Brittni Ray

CSD 380

Module 3.2

**Version Control**

Comparison

**Common Guidelines Across Sources**

* Keeping commits atomic
  + All of the sources emphasize the idea of small changes being referred to as “atomic commits.” Meaning that every commit made should be its own, self-contained unit that will deliver a single, logical change—such as fixing a bug, adding a feature, etc—to the codebase.
* Descriptive commit messages
  + The sources refer to this differently, but all three agree that writing good commit messages is essential. A well-written, descriptive commit message makes it easier for reviewers and developers to understand the purpose of the commit at a later date.
* Don’t break builds
  + This was another item that the sources refer to in one way or another, and although it might seem obvious, this is clearly important enough to be emphasized. Before committing, this guideline asks that the committer check before pushing to ensure that this commit will not break the build, which will then stall the entire team's progress.
* Branching strategies
  + The articles differ in how they talk about branching. Still, all cite that it is important and that the strategy to satisfy this guideline may depend on how any given company is/chooses to organize. However, they do emphasize to keep branching simple and that it is essential for managing releases, new features, and bugs.
* Code reviews
  + The importance of reviewing code before it is committed to a shared repository was also emphasized in all three articles. Some expand this to say that doing such reviews can even help improve code quality before it is pushed.

**Differences Among Sources**

* Protect your assets
  + Although two of the sources mention this, one does not. This guideline is meant to keep information and any code written safe. It states that steps should be taken to protect assets and that passwords or other sensitive information should never be kept in any repositories.
* Make sure every commit is traceable
  + Only one article mentioned this, implying that being able to trace each commit back to an author is critical. It also outlines that keeping the authors of a commit up-to-date is the first step in tracing issues in case they arise.

**Are they relevant today?**

All of the guidelines that I had read about still seem like they would be relevant in the current atmosphere. This may be because most of the articles that I chose to look at or those that I found were all written or updated fairly recently. There were a few that worded things or broke up certain points differently, but for the most part, I think most of these are still relevant in today’s world.

Personal Guideline Importance List

**Personal Guideline List Choices**

1. Keep changes atomic
2. Ensure commits will not break the build
3. Write descriptive commit messages
4. Develop using branches
5. Review code before committing
6. Protect your assets

I chose these because they were the guidelines that I saw mirrored across all three lists, which I believe to be a great indicator of what guidelines are essential. The first guideline makes a lot of sense to me—pushing things one by one and keeping them in their own unit makes for cleaner batches of things that are easier to trace if needed. The second makes sure that the build will not break due to a faulty commit, which will be more productive for the team in its entirety in the end. The third helps explain what each commit is doing, so anyone can easily review or trace any point in commit history when necessary. The forth will ensure that a development team is able to make any changes as a test without touching the main codeline. The fifth I think is just great practice overall; reviewing code is always good before committing it just in case there is anything in there that—again—could break the build. Finally—although not every source included this—I think that it is always good to keep security at the forefront and ensure that any commits will not expose a codebase to any vulnerabilities.

Sources

*6 best practices for git version control*. Nulab. (n.d.). https://nulab.com/learn/software-development/version-control-best-practices/

GitLab. (n.d.). *What are git version control best practices?* https://about.gitlab.com/topics/version-control/version-control-best-practices/#develop-using-branches

Schiestl, B. (n.d.). *8 version control best practices*. Perforce Software. https://www.perforce.com/blog/vcs/8-version-control-best-practices