Name: Raghvandersinh Solanki

**Git Workflow and Functionalities**

Git workflow purpose is to show how to use Git in a productive and consistent manner to accomplish work. It encourages developers to leverage Git effectively and consistently. Centralized Workflow is a type of Git Workflow, where teams use a singular point of entry for all changes for a project. The branch is called **main and all changes are committed to this branch, no other branches are created, think of the main branch as the trunk of the tree you can either grow(with commits) or cut down(by deleting the branch) if you not satisfy with the result. Commit** is snap shots of changes made to the working directory, **think of it as a check point in video games, but you can make those checkpoint anytime**. When you make a commit, you need to **write a very detailed message of changes you made,** to make sure you won’t forget the changes. **Branches** are pointer to your commit or commits. Centralized Workflow are usually used in small teams or people who just came from Subversion.

Imagine that you had an idea, but you don’t want to affect the main branch, you can create a **separate branch (I like to call the side branch)**, you can either delete the side branch or you can **merge it with the main branch.** Merging side branch to the main branch can cause conflicts. **“Git fails to start the merge”** happens If there are changes in the main branch happens and you try to merge your side branch. **“Git fails during the merge”** means that there is an issue between the local branch and the branch being merged.

**Central repository (The core that contains your main work and it effects everything) should always be bare repository, they shouldn’t have working directory**. To host central repository. Most central repository are created through third party services such as, Bitbucket Cloud or Bitbucket Services. They handle hosting services for you. **You can clone a repository, which creates a local copy of the central repository, called a local repository (you basically have a copy of the core that you can do anything, without damaging the core)**. You can edit, stage, and commit on your local repository, If you or your group are satisfied with the local repository you can, then **“push” your local repository to the central repository, this is equivalent to making your clone core the main core.** But “pushing” your work to get GitHub can cause conflict if you are collaborating with people. What if person A and person B pushed their work around the same time, this could overwrite person A’s work or person B’s work creating a whole mess, this is when you use the **“pull”**. Imagine there is a leader **(usually the person who created the repository)** who got a message or a email that person A and person B are pushing their code to the repository, they can’t push their code unless the leader allows it, the leader liked person A’s work thus, he **pulled his work to the repository, this is the essence of the pull command.**

**Reference:**

[**https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow**](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)