Austen Rhyce Erickson
CSD380: DevOps
Assignment 3.2: Version Control Guidelines
1/26/25

Regarding version control guidelines and best practices for software development, I found three articles online providing some good information.

The first, an article called What are Git version control best practices? by GitLab states that achieving good team collaboration and consistency in code can be achieved by adopting agreed upon best practices in Git/version control.

Some of the key recommendations included:

1) Making incremental changes: Small code changes reduce integration/merge conflicts and simplify testing.
2) Keep commits atomic: Ideally, each commit should address a single task for fix. Which makes code reviews and rollbacks easier.
3) Develop using branches: Branches allow for isolated development, ensuring that the main codebase remains stable.
4) Write descriptive commit messages: Clear messages provide context for changes, aiding future code reviews and collaboration.
5) Obtain feedback through code reviews: Regular reviews enhance code quality and promote knowledge sharing among team members.

The second, an article called 8 Version control best practices by Perforce also had a list of best practices but didn't make any claims as to why they are good to follow.

The key points from Perforce were:

1) Commit changes atomically: Ensure that all files related to a task are committed together to maintain consistency.
2) Commit early and often: Frequent commits help in tracking progress and facilitate easier debugging.
3) Use branching extensively: Branches support parallel development and feature isolation.

4) Use tags: Tags mark specific points in the repository's history, such as releases.
5) Implement gated commits: Automated checks before commits help maintain code quality.
6) Maintain a single source of truth: A centralized repository ensures that all team members work from the same codebase.
7) Avoid unnecessary dependencies: Keeping the codebase modular reduces complexity.
8) Document your work: Comprehensive documentation aids in onboarding and future development.

The third, an article called Version control best practices for efficient software by Modern Requirements advocates for the benefits of version control and offers best practice to maximize those benefits.

The best practices from Modern Requirements:

1) Use clear version numbers: Consistent versioning helps in tracking and managing releases.
2) Maintain a central repository: A centralized storage ensures accessibility and consistency.
3) Develop branching and merging strategies: Effective strategies allow simultaneous development and integration.
4) Provide good documentation and commit messages: Clear documentation and messages enhance collaboration and traceability.
5) Ensure security and protect intellectual property: Implementing security measures safeguards the codebase and proprietary information.

There were a few recurring best practices amongst these sources that stood out. The practice of using atomic commits was mentioned in both the GitLab and Perforce article. These two sources also both advised using frequent commits for small, incremental changes. All three of the articles talked about using branching strategies to enable parallel development and a stable main branch code base. One point that was mentioned in both GitLab and Modern Requirements but not Perforce was the recommendation of using descriptive commit messages.

Interestingly, the Perforce article was the only source that emphasized using tags to mark specific points in a repository’s history as well as the only source that talked about using gated

commits. The Modern Requirements source was also unique in that it talked about the use of clear version numbers for tracking releases and discussed security in the context of version control. Another difference was that GitLab did not mention the importance of maintaining a central repository while Perforce and Modern Requirements did mention central repositories.

All of these articles have general consensus on many of the same points while also offering a unique perspective on additional considerations, such as tagging and security. However, in regards to how relevant these tips are in an ever evolving technology landscape, it can depend on the scale and structure of the software project.

For example, central repositories are not as relevant nowadays as distributed systems like Git. Also, version numbering is not as relevant in CI/CD pipelines as the automation replaces the need to think about manual versioning. Similar idea for manually tagging releases. Documentation has also begun to be more of an automated task, especially with the rise of AI. Branching and merging practices as well as meaningful commits continue to be the most relevant.

If I were to describe the most important guidelines and best practices when using version control I would say the following:

Make **small incremental commits** with each commit adding something concrete and **complete**. It may be tempting to push commits as a way to save your work, but if that work is an incomplete unit of work, the commit history gets messy. Multiple commits that say “work in progress” is confusing. It makes reverting commits more difficult. Small, concrete units of work also lead to smaller pull requests. Smaller pull requests are easier to review which leads to more thorough r**eviews by team members**. The **commit titles should be brief**, descriptive, and succinct.

Use branches. Isolating development from the main code base is important. The integrity of the main branch should be preserved and not meddled with. Only intentional, approved changes should be added to the main branch. Enforce this with **branch protection rules**. Every team member should have the same copy of the main branch. When you check out a branch off of main, you have a working playground to do your development work. Breaking stuff here is not a problem as the main branch is unaffected.

Adhering to these version control practices gives way to better collaboration, workflow, traceability, code quality, maintainability, and security.

Sources:

"What Are Git Version Control Best Practices?" *GitLab*, about.gitlab.com/topics/version-control/version-control-best-practices/.

Schiestl, Brent. "8 Version Control Best Practices." *Perforce Software*, 21 May 2020, www.perforce.com/blog/vcs/8-version-control-best-practices.

Satpathy, Arunabh. "Version Control Best Practices." *Modern Requirements*, 21 Aug. 2023, www.modernrequirements.com/blogs/version-control-best-practices/.