Cachary Tolentino

Git is a popular open-source version control system, mainly used for software development. It is a very flexible tool for managing small to large projects that require precise control over project versions. Git alone would not be useful without having a repository. A repository is a centralized container for any and all changes and files within a project. Git and a repository work similarly to a filing system. For example, an employee may write a paper regarding business expenses, this paper could be considered as a file or code in a development environment. Once the file is finished and fully revised, the employee may move this file to a folder and then to a cabinet. In this scenario, the cabinet can be considered as the repository, the main container for all files or code. And Git acts as the tool or system of actions that process the file into reaching the repository.

Some basic commands that are useful in knowing when working with Git are: commit, push, and pull. Commit is essentially a recorded state of the current working file within a project. In comparison to the earlier example, when the paper is finished and moved into a folder, this is the same as executing a commit for a code. The commit is saving the current state of a file or code. Push on the other hand is the action of moving these saved files or code into the repository. In our example, this would be the action of moving the folder containing the file into the cabinet. When making a push, especially if it is directly to the repository, if there are other folders within the repository, this action would be considered as a merge. Merges are when separate works/pushes are being combined into the main repository. For our example, this would be emulated through combining the contents of other folders into the cabinet. Although, sometimes there is a possibility that this could lead to a merge conflict. Merge conflicts are when files or code have different changes that are not aligned with each other. For example, when merging the files for business expense, the already stored travel expense file could have data or information that has more up to date information compared to the business expense file. This would cause a merge conflict. The only way to fix this is through resolving the conflict, or updating the business expense information. Lastly, a pull is the action of updating or retrieving any new files or code from the repository that a local repository does not have. In our example, if there was another employee that made another file for traveling expenses and pushed it into the repository (cabinet), the pull command would be making a copy of that new file and moving it to our local repository (desk of the employee).

For users to fully utilize Git to its fullest potential, a Git workflow is advised. A Git workflow is a streamlined process of actions in which Git is used for a collaborative team of developers or users in a development environment. Essentially, it's an agreed upon process of actions on how to interact with Git in consideration of the entire team of developers. This not only promotes productivity and efficiency, but also promotes team collaboration and culture. Although, Git workflow is not necessary, it is useful when considering the scalability of a project, handling of conflicts, and overall development consistency.

An example Git workflow guideline is the Centralized workflow. In this workflow, all work revolves around the central repository. The first step in this workflow is to create either a new hosted central repository or import an existing project to the repository. Once this is finished, all members of a team working on the project would clone the repository (making a copy) to their local repository (local device). For a member, they can simply start following the traditional commands with Git during their development process such as commit and stage (a step before commit to select only the desired files or code to be committed). Once a member is fully finalized with their work, they would simply push their changes into the central repository. Any conflicts during the push can be resolved. If there are any changes made in the central repository, members can simply pull these changes.

The centralized workflow is only one of many types of workflow. Some others are more specialized depending on the use case for any development team. Some examples would be the feature branching workflow in which branches (or sections of a work) are made to isolate and focus on certain parts of a project/software.Others would be the Gitflow workflow in which it is very similar to feature branching, however, each branch have their assigned role and a guideline on how each branch would interact with one another. Last example would be the forking workflow in which any member in a team would have two copies of the repository, the hosted repository and a local repository. These are simply guidelines for development teams. Each team will differ in their needs and some may need to consider more than others. Some considerations to keep in mind are that any branches should be kept short-lived to minimize conflicts, a workflow that benefits the team when in need of reverting any changes, and a workflow that matches their needs such as one that can follow their release cycle. Overall, these types of workflows are a starting point for any development team seeking to further improve their workflow and general development experience.