7-2 Final Project:

Release Notes Report

Brandon Ricks

CS-310

Nick Schnack, MCS

Southern New Hampshire University

06/19/21

# Contents

[Contents 1](#_Toc74985475)

[Section 1: Overview 2](#_Toc74985476)

[Version Control Summary 2](#_Toc74985477)

[Section 2: Development 3](#_Toc74985478)

[Pushing Local Repositories 3](#_Toc74985479)

[Branching and Merging 4](#_Toc74985480)

[States of a Source File 5](#_Toc74985481)

[Improvements & Enhancements 5](#_Toc74985482)

[Performing Code Reviews 5](#_Toc74985483)

[Review Feedback 6](#_Toc74985484)

[Issue Resolution 6](#_Toc74985485)

[History & Newsfeed Tools 7](#_Toc74985486)

[Due Diligence 7](#_Toc74985487)

[Distribution, Deployment, & Troubleshooting Tactics & Strategies 8](#_Toc74985488)

[Valuable Features 8](#_Toc74985489)

[Workflow Strategies 8](#_Toc74985490)

[Best Practices 9](#_Toc74985491)

[Version Control Tools & Calculator App 9](#_Toc74985492)

[Version Control Tools & Jukebox Playlist App 9](#_Toc74985493)

[Version Control Technique 10](#_Toc74985494)

[References 11](#_Toc74985495)

# Section 1: Overview

## Version Control Summary

As defined by Atlassian, a software development company that specializes in software for development and project management, version control is “the practice of tracking and managing changes to software code” (Atlassian, 2021a). Version control software (VCS) is often an essential part of a modern software development team’s practices, allowing teams to manage source code and software development over time with minimal disruption to the team (Atlassian, 2021a).

Version control offers many benefits in software development projects. As development progresses through the lifecycle, and as team members make modifications, changes, fixes, and implement new features, version control software allows teams to keep track of prior versions, branch off existing source code, and merge code back into the main branch for others to use (*Why Use Version Control?*, 2013). The freedom for team members to work in tandem on any file or code at any time offers a lot more flexibility and a more streamlined development process. VCS also allows the team to keep track of prior versions for backing up work, or reverting to a previous version if needed. This mitigates risk to the project in case something goes wrong (*Why Use Version Control?*, 2013).

Version control helped me to create and manage branches of code as I would make changes to the parent branch. Branching off to implement a new feature allowed me to work on new code without any harm to the existing main branch in case something doesn’t work as intended (Somasundaram, 2013, pg. 126). I was able to code, test, and then commit this feature branch to the master through Git Staging, adding comments to dictate what changes were made. This would all show within the History View for others to see on the project as well.

This process made adding code to the program safe, and also made the process of merging this code into the main branch and pushing this branch to a repository easier. As a team would work on bug fixes, new features, etc., the ability to branch off existing code and commit changes with commented information creates an efficient way for a team to work in unison on a project, and keep one another in the loop on what is taking place at any given time (Somasundaram, 2013, pg. 62). This also ensures that anyone working on the project can have the most recent version available to them quickly, and if needed, the ability to revert back to a previous version (Somasundaram, 2013, pg. 8).

# Section 2: Development

## Pushing Local Repositories

Development with a team often requires collaborative effort. For developers to work together on the same project, access to a remote repository is a valuable tool. This allows developers to clone the repository to work on locally so work can be done in collaboration, but without developers disrupting the main branch (*Introduction to Remote Repositories*, 2021). As changes are made to a local repository, they can then be committed and pushed back into the shared repository for others to use, and to share work with one another through development. This is a more centralized workflow, where “a central repository serves as a single point of entry” (Atlassian, 2021b). A logical extension of this type of workflow is feature branching (Atlassian, 2021b). This incorporates the idea that feature development should take place on a separate, dedicated branch, rather than the master branch (Atlassian, 2021b). This concept ties into continuous integration, where the master branch should never have broken code, which is very advantageous in CI environments (Atlassian, 2021b).

All of the options available in the workflow can enhance and expand on the functionality of repositories. Rather than a place to just upload code, it can be expanded on to be a vital resource in development, especially in collaboration with other developers (Somasundaram, 2013, pg. 62). Cloning and branching also allows a branch to be merged back into the main branch easily, ensuring that changes made on local repositories and branches work fluidly when integrated back into the main branch and the remote repository (Somasundaram, 2013, pg. 63).

## Branching and Merging

Branching and merging are valuable tools in VCS, and through development. The ability to branch work away from a main branch, work on tasks, and then merging these changes back in is effective in organizing work within a team (Schiestl, 2020).

Branching is, effectively, copying a codeline so that a developer may work on features or bugs while others can work in tandem, and allows changes to happen isolated from the main branch (Schiestl, 2020). A stash is temporarily storing changes that have been made to a working copy (Atlassian, 2021c). Stashed work can then be returned to at a later time. Merging and rebasing add changes into the mainline code, but do so in different ways: merging merges branches into another branch and the target branch is altered, while rebase integrates changes from one branch to another, encompassing all changes into the target branch and flattening the history (Perforce Software, 2018).

