Version Control

My preferred version control utility is Git. Git is recognized as an industry standard and used around the globe to manage and share source code and projects. There are several repository cloud sites including GitHub and BitBucket that offer free accounts for individuals and small organizations, and paid accounts for larger groups or businesses. These hosting sites help manage repositories by allowing teams to define code commit rules, such as allowing commits to development branches only, managing peer reviews and commit approval, and repository access restrictions. There are many free tools available for managing repositories on local development machines and synchronizing local repositories with cloud-based repositories.

For the purposes of the Code Challenge exercise, I will focus on using the Windows command line interface and GitHub.

Prerequisites:

1. Download and install Git for Windows following the instructions at:
   https://git-scm.com/download/win
2. Create a GitHub account or sign in to an existing account at:
   https://github.com/

Creating a new, local Git repository:

1. Open a command prompt window.
2. Create a new project directory or use an existing project directory to be added to Git.
3. Set the command prompt working directory to the root project directory.
4. Initialize a new repository by executing the command:
   ```
   >git init -b main
   Initialized empty Git repository in C:/Projects/Meliora/.git/
   ```
5. Create a file named .gitignore which contains a set of file descriptors that will be ignored by Git will not be tracked or committed to the repository. You typically want to ignore files that are generated from source code at compile time, tool support files that are specific to a user such as Visual Studio .vs directory files, build log files, etc. For this exercise, I copied a file from GitHub which defines a set of files typically ignored for Visual Studio projects.
   https://github.com/github/gitignore/blob/main/VisualStudio.gitignore
6. Add files to the project directory or use existing files, as necessary.
7. Add project files to the repository.
   ```
   >git add .
   ```
8. Commit changes.
   ```
   >git commit
   ```

9. Enter a commit message, some text describing the nature of the changes included with the commit or use “Initial commit” if committing for the first time. The commit command uses a Unix VI style editor to enter commit messages. A short overview of the process is given here:
   https://stackoverflow.com/questions/33504984/adding-a-git-commit-message-using-vi-on-os-x

Pushing local repository to GitHub

1. Create a new GitHub repository for your project by logging into your GitHub account and clicking the New button on the Repository page. Enter the repository name and a brief description then click Create Repository. Follow the directions on the GitHub page to complete the process which includes setting your local repository to track (sync) with the remote GitHub repository and pushing local changes to the remote repository.
   ```
   >git remote add origin https://github.com/martinpetrella/UofR.git
   >git push -u origin main
   ```

Working on an existing Git Project

1. Logging into GitHub.
2. Navigate to the Repositories page.
3. Select the project you wish to copy and click on the Code dropdown menu.
4. Select the HTTPS URL and copy to the clipboard.
5. Open a Windows command prompt and navigate to the directory you wish to create the new project in.
6. Enter the following command where <url> is the URL in the clipboard. Enter your GitHub credentials if prompted:
   ```
   >git clone <url>
   e.g.
   >git clone https://github.com/martinpetrella/UofR.git
   ```
7. Edit, add, or delete files as needed. Be sure to use the Git add command to add new files to the local repository and the Git delete command to delete existing files from the local repository. Use the Git status command to see local changes.
8. Commit and push changes using instructions above to update the local (commit) and remote (push) repositories.

Examining Change History

To see a complete list of commits made, use the ***git log*** command. This command will display a summary of each commit including a unique commit id, who made the change and when, along with the commit message. Adding the ***–name-status*** flag to the ***git log*** command will add a list of files that changed along with a flag indicating if the file was added, deleted, or modified, to the output. Use ***git diff <commit-id 1> <commit-id 2>*** to get a list of changes between two commits. Use ***git diff <commit-id 1> <commit-id 2> <filename>*** to see the differences between for a specific file between two commits. These commands are useful in tracking down specific changes to files that are of interest, for example, when trying to find where a bug was introduced.