**GIT:** This is a system designed to keep track of changes that happen across a set of files. It is the most popular version control system in the world, and because of this is one that I should take more time to understand deeper. One thing it does really well is keep track of the history of changes made by who, and also revert back to a previous state in case it’s bad. Changes are automatically merged which helps big time in team work.

Git is a distributed system, meaning that each team member has a copy of the project with its history on their machine. This is super good in the case of failure, as when a centralized system goes down, everybody’s hands are basically tied until it comes back online. As Mosh says, the 4 big things that make Git superior are that it’s **Free, Open Source, Super Fast,** and **Scalable.**

**Subversion (SVN)**: This system is a centralized system, which is arguably one of its greatest impediments. This means each team member must always connect to a centralized repository to check in when working. To my understanding, SVN is easier to learn despite some complexities, which I would imagine may have something to do with its ability to handle binary files on a large scale efficiently. Understandably, all files within these kinds of directories are .svn files, making them somewhat homogeneous.

**Mercurial:** This system is also a distributed one similar to Git, but it is a bit simpler and friendly in terms of its user interface. Along with this it has the lovely fact that it’s an open source software and is relatively quick in terms of the speed of its function. This makes it a viable option for large projects as it can smoothly handle projects with even thousands of files and changes all together. Things such as cloning, updating and merging are all said to be no problem for this system, though I do not really know as I have not personally used this. The main drawbacks that I can gather are that it does not handle “partial checkouts” well, leading to increased storage requirements and is inconvenient on those larger projects in these cases. Another drawback is that it is not as popular as Git, leading to less third-party integrations and tools unfortunately. Also, with these less integrated tools comes the issues that arise with compatibility among those tools that do exist.

**Perforce:** Perforce Streams is how branching and merging are done with Helix Core version control. This is a centralized system, which as explained will lead to its own unique downsides. To my understanding Perforce is designed to handle millions of transactions and files smoothly, making it a good option for those operating at a larger scale. The branches in this system are done at the file hierarchy level. Something called “Exclusive Checkouts” give team members visibility into what others are working on, and admins are conveniently able to protect the super important information. Perforce is recommended to be used when dealing with large codebases, non-code assets (binaries or graphics), extensive code reuse, code dependencies, and generally with large spread out teams.

**Team Foundation Server:** This is Microsoft’s version of software development management. This is another centralized system, which is less surprising to be coming from them. The way it’s setup is with a focus on enabling a collaborative environment within the team by bringing together multiple parts of the cycle such as source control, team build , project tracking ,testing, data collection/reporting, team project portal and shared service. This is also available as something that is standalone, or as “the server side back end platform for Visual Studio Team System”. This is very good news for someone like me if I were to be at a company who uses this because I have only really used Visual Studio and am very accustomed to it. Also this is good because of the wide range of extensions and support that VS provides.