Kody Pope

6-15-24

Assignment 3.2

Version Control Guidelines

(Lott), shows us that version control guidelines aim to help everyone stay on the same page. At the same time, the article stresses that if we were presented with three documents, how would we know which one was the final document? The document examples could be document-name-V7, document-name-final, and document-name-new. Which one is the latest? However, we can eliminate this massive issue by letting us know which one the most up-to-date document is by using version control. It also helps us go back in history to see where a problem might have occurred and makes fixing or rectifying this issue much easier, as there is clear documentation of everything that took place.

(Lott) shares with us good practices to follow. These are as follows: define naming conventions to ensure everyone knows which version is the latest. Do not say this, "file.doc," and another file saying, "finalFile.doc." Instead, file1.doc, file2.doc, and file3.doc. Number two is to draw a clear line between what a version is to avoid half work or overwork. Number three is to collect input and collaborate all in one place to prevent duplication of the version, and number four is to make it easy to review and compare versions so they can easily see what has been changed.

(Schiestl) shares excellent practices they view as essential for version control guidelines. They are as follows: one, make commits automatically. This means you do not push the code yourself but have your code editors push the code for you. Two, commit files with a single purpose, not as a backup. Third, write good commit messages so that it is easy for someone to see why a commit was made. Fourth, do not break builds. Five, review before committing, although this may be an issue if commits are made automatically. Six, make sure every commit is traceable. Seven, follow branching practices. And eight, protect your assets.

("Best Practices for Version Control in 8 Steps - Rule of Tech"), shares with us their eight best practices for version control guidelines. They are as follows: one commit logic change sets, or in this case, they mean to say that we shouldn't be making significant massive changes but relatively small tiny little changes and then push it with the message saying here's what we did so that if a large portion of the code is being edited, we don't instead have one commit for a much more significant portion of change. Still, instead, a few tiny commits, such as taking away one or two lines of code. Number two would be committing early and often. This goes with the first one: we want to make many tiny little commits as soon as possible, not one big commit later. Third, write reasonable commit messages. Like the first article, this agrees that everyone should understand what we are trying to do. We should write a reasonable commit message that is easy to follow and understand. Number four is do not commit generated sources. ￼I believe this￼ means here do not create a whole other library file and then commit to our branch when this library can already be used and pulled from somewhere else. Number five is not to make a commit if the work is only half done. Number six, test before you commit, so only commit something that works. Seven is to use branches, and eight is to agree on a workflow. This means we only make necessary changes or decide what needs to be changed to save the company time and resources.

My list of best practices would be:

1. Many tiny commits early and often
   1. I suggest making minor changes that make it easy to understand what was done and commit to if anyone needs to go back and look and understand what was changed.
2. One single purpose per commit
   1. This one goes hand in hand with the first one. It is okay to make many tiny commits so that it is easy to understand what is being changed. Therefore, what was altered must have one purpose and a few changes with many purposes all in one commit, making it easy to follow and understand what went wrong, where, and why.
3. Write reasonable commit messages/ file naming conventions
   1. We need to make the commits small and easily understood, and the message as to why we made this must also be easy to follow. Otherwise, we might end up with a commit that never needed to be made in the first place.
4. Do not commit half-done work/ test before committing
   1. Again, if a commit does not need to be made in the first place. Or if it does not even work, then the work is half done or does not even work after testing, then we are not yet done with this change and, therefore, should not make a commit to the branch.
5. Agree on workflow/ clear definition between the start and end of a commit.
   1. I think it would be great to understand what change needs to be made for commit to be made in the first place; if it would waste company time and resources to make a change that isn't even necessary, then we shouldn't be adding a commit to our branch that may ultimately lead to other errors or confusion later on, if such work, didn't even need to be added to the enterprise.
6. Protect your assets
   1. I like this role as a guideline, as not only is it a good guideline to follow, but it is also good to consider always protecting our assets and all points in software development. What good is a commit or branch, for that matter, if malicious actors compromise and change it?

As far as practices that are no longer relevant, I would have to argue that making changes automatically may possibly lead to issues. giving that there are many best practices that include such as not making a commit if only half the work is done or a commit if the code cannot even pass the test, or even making a commit were can your message may not fully address what exactly was being changed. If what was being changed for example was deleting one character, and then suddenly your text editor pushing that change. It may auto fill in that one character was deleted, but what exactly was that change necessary for, which could lead to overall confusion.

For this reason alone, I would have to omit that practice from my personal list of best practices and keep my best practices as making full complete small commits with messages that show what was changed so that if anyone looked at it, it would be easy to follow. As well as making sure to protect my assets. And finally, not even making commits, if it was not even agreed-upon to make them, as well as following standards and practices for naming these commits and versions to help further better communication within the company.

Works Cited

"Best Practices for Version Control in 8 Steps - Rule of Tech." Rule of Tech, 29 May 2019, ruleoftech.com/2019/best-practices-for-version-control-in-8-steps. Accessed 17 June 2024.

Lott, Ethan. "Document Version Control Best Practices to Make Your Life Easier." Filestage.io, 9 Sept. 2022, filestage.io/blog/document-version-control/.

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