practices](https://www.perforce.com/blog/vcs/8-version-control-best-practices?utm_source=chatgpt.com)