# Overview of the Assignment:

Please analyze the described GIT process and respond to the following six questions.

*Be sure to submit your completed assignment via Blackboard using the Assignment tool.*

|  |
| --- |
| 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. |
| Question 1. What risks do you see when executing commits directly into master instead of your branch?   1. The code that imports directly into the master branch is missing the unit test and peer view in the branch, so it is likely that there are bugs and mistakes that could have been avoided. When such a code goes directly into the master branch, which is usually already complete, it will most likely pollute the pool. 2. The personal branch ensures the isolation of individual code, ensuring that each programmer's code does not affect others. Directly committing to the master branch, even if there are no errors in the code itself, may still lead to incompatibility with the existing complete code or other people's code. 3. Committing directly to the master branch disrupts the strict steps (branch-master) that need to be followed, potentially causing issues when tracking and reviewing the history.   Question 2. Why it is important to stay on your own branch; not to commit into branches of other teams?   1. Using your own branch clearly helps avoid both the risks mentioned in Question 1 and other potential risks that haven't been explicitly addressed. 2. From a positive perspective, at the team collaboration level, it allows each person to focus solely on their own code without needing to worry about additional factors, thus making teamwork more efficient. At the project level, it avoids conflicts between different pieces of code, ensuring code isolation and making the process easier to maintain.   Question 3. What risks can you envision, if the rebase of step (6) is skipped?  If step (6) is skipped, that means executing merge without rebase.   1. May lead to severe merge conflicts, which can disrupt the project timeline and impact progress. 2. Rebase in Git provides a visual result that presents the project in a clear, structured manner. Skipping this step can make project management, tracking, and reviewing more difficult, as the commit history may become disorganized, with unrelated changes intertwined, making it harder to trace the evolution of the project. 3. Code that lacks rebase is highly likely to conflict with subsequent updates in the master branch.   Question 4. What potential issues can result in a rebase?  Rebasing step may results in data loss and conflicts. When merging commits onto a new base, there is a possibility of data loss during the transition, or conflicts arising with the master branch. This can occur if there are changes in the master that are incompatible with the commits being merged, leading to potential overwrites or the need for complex conflict resolution. |
|  |
| Question 5.  **Follow the tutorial** for an introduction to GitHub. https://guides.github.com/activities/hello-world/  First, create your personal account at GitHub. Then, make sure your facilitator / instructor is invited  to the repo through these commands, <Settings>, <Manage Access>, <Invite Collaborator>.  @elentukh . @Srinikhil0 .  For you to receive a full grade, your instructor needs to confirm these activities,   * Create a Repo * Create a Branch * Make a Commit * Open a Pull Request * Merge Pull Request   Question 6.  Students of the previous class developed the Git Repo Analysis Tool recommended for a team project. As an introduction to this tool, please access it at Firefox gitrepoanalysis.onrender.com/, then enter the URL with your personal repo and attach the resulting report to the submission of this assignment. You are encouraged to run the same report for your final presentation of team project. A similar app was developed by students and can be used on your project - github.com/ed239/GitScanner  Additional Notes   * People interpret the notion of 'collaboration' quite differently. It would not be a surprise to read about 'holding hands and singing cheerful tunes' as a 'collaboration'. In a context of this class, we delve into such technical tasks as 'rebase' and 'merge' as key parts of 'collaboration'. In GitHub, members of a development team, respond to Pull Requests with comments and approval of a code change. In module 5, when covering Continuous Delivery, most 'collaboration' steps are subsumed by an extensive regression test. * *Rebase* is an important concept in source control. During second module, we talk about *rebase* that can mess up your revision history. Here is the link to Git online book.   <https://git-scm.com/book/en/v2/Git-Branching-Rebasing>  This is a truly comprehensive collections of all kinds scenarios. Note that in other source  control systems, e.g. Clear Case, commands are different, although concepts remain the same.   * In this class, we prefer relying on a commonly used standard or a tradition. Brian Kernighan   starts his classic 1978 book titled "C - Programming Language" with a tutorial - how to printf "Hello World". Here we are a half a century later. See below the screenshot from GitHub tutorial, estimated as 10 minutes read. |

