Ivan Wang

Git Hub Assignment

Git is a version control system that allows multiple people to work on a project at a time. Its main feature is that changes to files are recorded over time so that any version can be found later. A git workflow is a system that a team would adopt to ensure uniformity when handling the project. The article by Atlassian "Comparing Git workflows: What you should know" talks about a popular way to utilize Git, called the "Centralized workflow". This specific workflow is popular because of its straightforward nature. The Centralized Workflow is based on a central repository where all changes made to the project will be tracked.

In a centralized workflow, each project is centralized on one remote repository (repo), or storage. With this central repo, each member can *clone* it and make their own local copy on their machine. Each member in this workflow can edit their own local copy and their edits to the clone will be tracked. When they're ready to save their changes, they can *commit* their changes to their local repository, essentially saving their work. Like saving a text file, each member can commit many times, but every commit is added to the commit history, which has the complete version history.

Committing their changes only affects their local version though, and when a team is ready to share their changes with the team, they can *push* their commits. Pushing their commits posts all their changes along with their commit history to the shared central repository. The central repository must have a linear history, meaning that each commit follows another in a straight line, like links in a chain. This linearity allows each version to be easily traced back to its origin and makes the progression of the entire project clearer. To maintain linearity, many teams will use rebasing, which is taking different commits and making those changes to one branch, effectively making one line of commits.

If other members push their changes to the central repo, that means everyone else's local repository is now outdated. To fetch these new updates, they must *pull* the latest changes from the central repo, which is the opposite of pushing. Pulling gets any updates from the central repository and brings them to the local repository so that any further updates on your local repository are based on the central project.

Sometimes when pulling changes, there might be a situation where it is not clear how to combine the incoming changed with the local ones. This is called a *merge conflict* which happens when two people make different changes to the same file. Since we need to preserve linearity, conflicts need to be resolved before any commits can be made. Conflicts are resolved by picking which changes to keep and which changes to discard by hand in the local repository. Conflicts can be mostly minimized with regular communications and frequent pulling.

A centralized workflow is very simple and convenient, but it does have drawbacks. One notable downside is when multiple people try to push to a central repository at the same time. If multiple people try to push their changes at the same time, one will have to wait for the other to finish. This can cause delays, especially when the team of developers is large. The more contributors that are on one project, the higher the chance is that some of them will try to modify the same part of a file. If they both try to push, it will lead to a merge conflict and they will

become more frequent with more people on a project. Since everyone on a centralized workflow relies on the central repository, if the repository becomes unreachable, the entire team will have to wait before any changes can be pushed or pulled. The centralized workflow in Git offers a simple way to control versions of a project. However, while its streamlined nature can foster collaboration, it also can lead to potential bottlenecks, especially in larger teams.