Crystal Luong

Git is a distributed version control system that allows users to observe and review previous versions of code shared among a team. It is especially used in a team setting to help facilitate workflow and improve productivity. The functions Git offers are creating a repository, cloning the repository to a local machine, pushing or pulling changes to the file in the repository, publicizing the repository, collaborating with other users, and other features. Git workflows are recommended to properly make full use of Git in a team of developers. The Centralized Workflow is a version that uses a central repository where all team members commit changes to the code from a single entry point. For those who have used Subversion (SVN), this is a good transition to Git workflow since Subversion is a centralized version control system. Git offers similar features to SVN so transitioning to Git is not confusing. The Centralized Workflow has a 'main' default development branch in which all changes will be committed. Since it is centralized like SVN, it does not have any other branches besides the 'main' branch. Unlike SVN, Git provides a copy of the entire project onto the local machine. That means developers can work independently on the code and not step on the toes of their teammates. They can make changes to the code without having to save the changes to the 'main' branch in the shared repository. In addition to the isolated work environment, Git offers a seamless method to share and integrate any changes to the code between repositories. Like other types of Git workflows, this permits users to push and pull code from a repository. The difference it has against other workflows is that there is no concrete pull request or forking layout. Hence, it is recommended for people who worked with SVN and for a smaller size team. The workflows work by creating a central repository that does not have a working directory and it can be a new, empty repository or an existing Git or SVN repository. The command to make a bare repository is 'ssh user@host git init --bare /path/to/repo.git'. The 'user' is the SSH username and the 'host' is the domain or IP address of the server. Bitbucket Cloud is a 3rd party Git hosting service that creates a repository. Developers start by cloning the central repository using the command 'git clone ssh://user@host/path/to/repo.git'. After cloning, Git adds an 'origin', which is a shortcut to the parent repository. The central repository has the project that the team will be working on and by cloning, each person will have their copy of it on their local machine. Each member can edit the project and save the changes while being isolated from the shared central repository. Once a person wants to share their changes, they can push it on their local 'main' branch to the central repository. All the changes from the local 'main' branch will be pushed to the central 'main' branch similar to when previous users of SVN use 'svn commit'. Some commands used to edit, stage, and commit locally are 'git status', 'git add', and 'git commit'. Staging provides a way to prepare to commit without committing every change to the working directory. To push and share the changes to the centralized repository, the command 'git push origin main' will push any changes to it. The 'origin' indicates the remote connection to the central repository and the 'main' argument tells Git to make the original's 'main' branch look like the local machine's 'main' branch. The issue with pushing the commits to the original repository is that some of the changes may diverge from it, stopping any commits that will overwrite official commits from other developers. If Git notifies the user of the conflict, the command 'git pull' can help with the push conflict since it fetches the centralized repository's commits. If the local machine's commits conflict with the centralized repository's commit, Git allows the user to manually resolve the conflicts and rebase the local machine's 'main' branch with the upstream's commits. Centralized workflow is only one of the many popular Git workflows. Other examples include Forking Workflow and Feature Branching. Forking Workflow ensures that developers have two Git repositories, one as their private local machine repository and a public server-side one. Feature

Crystal Luong

Branching is an extension of the Centralized Workflow as it has a separate branch that is dedicated to any feature commits instead of using the ‘main’ branch. That way developers can work on certain aspects without changing the main base code.