**MET CS 633 ASSIGNMENT – BRANCHING\_MERGING**

|  |
| --- |
| Consider the sequence of ten steps, while collaborating with others to develop software, |
| 1. create your own branch |
| 1. execute commits - make changes on your branch |
| 1. execute unit test on your branch |
| 1. conduct peer review of changes on your branch |
| 1. make additional changes on your branch |
| 1. perform rebase from master into your branch |
| 1. resolve any conflicts that might occur from rebase |
| 1. attempt performing an auto-merge from branch into master |
| 1. if auto-merge does not go thru, perform manual merge |
| 1. resolve any conflicts that might occur from merge |
| **Write few lines** responding to each related question. |

1. What risks can you envision, if the rebase of step (6) is skipped? What potential issues can result in a rebase?

The rebase command in Git merges modifications from one branch into the other by rewriting the commit history, resulting in a straight, linear sequence of commits. This will fetch the most recent information from origin, then base your branch on the most recent version of master. This eliminates the need for a dozen fresh merges from master commits every time you want the most recent modifications to master in your current branch. As a result, if this step is skipped, depending on how long a developer has worked on their private branch, the version of the main code base he used when he started may have changed significantly, and if he doesn't rebase, he will be working with stale code. The branch does not have the most recent master commits. Also, without rebasing, there's a risk that no coherent sequence of modifications may emerge.

Reference:

Brindescu, C. (2020). An Investigation of the Effects of Merge Conflicts on Collaborative Software Development.

1. Why it is important to stay on your own branch and not to commit into branches of other teams?

This is because when you commit to another branch, the changes made in your branch may affect the other branches of the other teams it is safer to merge the branches after the changes are made. Merging is usually done when work on a particular branch is done, and this makes it possible to implement the changes onto the main branch. When one commits to other branches it means you will bring along the changes made onto the other team's branch. This will mean that the changes will have to be implemented along with that other team, meaning there will be an impact on the other team. This may cause problems and confusion to the other team.

Reference:

Wang, X., & Yang, C. (2017). Merging-branches impact on decision tree induction. CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-, 30(8), 1251.

1. What risks do you see when executing commits directly into master instead of your branch?

It's terrible to use master as a dev branch instead of a private branch. It's possible that someone will break master and have an impact on everyone. Undoing this procedure after disagreements can be a time-consuming task that requires a lot of work. Master is the primary branch, which may be used in production, and others will continue to merge from it, thus it's best to avoid making direct changes to it. The master should always be stable. Feature modifications will be moved to a new branch, where they will be tested, stabilized, and reviewed before being merged into the master. We should not tamper with the master's domain. If we make modifications to the master branch of a collaborative project while others are working on it, those changes will swiftly spread to everyone else, resulting in merge conflicts, new problems, and locust plagues. That's how serious it is. The master branch is supposed to be stable, and it's part of the open-source software social contract to never, ever push anything to master that hasn't been thoroughly tested or that breaks the build.