Alexis Lopez

Git is one of the most commonly used version control system, a software that allows users to track, edit, and compare changes over time. Since git is a very effective software, a variety of git workflows are out there recommending users on how to effectively and efficiently get work done. Although they are useful, a team needs to agree upon a workflow before any projects in order to maximize productivity. Git workflows are not set in stone though, so teams are free to implement rules from a variety of workflows to suit their needs or preferences. There are some rules that should be followed though. Does the workflow scale with team size? Is it simple to correct mistakes or errors? Does it provide unnecessary cognitive overhead to the team? One workflow that I have used in the past is the branch workflow. This is where specific features get branches dedicated to just one branch. As a result, code that would result in errors never reach the main branch.

In order to make changes to the main code from your local repository, you can use the command **git push <remote> <branch>.** Once there are changes made to your local repository, you use the command in order to share the code you created with the rest of the team, provided that there are no errors or merge conflicts with your code. For example, if i was working on a part of the code where I would have to make methods for a calculator. Once i make those new files and changes, my team cannot access these files. This is because i have not updated the main repository with my changes. So I use the command **git push origin main** in order to do so.

Once those changes have been made, another team member can use the pull command in order to update their local repository to a newer version. To execute this command, the user must type in the command **git pull.** This command is a combination of two other commands used in git, **git fetch** and **git merge**. The first part of the git pull uses the **git fetch** command, where it where download the contents where it is pointed at. After that command finishes running, it will go onto the the second part, the **git merge** command. The command will create a new merge commit and the HEAD will be updated to the new point at the new commit. So once my team member has made changes to the main repo, i can execute this command in order to update my local repository to the newer version. I can then access these files on my own repository and continue working on any other changes or update that needs to be made to the project

Going further into the **git merge** command, it is git’s way of putting a forked history back together again. It takes independent lines of development created by branches and combines them into one. If two developers are working on the same content, it is possible for them to encounter a merge conflict when they try to merge into the main repo. This is why it is recommended for developers to work on separate isolated branches. So if another developer and I were trying to fix an issue with an instance of code. If we both merge our changes into the main repository, then we will encounter a merge conflict.

A repository in this instance is a virtual folder containing all of the code, files, and extras used in the creation of a project. Developers can access this repository and make any changes that is required for making a successful project. When commands such as **git pull**, **git push**, or **git merge** are executed. They access this repository and download or update the files stored in the repository.