Aaron Camp

CSD-280

Module-3 Version Control

In software development, a proper standard process for managing Version control is essential to a successful project. Access to the previous software versions will save time when debugging. Reverting to a previous state allows you to check for bugs on older versions of the software. This can help the cause of the issue by observing what changes had been made to the software between the first version to experience the bug and the previous version. Version control tools also make collaborating on a project more manageable. Each Developer can work on their own branch of the product without affecting the state of the main branch. With version control playing such an essential role in the software development process, it is best to have guidelines in place to ensure that version control is maintained correctly.

We will look at three different sources to see what version control guidelines each recommends. Some of the guidelines will overlap but some are unique to a particular source.

The first source we will review is an article from the createbytes website. I found this source to be the most informative of the three. It highlighted guidelines on several aspects of version control, from managing branches to maintaining clear and descriptive commit messages.

The use of feature branching is the first guideline covered in the article. Feature branches are used to manage the versions of a specific feature of an application. The article suggests using the Git Flow branching model. We will cover the Git Flow Branching model in more detail while reviewing the next source.

Writing clear and concise commit notes was the second guideline on the createbytes website. Commit messages should be able to provide enough detail of the changes applied to the branch so that other developers can easily understand the changes that have been made.

Maintaining the main branch's state of continuous deployment was a guideline that was common to all three sources. Using other branches to develop new features or update existing ones will keep the code alterations from affecting deployment. Only after the features have been tested and debugged will they be merged into the main branch. Enforcing permissions that only allow a few developers to merge a branch with the main branch is another guideline that will help preserve the product's state of continuous deployment.

The toxigon,com website will be our next source. This article focuses on strategies used to organize the branches used for the ongoing development of the application. Toxigon provides two examples of branching strategies. The Git Flow strategy creates separate branches to use for development. The development branch is used to save the latest version under development. This branch will be merged with the release branch once the development and testing of the new version is complete. The feature branches are specific to a feature; each feature being updated or added will have its own branch. This branch is merged with the development branch when the new features have been tested and de-bugged. Hotfix branches are copies of the main branch used to fix issues found after deployment. These post-deployment bugs are corrected in the hotfix branch and then merged back into the main branch.

Trunk-based development only uses two types of branches: the main branch and a short-lived features branch. The short-lived feature branches are copies of the main that are used to update or add features. Once the new and updated features have been tested in this branch, they are merged back into the main.

The Daily Dev website listed maintaining continuous deployment as their first guideline. The second recommendation the site provided was documenting and tracking all of the changes made to each software version. This coincides with the guidelines provided by the createbytes website that stressed the importance of clearly stating the changes made when adding commit notes in Git.

I found the Git Flow strategy to be the best method of branch management. Features are developed in a feature branch, then work through the release and development branches until they are merged with the main. This assembly line approach would make debugging easier since the new feature is debugged in stages. After integrating them into the next branch, the features will be tested again. Having controls in place to prevent the main branch from being merged prematurely was another guideline I would follow. The importance of protecting the main branch is key to maintaining a continuously deployed product.

1. Toxigon. (2024). \*Version Control in DevOps: An All-in-One Guide for 2024\*. Retrieved from [https://toxigon.com/best-practices-for-version-control-in-azure-devops](https://toxigon.com/best-practices-for-version-control-in-azure-devops).

2. Kramer, N. (2024). \*Documentation Version Control: Best Practices 2024\*. Daily Dev Ltd. Retrieved from [https://daily.dev/blog/documentation-version-control-best-practices-2024#setting-up-a-version-control-plan](https://daily.dev/blog/documentation-version-control-best-practices-2024#setting-up-a-version-control-plan).

[Version Control in DevOps: An All-in-One Guide for 2024](https://createbytes.com/insights/version-control-in-devops-guide)

<https://toxigon.com/best-practices-for-version-control-in-azure-devops>

https://daily.dev/blog/documentation-version-control-best-practices-2024#setting-up-a-version-control-plan