Activity 5 - VCS use report

## How I complied with the organisational requirements

It was helpful that Globex was using **Git** because it supported a Distributed method of development. It was also using **Github** because most developers were familiar with it and it facilitated interacting with Git and other developers.

Since Git was already the preferred mode of software development, I made sure to:

1. Ensure that the Musoplan repository configurations were set correctly in the beginning, so that each commit reflects my name and reference.
2. I used ‘main’ for initialisation but also ensured changes are committed only to branches
3. I used commit messages that followed industry standard, were frequent and had a scope of one change at a time.
4. Once a feature change was complete, I submitted a Pull-Request (on Github) to request merging my changes into the Source Repository, after a code-review by peers.

## Difficulties or challenges during development

There were a few short difficulties that I experienced during development:

* Once I was making a long-term change in branch1 and got a small one-line change to be made in branch2, I found it was easy to make a mistake and commit the small change in branch1 (instead of branch2). So although switching branches was easy, remembering branches before committing needed some practice.
* Another issue was that I found Github a little difficult to use for back-and-forth review and revision of a pull-request.

## How I verified the VCS was performing as expected

* Branches:
  + Each (long-term) feature development meant I had to branch out from the *‘main*’ branch.
  + During development I often need to merge from main, to make sure that my changes were in tandem with work being done by my peers.
* Stages:
  + At times a logical change was very large and although correct, it was good practice to be submitted in smaller change-sets. Staging allowed me to choose a small chunk of files to be committed with an appropriate commit message.
* Commits:
  + A commit is a uniquely identifiable state of a Repository, that can be reverted to at any point in time. While making changes once I made a minor change that I had to undo. I was able to use Git to revert to the most recent commit and thereby start afresh.
* Merges:
  + I used merge not just during development of a feature branch, but importantly when a feature development was complete and I had to merge the feature back into the main branch.
* Push/pull to share commits with remotes:
  + This was used when I needed to sync my local repository with my Github repository.
* Pull requests
  + Once my local changes are pushed to my Repository online, submitting a Pull-Repository upstream allows me to share changes and therefore work together with peers on a common project.
* Commit log
  + Being able to review my recent changes allowed me to see the context of the current branch. This is helpful when switching branches and I want to see the current state of the branch, or getting back to development after a short break.

### If my repository was forked from another repository, how I would go about having them integrate my changes to the central repository

Having my changes integrated to an Upstream repository is a 2-step process. The first step is to ensure that my local changes are based on the latest version of the upstream repo and are then uploaded to my Github repository. This is often done via a *git push* command. Once my local changes are online, I then go to my Github Repository and click on ‘Open a Pull Request’ to create a Pull-Request for Another (Upstream) repository. This would require me to provide a Pull Request message before submitting, and then I would await for any queries / feedback from the other repository’s owner before being accepted / merged.