Ryan Nguyen

Git Workflow Essay

As we know, Github is a very flexible database that developers and DevOps teams are using to communicate and work together in the Computer Science field. As flexible as it is, this makes Github a very confusing system because of how vast it could be used. Git does not provide a definite process of how teams should be managing and accessing their database. Git workflow helps users have a basic standard for how to use Git in a more efficient and consistent way. With an agreed Git workflow, the development team should be able to work more on the same level to understand what each person’s job is. Github’s main functions include commits, pushes, pulls, merge, and brancing.

After creating a local branch from the central repository, commits can be made to add your content or changes to be added. When you commit, you are trying to edit the files in the repository and commit changes. These commits aren’t directly affecting the main repository. They are isolated locally instead of messing up the central repository’s programs. To make changes to the central repository after you commit changes to the local repository you created, you would need to push the changes after someone on your team reviews the changes you have made. When a push is made, it directly changes the main repository’s code that you were working on. Conflicts might occur when pushing your content to the central repository because a team member may have code already pushed that conflict with your code causing a merge that would inflict conflicts. Instead of allowing conflicts to happen without notice, Git will output a message telling the user the conflict. When conflicts arise, you will need to use the Git pull command. Pulling from the central repository, you are allowed to use your other team member’s update to integrate them with your own local changes to fix the conflicted code. After fixing the code you are able to use a function called rebase, which allows you to commit all the changes that you have fixed, to the tip of the main branch.

A successful Git workflow does not limit productivity. When looking at a Git workflow to see if it works, the size of the team matters. Being able to undo mistakes and errors in this workflow is important as well because if one person on the team makes a mistake, it should take long for the team to fix said mistake. It ruins efficiency when a team takes too much time on a mistake because it could effect other members of the same team working on that specific code. A solid workflow means that the whole team understands and could go through their work without having to think too hard about it.

The Centralized Workflow is a good workflow to start with especially when you previously were a team that used Subversion (SVN). Like SVN there is a central repository, but instead of it being called trunk, Git uses the default development branch called main. This workflow doesn’t require any other branch because all changes in the Centralized Workflow are committed to the main branch. Subversion’s development of projects is the same way as Git, but the only difference is that Git has a few advantages compared to SVN.

Firstly, when developing a project each member of your team will get a copy of the entire project. This gives an environment that allows for individual progression towards the app for more efficiency. After they are done working on their own part of the project, all they have to do is add commits to their local repository. Secondly, having many different individual sections of the project being committed allows for Git’s variability to branch and merge models. Git branches are made to be able to control mistakes when integrating code between repositories. The Centralized Workflow is generally a good workflow when using Git when you have came from another database because of its similar remove server-side repository.

Sources:

<https://www.atlassian.com/git/tutorials/comparing-workflows#:~:text=A%20Git%20workflow%20is%20a,in%20how%20users%20manage%20changes>.