Assignment 01

Question 01: Briefly explain the difference between git and GitHub? What is the relationship between them? (/3 marks)

Git is a version control system that helps developers track and manage changes to their code over time. It allows for local repositories where you can create, edit, and store versions of a project on your machine.

GitHub is a cloud-based platform that hosts Git repositories, enabling collaboration on Git-managed projects. It provides additional tools for code sharing, collaboration, and project management.

The **relationship** between these two are: Git is the tool used to manage versions of the code, while GitHub is a service that allows you to store your Git repositories in the cloud and collaborate with others.

Question 02: What are three advantages of version control systems (VCS)? (/2 marks)

The three **advantages** of version control systems (VCS) are:

- **Collaboration**: Multiple developers can work on the same project simultaneously without overwriting each other's changes.
- **History Tracking**: Version Control Systems keeps a history of all changes, allowing you to roll back to previous versions of the project.
- **Branching and Merging**: Developers can create separate branches for new features or fixes and later merge them back into the main codebase.

Question 03: Complete the following table outlining a series of common git Commands. You should briefly describe the purpose of each command in the area provided. (/5 marks)

GitHub Command	**Description**
clone	**Clone**=>Copies an existing Git repository from a remote source (e.g., GitHub) to local machine.
add	**Add**=>Stages changes (files or updates) that you want to include in your next commit.
commit	**Commit**=>Saves the changes in the staging area as a new version in the Git history with a message describing the changes.
push	**Push**=>Sends your committed changes from your local repository to a remote repository (e.g., on GitHub).
pull	**Pull**=>Fetches and integrates changes from the remote repository into your local repository.

Question 04: What is the purpose of branching in git? (/ 3marks)

The main **purpose of branching in git** is it allows developers to create isolated environments to work on new features, bug fixes, or experiments without affecting the main project. Each branch can evolve independently, and changes can later be merged back into the main branch (typically called master or main).

Question 05: Assuming you have created a new feature in branch, how do you merge that branch back into your master? (/3 marks)

If I have created a new feature in branch, then I will follow the given steps to merge that branch back into my master:
Step 1: Switch to the master branch: git checkout master
Step 2: Merge the feature branch: git merge feature-branch
Step 3: Push the changes to the remote repository: git push origin master

Question 06: Please explain what is meant by the term "merge conflict" in git. Try and provide an example where such an incident might occur. (/3 marks)

A **merge conflict** occurs when Git is unable to automatically merge changes from two different branches because the same part of a file has been modified differently in both branches. For example, if two developers modify the same line of code in a file, Git won't know which change to keep, leading to a conflict that must be resolved manually.

Question 07: Relating to question 06, what is the common (or manual) way of resolving a "merge conflict"? You can explain the main idea or use git commands. (/3 marks)

The following ways are the common or manual way of **resolving a merge conflict:**

1. Open the conflicted file(s) and manually choose which changes to keep.
2. Remove conflict markers like <<<<<<<, =======, and >>>>>>>.
3. After resolving, add the file back to the staging area using -- git add.
4. Commit the changes with -- git commit.

Question 08: An alternative to "merge" is "rebase". Briefly research and explain the differences between "merge" and "rebase" operations in git. (/4 marks)

In Git, both "merge" and "rebase" are used to integrate changes from one branch into another, but they work differently:

- **Merge**: Combines the changes from two branches into one. It creates a new "merge commit" that includes both sets of changes, preserving the history of both branches. It's straightforward and keeps the history intact but can lead to cluttered logs.

- **Rebase**: Re-applies the changes of one branch on top of another. Instead of combining histories, it moves (or "replays") the commits from one branch on top of the current branch. This results in a cleaner, linear history, but it rewrites commit history, which can be risky when working on shared branches.

The main Key Differences between merge and rebase:

- **Merge** preserves the commit history, showing how branches diverged and merged.
- **Rebase** creates a clean, linear history by placing changes from one branch onto another but rewrites commit history.

Both have their use cases: merge is safer for public branches, while rebase is useful for maintaining a cleaner history in feature branches.

Question 09: What is a “pull request”? Explain the process of using a “pull request” to submit changes from a feature “branch”. (/3 marks)

A **pull request** is a request to merge changes from one branch into another, usually from a feature branch into the master/main branch. The process typically involves:

1. Pushing the feature branch to a remote repository (for example., GitHub).
2. Creating a pull request through GitHub’s interface.
3. Team members review the changes, and once approved, the feature branch is merged into the target branch.

Question 10: What is a “commit” in git? (/2 marks)

A **commit** can be defined as a snapshot of the current state of the project. Each commit records the changes made to the files along with a message describing those changes. Commits create a history of modifications, which can be revisited.

Question 11: Briefly explain the concept of GitHub Actions. What are they? How are they used? (/3 marks)

GitHub Actions is a tool within GitHub that automates tasks like testing, building, and deploying code. It allows developers to create custom workflows that are triggered by events in a repository, such as pushing code or opening a pull request. These workflows help streamline Continuous Integration and Continuous Deployment (CI/CD) processes, improving efficiency and code quality.

Question 12: What is a CI/CD pipeline? (/2 marks)

A **CI (Continuous Integration) / CD (Continuous Deployment) pipeline** is a set of automated processes used in software development to build, test, and deploy code changes more frequently and reliably. CI (Continuous Integration) ensures code changes are tested and merged regularly, while CD (Continuous Deployment) automatically deploys the changes to production.