In some cases, if one hasn’t merged in some time, and as others are also merging changes, there is potential for merge conflicts. These conflicts are a result of when changes being integrated conflict with one another, such as the lines of code in the same file being changed and conflicting (Lubański, 2019). These conflicts need to be resolved before one can finish merging a branch.

## States of a Source File

Files go through many states in a file status lifecycle. Untracked files are present locally, and not in a repository or under version control, whereas tracked files are the opposite (*The Workflow of Version Control,* 2021).

Tracked files can be unmodified, modified, or staged. An unmodified file is one that has had no changes since its last commit (*The Workflow of Version Control,* 2021). New files that will be included in the next commit move directly from untracked to staged. A modified file is one that has been changed locally since its last commit, but has yet to be committed. These files will appear when the next commit takes place (*The Workflow of Version Control,* 2021). Staged files are either new files that aren’t present from the last commit (untracked until the next commit), or modified files that will be committed (*The Workflow of Version Control,* 2021). These files are ready to commit and push to a repository.

# Improvements & Enhancements

## Performing Code Reviews

Code reviews are a crucial part of team development work. When a pull request is created for a branch, others on the team are given an opportunity to help and inform each other, while also making sure the code is functional and ready to be merged into the main branch. For example, in the project for the Jukebox Playlist App, I reviewed a team member’s code and informed them of a possible issue with a variable name, and I also made sure to inform team members of great work they had accomplished. In doing these reviews, it reminded me to double check my own work, and consider following advice or information I may be giving to others, as well as implementing the advice given to me.

## Review Feedback

Receiving feedback can inform oneself on where improvement can be found, or where a possible issue may need resolved. In feedback I received, most of the information given was that my code was neat and well organized, which I appreciated. I did receive feedback on spacing, which was a result of how the code looked in the pull request, and not actually how it appeared. However, the feedback did have me look over my own code for any possible excess whitespace, or and areas where the code could be further cleaned up for readability and functionality.

## Issue Resolution

An example of a situation in which a team can use code reviews to resolve an issue would be to ensure code is clean and functioning properly. If code won’t run, or tests fail, prior to a pull request, it may be better to resolve these issue prior to the pull, but the rest of the team could help to both pinpoint possible problems, as well as possible resolutions, through the pull request.

Another example is to pinpoint and fix any vulnerabilities or bugs in the code. Small coding errors can result in major vulnerabilities, non-functioning code, or security issues, so taking the time to check through these reviews are a great practice to apply in a team (kexugit, 2019). With each code review is an opportunity to improve and learn from the feedback given by your peers.

## History & Newsfeed Tools

The history tool allows a developer to quickly and easily see what work is being done, and what work has completed. Seeing each branch, committed branches, and messages linked to each commit, gives a lot of insight into what a branch’s purpose may be, and what has been done with the branch thus far.

The newsfeed tool offers real-time updates and messages regarding branch changes or updates. If there is a comment in the pull request, or changes to a subscribed repository, these notifications allow a developer to be on top of any changes taking place, and are a very useful tool.

## Due Diligence

My due diligence in preparing code prior to review is similar to my due diligence in writing code. I write in a sort of “measure twice, cut once” method, where I do a lot of testing and checking prior to moving forward. My preparation method is to check my code’s functionality, and run any JUnit tests to ensure the code is both functional and passes testing. I go through to check for neatness, readability, and check my comments to make sure they are proper and informative. Once I feel it is acceptable, I make any final commits and push the branch. If changes are needed, I update my branch with the newest main branch, and make said changes. Then I go through the process again, before pushing the altered branch.

# Distribution, Deployment, & Troubleshooting Tactics & Strategies

## Valuable Features

One valuable feature I find particularly useful is access to EGit natively within the IDE. This, combined with a remote repository like BitBucket, adds many new layers to the possibilities in both personal individual coding, as well as team focused work. Having this access within the IDE streamlines the process of checking out branches, pushing local branches, committing work, etc., and makes team collaboration easier.

As you update origin branches from the repository, you can have a direct access to it, and are able to see commit messages, history, and all other tools necessary to keep up to date on the progress of a project. You are also able to push branches directly to the repository, and set it up so that the process is as simple as a few button clicks. Much of the information on the project can also be accessed on the repository, but it is very handy to have all of this directly in the IDE.

## Workflow Strategies

In individual projects, I would be the only one who had access to the code, so I didn’t have the opportunity to have others assess my work or add to it. Nevertheless, I would do my due diligence with code structure, ease of readability, and I would ensure my code is functional. I would still add comments as well, even if just to inform myself of what code does, or my intentions with it, in case I needed to come back to it later.

