Git is a version control system system developed in 2005 by Linus Torvalds, the known creator of the Linux operating system kernel. Version control systems (VCS), like Git, allow for the tracking of changes made to code. More importantly, VCS protects the source code from any major errors that may happen due to changes. VCS also allows for multiple people to work from the same repository, where the source code is stored. Changes to the source code are preserved by Git, safeguarding the code and preventing overwrites. This means multiple people can work on the same shared codebase, but their individual changes are tracked by Git to ensure no conflicts exist before changing the actual source code in the repository. This also means that if one user were to break the code in their local system, it would not break for other users working on the same code. The changes made are also stored by Git, meaning that we could go back to an older version of the code if the new version is broken. You could also go back to work on an older feature and then revert to the present version of the source code. As a result of this flexibility, Git has become the de facto standard within the industry. Many IDEs already come with Git integration and facilitate its use.

All Git projects start with a repository. A repository is a virtual storage of your project. You must first initialize the repository by using the *git init* command. It is a one-time command, and it creates the repository. If you are getting an existing repository, you could clone it by using *git clone*. With *git clone*, the latest version of the remote repository files on the main branch will be collected and added to a new folder on the user’s local machine. Once the repository is acquired, work on the code can begin. When changes are made, and users are ready to send their code to the main repository, they must commit those changes. Using the command *git add* adds the updated file to the staging area, a place to see all the files that are going to be changed. Using *git commit* commits the changes to the repository. These changes are then pushed onto the remote repository, where the source code is stored, by using *git push*. Pushing changes is mainly decided by the team as a collective, and pushing new code could have serious consequences that could cause damage.

Once working on a project, changes are bound to occur from many different sources. The command *git fetch* allows the user to download all new commits people have made into your local repository without merging such changes into your repository. The command *git pull* is the more aggressive version, as it automatically merges the changes into your local repository, but that could cause problems called merge conflicts. Pull requests are extremely important to the workflow used in your work environment. When filing a pull request, you simply request that the main project maintainer take a look at the changes you made. They can provide feedback and even commit changes of their own before finally merging the code into the source code.

There are a couple of different workflows that Git is especially useful in and allows for smooth collaboration between developers. The first of these is the Feature Branch workflow. All feature development occurs in a separate branch from the main branch. This makes it easy for multiple developers to work on a feature while leaving the main branch undisturbed. This also means that the main branch never contains broken code, as branches are heavily reviewed before being merged into the main branch. The discussion takes place, and the code is corrected to allow for seamless integration. Gitflow is yet another workflow that is used by teams for developing software. Gitflow is best for projects that have a scheduled release cycle. Two branches keep track of the version history: the main branch and the develop branch. Feature branches are forked off the develop branch, and once finished, the develop branch is merged into the main branch for release. These workflows allow the use of Git to its maximum potential, and developers work more efficiently to achieve the goals set by the team.