Keanu Foltz

Module 3

CSD 380

Version Control Guidelines

Version control is an important part of modern software development that allows teams to manage changes in code, collaborate better, and keep track of the project and it’s development. The more complex a software project becomes, the more important it is to stick to version control guidelines. For my research, I looked at a Medium article on version control best practices, a Perforce article that gave 8 version control best practices, and a GitHub article that explained different aspects of version control. Each article had similarities and differences in the best practices that were highlighted.

The Medium article gave a good rundown on best practices for version control and real world work patterns. The best practices mentioned were:

* Choose a branching strategy and commit to it/use it consistently
* Leave clear and specific commit messages so that our team members and future self is not left in the dark
* Do pull requests and code reviews that are small, focused and discussed with the right people.
* When reviewing code make sure it is functional, readable, and that your feedback is specific and actionable.
* Automate everything that you can such as testing, code style violation checks, and continuous integration.
* Use advanced tricks such as rebasing for cleaning up commit history, interactive staging for granular commits, and git bisect, which allows us to find the exact commit that introduced bugs

The Perforce article gave 8 clear guidelines for best practices with version control. Those best practices listed were:

* Commit changes atomically, all files that belong to a task should be committed together in one operation so that the project stays consistent
* Commit files with a single purpose, which means each change should be independent when committing the files
* Write good commit messages, include the what and the why and be specific
* Don’t break builds, prevent the breaking of builds by doing complete commits, so it is easier to spread between branches.
* Do reviews before committing to a shared repository, this is so that we can get feedback on how to improve and change code as well as so the team is on the same page
* Make sure every commit is traceable, this allows for bugs to be tracked, traced and fixed more easily, in addition to providing better accountability and security.
* Follow branching best practices, such as keeping it simple, well defined branch policies, protect the mainline, and merge down/copy up. It is important to follow these best practices so that we can avoid lost updates, merge conflicts, or unintentional overwrites.
* Protect assets, meaning that we should use the right security measures to protect our repository and its assets. This can include access control, visibility of activity, and backup/failover.

The GitHub article provided a very comprehensive list of best practices as well, including:

* The repository set up should be compliant with access control, disaster recovery and failover plans
* Workflow alignment should be established so that collaboration is smoother and there is better mutual understanding on the team.
* Commit messages should be clear
* Small commits should be done often to keep development consistent and continuous
* Code should always be tested before committing
* Only completed code should be committed so the codebase works well
* Conflicts should be avoided by pulling code from upstream to make sure they work with the most recent code and large files should by broken into smaller ones.
* Use branches
* Limit repository access so that only contributors that need it can access it in the best interest of security

The articles all provided a ton of great guidelines for us to follow with quite a few similarities between them that I believe can be broken down into the following list of the most important ones:

* Commit messages should be clear and descriptive. This was a common point in all of the articles. We should do this so that the team is on the same page as far as what the change is and why it was made
* Commits should be atomic, each one should represent one change, so our software is clear and traceable
* Create branches, name them clearly, and use them strategically to isolate changes and allow for parallel development
* Keep the main branch stable and avoid broken or untested code
* Review code before merging to catch bugs and allow for better collaboration and learning
* Sync often to avoid metge conflicts
* Use access control for the security of the repository
* Automate testing and deployment for consistent and reliable delivery of code
* Make sure that changes can be undone cleanly if they are needed

References:

<https://medium.com/@paulosejithu/version-control-best-practices-for-modern-software-development-67bdfeddd74a>

<https://www.perforce.com/blog/vcs/8-version-control-best-practices>

<https://github.com/resources/articles/software-development/what-is-version-control>