Jacob Ambrose

09/01/2024

CSD 380

Version Control Guidelines

Version control in software development refers to a software or a tool that assists developers in maintaining their code. There are various forms of version control software available but typically they allow developers to make branches, merge updated code, and even revert to previous versions of an application as needed. A branch is typically used to work on a new feature or bug-resolution, code merging is performed when the branch is completed and ready to be merged into the main application. Not relating to version control, many companies maintain guidelines and external regulations such as the European Union's General Data Protection Regulation (GDPR). Regarding version control as of this time there are no national or international regulations written or expected. There are many best practices or recommended guidelines that we will explore below

For example, the organization Perforce advises that a commit should be used to break down tasks into much smaller doable items that are then pushed into the repository once the bug issue is resolved or the new feature is ready to implement. They maintain eight best practices for version control:

- commit changes atomically: All files committed together or not at all.
- commit files with a purpose not as a backup: Commits should not be a backup of your local files but should have a single purpose such as addressing a bug issue or adding a new feature.
- write good commit messages: descriptive commit messages help everyone. If addressing a bug or issue referencing the case or id and how the commit has addressed that issue can be helpful.
- don't break builds: Avoid complete commits, ensure these have passed test cases before being sent for a pull-request or review.
- review before committing to a shared repository: Maintain some form of review before merging- whether that is a pull-request or other review system.
- ensure every commit is traceable: projects should be able to pass test cases before and after commits.
- follow branching best practices: keep branching simple, give code lines to an owner, and use branches for milestones or new releases.
- protect your assets: ensure that passwords or other intellectual property are secure, once a password is online, the internet never forgets even if immediately removed.

Gitlab which uses Git for version control maintains their recommendations for version control as well. Gitlab's recommendations cover many similar topics as Perforce's recommendations.

- Make incremental, small changes: breaking new features or bug reports into smaller tasks makes the commits easier to complete and upload.

- Keep Commits atomic: Similar to Perforce not pushing the commit to the repository until the work is completed.
- Develop using branches: repository branches allow developers to make changes without affecting the code already in production.
- Write descriptive commit messages: being descriptive helps everyone in a code review when there are errors. Gitlab even suggests using descriptive changes in the imperative mood “make sign in form open in new page on submit” as opposed to “this patch makes x do y” or “I changed x to do y”.
- Obtain feedback through code reviews: code reviews ensure that code is written cleanly, bug-free, and find any additional issues.
- Identify a branch strategy: This will be organizational and project dependent but there are various approaches to branch strategies such as: centralized, feature-branching, or even personal branching.

Adekola Olawale with free code camp also discusses version control best practices. While Olawale focuses mostly on git I believe these can mostly apply to all version control guidelines. Olawale advises to:

- Keep commits small and focused: “fix broken link in footer” is better than “fix broken link in footer, removed unused imports & fixed login button alignment issue” as that can create confusion if future errors occur with any of those three items.
- Clear and concise commit messages
- Branch frequently to isolate changes.
- Use pull requests for code reviews: this allows any errors or issues to be caught prior to merging the feature branch with the main branch.
- Keep the Repository Clean and up-to-date.
- Avoid committing unnecessary files.

Each of these resources had similar guidelines which can be summarized as keeping the branch or commit to a specific feature or issue to be resolved, writing clear and concise messages, pushing commits after the feature is completed or bug resolved, and lastly breaking tasks down into smaller easier to complete tasks. While each of these resources come from different years and different goals I feel these are a good start for an organization to build from to tailor to their specific needs. To compare and contrast all of these guidelines I would have to follow a quote often attributed to Leonardo DaVinci “Simplicity is the ultimate sophistication.” By focusing on the following guidelines organizations can add guidelines to suit their needs.

- Keep commits small and focused.
- Develop using branches.
- Write descriptive commit messages.
- Keep Commits atomic.
- Make incremental, small changes.

Works Cited

- Adekola.olawale. (2023, May 16). *Git best practices – A guide to version control for Beginners*. freeCodeCamp.org. https://www.freecodecamp.org/news/how-to-use-git-best-practices-for-beginners/
- GitLab. (n.d.). *What are git version control best practices?* https://about.gitlab.com/topics/version-control/version-control-best-practices/
- Schiestl, B. (n.d.). *8 version control best practices*. Perforce Software. https://www.perforce.com/blog/vcs/8-version-control-best-practices