Working as a team alters a workflow to consider other developers also interacting with your work. While much of the steps are the same when working with a team, some considerations are given even more importance, such as commenting code and readability.

## Best Practices

As shown in *Version Control Best Practices* (2020), there are many fundamental rules and considerations to follow when working in version control, especially when working on a team with others. Every best practice is important, but one that I find especially effective is to not commit half-done work. Throughout Jukebox Playlist app we all worked on in tandem, there were dozens of open branches at any given time. With this constant movement of new branches, pull requests, and mergers, its important to make sure what is in the repository is functional, and perhaps even ready for pull request.

Branches that aren’t functional or only partially completed may accidentally be used or seen as complete, and I believe a repository should stay neat and organized. If work could not be completed on a particular branch, a good practice is to break the work down further into logical chunks, commit work that is complete, and consider stashing unfinished work to be finished at a later date.

## Version Control Tools & Calculator App

In the process of developing the Calculator app, I was able to resolve an issue with mistakes I had made in a branch merger. By using history, as well as making sure to branch off of the main branch when making changes, I was able to revert a branch back, and fix the issue. Using history, as well as keeping branches separate, helps to avoid possible mistakes that are irreversible or otherwise more difficult to fix.

## Version Control Tools & Jukebox Playlist App

In the process of developing the Jukebox Playlist app, I ran into an issue with a branch disconnecting from the main branch. This resulted in me being unable to create a pull request, as my branch now no longer had a parent branch. One way in which version control tools helped in resolving this issue was the ability to check out the newest main branch as a local branch. By doing this, I was able to combine my branch and its changes into the main branch. This newly combined branch was connected to the main branch as a feature branch, and also incorporated the changes I had made.

## Version Control Technique

In both of my development apps, one technique I will make great use of moving forward is descriptive commit messages. I value good comments in code to describe the code’s functionality and purpose, and I see good commit message as expanding on this practice. This, combined with a well-coordinated team, can really help to make sure everyone is on the same page.

When working alone, it is still good practice to use messages, especially if you consider that maybe down the line someone may interact with it, but as I worked with others in a development environment, I learned that its even more important to convey your intentions and leave descriptive messages for others.

# References

Atlassian. (2021a). *What is version control | Atlassian Git Tutorial*. Atlassian. Retrieved from <https://www.atlassian.com/git/tutorials/what-is-version-control>

Atlassian. (2021b). *Git Workflow | Atlassian Git Tutorial*. Atlassian. Retrieved from <https://www.atlassian.com/git/tutorials/comparing-workflows>

Atlassian. (2021c). *git stash - Saving Changes | Atlassian Git Tutorial*. Atlassian. Retrieved from <https://www.atlassian.com/git/tutorials/saving-changes/git-stash>

Atlassian. (2021d). *Merging vs. Rebasing | Atlassian Git Tutorial*. Atlassian. Retrieved from <https://www.atlassian.com/git/tutorials/merging-vs-rebasing>

*Introduction to Remote Repositories*. (2021). Git-Tower.com. Retrieved from <https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction/>

kexugit. (2019, October 2). *Code Reviews: Find and Fix Vulnerabilities Before Your App Ships*. Microsoft.com. <https://docs.microsoft.com/en-us/archive/msdn-magazine/2007/november/code-reviews-find-and-fix-vulnerabilities-before-your-app-ships>

Lubański, M. (2019, June 6). *How do you handle conflicts in your Version Control System?* Medium; FAUN. Retrieved from <https://faun.pub/how-do-you-handle-conflicts-in-your-version-control-system-6522e8ef77ef?gi=606dcd98f18e>

Perforce Software. (2018). *Git Rebase vs. Merge: Which Is Better?*. Perforce Software. Retrieved from <https://www.perforce.com/blog/vcs/git-rebase-vs-merge-which-better>

Schiestl, B. (2020). *Branching Definition — What Is a Branch? | Perforce Software*. Perforce Software. Retrieved from <https://www.perforce.com/blog/vcs/branching-definition-what-branch>

Somasundaram, R., & Ebooks Corporation. (2013). Git : Version Control for Everyone: the Non-coder’s Guide for Everyday Version Control for Increased Efficiency and Productivity: Beginner’s Guide. Packt Publishing. Retrieved from <https://search-ebscohost-com.ezproxy.snhu.edu/login.aspx?direct=true&db=nlebk&AN=533211&site=eds-live&scope=site>

*The Workflow of Version Control*. (2021). Git-Tower.com. Retrieved from <https://www.git-tower.com/learn/cheat-sheets/vcs-workflow/>

*Version Control Best Practices*. (2020). Git-Tower.com. <https://www.git-tower.com/blog/version-control-best-practices/>

*Why Use Version Control?* (2013). Git-Tower.com. Retrieved from <https://www.git-tower.com/learn/git/ebook/en/command-line/basics/why-use-version-control/>