Republic of Rwanda
City of Kigali

GASABO DISTRICT

END OF TERM II GASABO DISTRICT COMPREHENSIVE ASSESSMENT 2022-2023

Exam Title: SWDVC301 Conduct Version Control

SECTOR: ICT

RTQF LEVEL: 3

TRADE: Software Development

Marking guide

MARKS: /100

SECTION A: Attempt All questions **(55marks)**

---

01. Answer as indicated**(1mark for each)**

Choose the correct answer by writing the letter that best fits

I. What is a repository in version control?

A) A tool used to view the changes made to code over time

B) A central location where all versions of code are stored

C) A process used to merge changes made by different team members

D) A system used to track the status of bugs in code

Answer: B) A central location where all versions of code are stored.

II. Which of the following is an example of a version control system?

A) Excel

B) Microsoft Word

C) Git

D) Adobe Photoshop

Answer: C) Git.

III. What are the benefits of using a version control system?

A) It allows multiple developers to work on the same codebase simultaneously

B) It tracks changes made to code over time

C) It allows you to roll back to a previous version of code if needed

D) All of the above

Answer: D) All of the above.

IV. What is the difference between a local repository and a remote repository?

A) A local repository is stored on a server, while a remote repository is stored on your local machine

B) A local repository is used for testing, while a remote repository is used for production

C) A local repository is stored on your local machine, while a remote repository is stored on a server

D) A local repository is used for collaboration, while a remote repository is used for individual development

Answer: C) A local repository is stored on your local machine, while a remote repository is stored on a server.

V. What is a branch in version control?

A) A separate copy of code that you can work on independently

B) A tool used to view the changes made to code over time

C) A process used to merge changes made by different team members

D) A system used to track the status of bugs in code

Answer: A) A separate copy of code that you can work on independently.

02. Define the following terms: **(5marks)**

a) Version control
b) Terminal
c) Status
d) Commit
e) Fetch

a) Version control: Version control is a system that allows you to keep track of changes made to files and documents over time. It helps you to manage and keep track of different versions of the same file, collaborate with others, and restore previous versions if necessary.

b) Terminal: A terminal, also known as a command-line interface or console, is a text-based interface for interacting with a computer's operating system. It allows you to execute commands and perform tasks using text commands rather than a graphical user interface.

c) Status: In the context of version control, "status" refers to the current state of a file or repository. For example, a file may have a "modified" status if changes have been made to it since the last commit, or a repository may have a "clean" status if there are no outstanding changes.

d) Commit: A commit is a snapshot of changes made to a file or repository in a version control system. It is a way of saving changes and making them permanent, so that they can be tracked and restored later if necessary.

e) Fetch: Fetching is a command used in version control systems to download changes made by others and update your local repository with those changes. It allows you to keep your local copy of a repository up-to-date with changes made by other team members.

03. By using a graph, explain local version control**(5marks)**

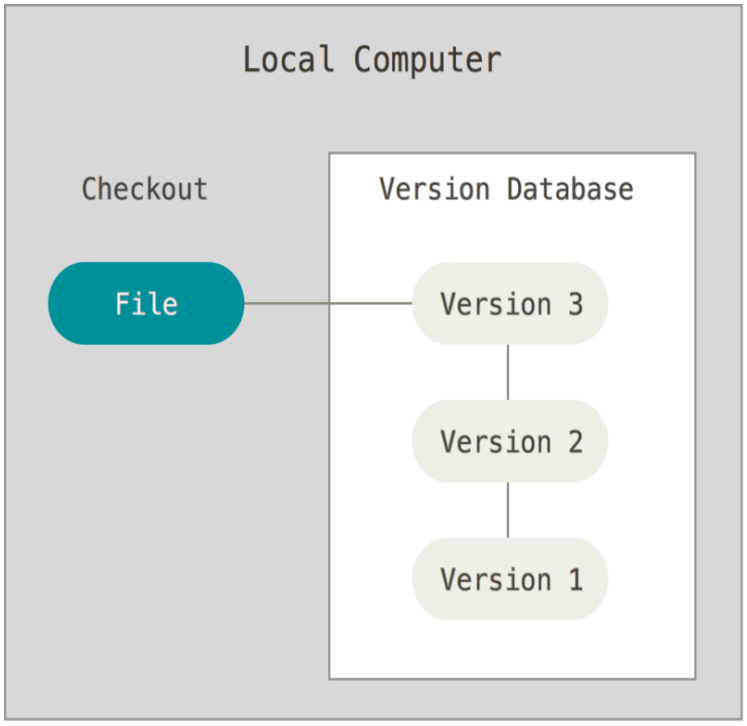
sol:

A local version control system (VCS) is a type of version control system that allows a user to manage changes to a project or file within their local computer or device.

