Mason Buller

Module 3.2 Assignment

11/10/24

Each source ranges from 4 to 6 “best practices”, and they each differ from each other. In the first source, the first practice is to define naming conventions so that everyone knows which version is the latest. Essentially, coming up with a consistent pattern to use when pushing updates so you can identify which is the most recent version. Second is to draw a clear line between versions so everyone knows when to stop giving feedback. If you don’t set a window for feedback, reviewers could be giving feedback when you’ve already begun working on a new version. Third is to collect feedback and collaborate in one place to avoid version duplication. This just means specifying where people should leave feedback so that everyone isn’t making their own versions and documents. Fourth is to make it easy for reviewers to compare versions so they can easily see what’s changed. You can save a copy of the old version so once you’ve created a new version, you can go back and compare changes.

In the second source there are five best practices listed. First is to commit frequently with meaningful messages. This can help other contributors better understand what changes have been made without having to review the entire file. Second is to use branching wisely. Creating a large number of branches can start to make the structure more complicated and difficult to navigate. Keeping them limited to major updates can reduce conflict later on. The third practice is to keep a clean history. A large number of commits can clog up the projects’ history and make it difficult to look back on. Fourth is to set and follow naming conventions. You can use something like bugfix/[issue-name]’ so that anyone can easily identify the relevant version and review it. Lastly, it is to regularly push and pull changes. It’s important to ensure that you are working on the most current version available to help avoid merging conflicts.

The third source has six best practices listed. First is to make small incremental changes. Committing in small increments limits the possibility of merge conflicts and makes it easier to spot the error if there is one when merging. Second is to keep commits atomic. Each commit should change one problem. If you have two problems, you should have two separate commits rather than one that addresses multiple issues. Third is to develop using branches. One main rule is to not work in the main branch when collaborating with others. Create a new branch, test your code, then merge with the main branch. Fourth is to write descriptive commit messages, which I described in the second source. Fifth is to obtain feedback through code reviews, and sixth is to identify a branching strategy. Determining how your team will use branching and when can help improve overall workflow.

If I was making my own list, I would include the following guidelines: commit frequently with meaningful messages, set and follow naming conventions, keep commits atomic, identify a branching strategy, and draw a clear line between versions. I think that committing frequently with messages also can fit in with making small incremental changes mentioned in the third source. Setting and following naming conventions along with keeping your commits atomic allows for versions to be easily identified, which can fall under making it easy for reviewers to compare versions as mentioned in the first source. Identifying a branching strategy can help your team use it wisely and keep the project’s history clean. And lastly, drawing a clear line between versions to give feedback times more structure and kind of falls in with making versions clear for others to view. These stood out as the most important because they are all kind of merged from multiple guidelines. I just picked the best wording, then grouped the definitions. I think these articles can really make their guidelines broader and add more definitions to them, rather than making all of these super specific guidelines.

Sources

Source 1: <https://filestage.io/blog/document-version-control/>

Source 2: <https://nulab.com/learn/collaboration/document-version-control/>

Source 3: <https://about.gitlab.com/topics/version-control/version-control-best-practices/#develop-using-branches>