In a local VCS, the user creates and maintains a repository on their own machine, which contains all the versions of the files they are working on. The user can make changes to a file and save multiple versions of it within the repository.

Local VCSs typically track changes by creating a copy of the file every time a change is made. This allows the user to revert to a previous version of the file if necessary. Local VCSs do not require an internet connection, as all changes are stored locally on the user's machine.

Examples of popular local VCSs include Git, Mercurial, and Subversion.



04. After explaining GitHub, list some three(3) features of GitHub which make it so popular **(5marks)**

sol:

GitHub is a web-based platform that provides a centralized repository for version control and collaboration for software development projects. Here are three features that make GitHub so popular:

Collaboration: GitHub provides a platform for developers to collaborate on projects with other developers in a streamlined and organized manner. It allows for real-time collaboration, which means that developers can work on the same project simultaneously, regardless of where they are in the world. The platform also allows developers to review and comment on each other's code, making it easier to catch errors and improve code quality.

Open-source community: GitHub has a large and active open-source community, which makes it easy for developers to find and contribute to existing projects. This community allows developers to share code and collaborate on projects that are of interest to them, making it easier to build complex software projects.

Integration with other tools: GitHub integrates seamlessly with a wide range of other tools and services, making it easy for developers to incorporate GitHub into their existing workflow. For example, GitHub can be integrated with continuous integration and deployment tools, project management tools, and issue trackers, among others. This integration allows developers to automate many aspects of the software development process, reducing the risk of errors and speeding up development time.

05. Briefly, differentiate Git from Mercurial (also known as "hg") as distributed version control systems **(5marks)**

sol:

Git and Mercurial (also known as "hg") are both popular distributed version control systems that allow developers to manage changes to their codebase. While they share many similarities, there are some key differences between the two:

Terminology: Git and Mercurial use different terminology to refer to the same concepts. For example, Git uses the term "commit" to refer to a change to the codebase, while Mercurial uses the term "changeset". Similarly, Git uses the terms "fetch" and "pull" to refer to different actions that can be taken to update the local repository, while Mercurial uses the term "pull" to refer to both actions.

Workflow: While both Git and Mercurial support similar workflows, they differ in some key areas. For example, Git allows developers to work on multiple branches simultaneously, which can be useful for complex projects with many contributors. Mercurial, on the other hand, has a simpler workflow that may be better suited for smaller projects.

Performance: Both Git and Mercurial are designed to be fast and efficient, but they use different algorithms to achieve this. Git uses a content-addressable filesystem to store its data, which can make it faster for certain operations, such as branching and merging. Mercurial uses a more traditional filesystem, which can make it slower for some operations but more predictable in its behavior.

Community and ecosystem: Git has a larger and more active community than Mercurial, which means that there are more resources and tools available for developers who use Git. However, Mercurial has a dedicated and passionate user base that has developed a wide range of plugins and extensions to enhance its functionality.

In summary, while Git and Mercurial are similar in many respects, they have some key differences in terminology, workflow, performance, and ecosystem that may make one more suitable than the other for a given project or team.

06. Discuss about Concurrent Version System **(5marks)**

sol:

Concurrent Versions System (CVS) is a version control system that was popular in the 1990s and early 2000s. CVS was one of the first widely used version control systems and was designed to enable concurrent access to files by multiple users.

CVS works by creating a central repository that stores all versions of the files in the project. Developers can check out files from the repository to their local machines, make changes to the files, and then check them back into the repository. CVS keeps track of all changes made to the files, allowing developers to revert to earlier versions of the files if necessary.

One of the key features of CVS is its ability to handle multiple branches of development in a single repository. This allows developers to work on different versions of the code simultaneously, without interfering with each other's work. CVS also includes features for merging changes made by different developers to the same file.

However, CVS has some limitations compared to more modern version control systems like Git and Mercurial. For example, CVS does not support distributed development, meaning that all changes must be made to the central repository. This can make it difficult to work with geographically dispersed teams or in environments where internet connectivity is unreliable.

In addition, CVS has some limitations in terms of performance and scalability. As projects grow in size and complexity, the time it takes to perform certain operations (such as branching and merging) can become prohibitively slow.

07. After defining SVN(subversion), mention how it works with its features. **(5marks)**

Subversion (SVN) is a centralized version control system that allows multiple developers to collaborate on a project and track changes to the codebase. Here's how SVN works and some of its key features:

1. Repository: SVN uses a centralized repository that stores all versions of files in the project. Developers can check out a copy of the repository to their local machine, make changes, and commit those changes back to the repository.

2. Versioning: SVN keeps track of all changes made to files in the repository, including who made the change, when it was made, and what was changed. This allows developers to easily roll back to previous versions of files if necessary.
3. Branching and Merging: SVN allows developers to create branches in the repository, which are separate copies of the codebase that can be worked on independently. Changes made to a branch can be merged back into the main codebase when they are ready.
4. Access Control: SVN has robust access control features that allow administrators to control who can access the repository, and what level of access they have. This can include read-only access, write access, and more.
5. Integration: SVN can integrate with a wide range of tools and workflows, including build systems, bug tracking systems, and more.
6. Cross-Platform: SVN is designed to work on a wide range of operating systems, including Windows, macOS, and Linux.

Overall, SVN is a reliable and feature-rich version control system that is well-suited for centralized software development projects. Its centralized architecture and robust access control features make it a popular choice for many organizations. However, some developers prefer distributed version control systems (DVCS) like Git or Mercurial, which offer different workflows and capabilities.

08. List and explain five (5) applications of Github in software development **(5marks)**

sol:

GitHub is a web-based platform that is widely used for software development and version control. Here are five common applications of GitHub in software development:

Version Control: GitHub is primarily used for version control, allowing developers to store and track changes to their code over time. With GitHub, developers can easily collaborate on code by forking repositories, making changes, and submitting pull requests to merge their changes back into the main codebase.

Issue Tracking: GitHub provides a built-in issue tracking system that allows developers to create, track, and prioritize issues related to their projects. Issues can be used to report bugs, suggest new features, or track progress on specific tasks.

Continuous Integration/Continuous Deployment (CI/CD): GitHub integrates with many popular CI/CD tools, such as Jenkins, Travis CI, and CircleCI. This allows developers to automate the build, testing, and deployment process of their code, improving efficiency and reducing errors.

Code Reviews: GitHub provides a robust code review system that allows developers to review and comment on each other's code changes. This can help catch errors and improve the quality of the codebase.

Documentation: GitHub provides a platform for hosting and managing project documentation. This can include documentation for code, APIs, or other technical resources. With GitHub, developers can easily create and update documentation, making it accessible to other developers and stakeholders.

Overall, GitHub is a versatile and widely used platform that offers many applications for software development. Its integration with other tools and workflows, robust version control, and collaboration features make it a popular choice for many developers and organizations.

09. Illustrate step by step how to Create account on GitHub**(5marks)**

Sure, here are the steps to create an account on GitHub:

1. Open your web browser and navigate to the GitHub website: https://github.com/
2. Click on the "Sign up" button in the top right corner of the page.
3. On the "Create your account" page, enter your username, email address, and password. You can also choose to sign up with your Google account, if you prefer.
4. Once you have entered your information, click the "Create account" button at the bottom of the page.
5. On the next page, you will be asked to choose your plan. GitHub offers both free and paid plans, depending on your needs. You can choose to start with the free plan, which allows for unlimited public repositories and a limited number of private repositories.
6. Once you have chosen your plan, click the "Continue" button.
7. On the next page, you will be asked to verify your email address. GitHub will send you an email with a verification link. Click on the link in the email to verify your email address.
8. Once you have verified your email address, you will be taken to your new GitHub dashboard. From here, you can start creating repositories, exploring other projects, and collaborating with other developers

10. Explain the following various commands used when viewing a list of commits **(5marks)**

a) git log
b) git log –oneline

a) git log: This command displays a list of commits in reverse chronological order (newest to oldest). Each commit is displayed with its commit hash, author, date, and commit message. By default, git log will display all commits in the current branch, but you can also specify a different branch or range of commits to view.

b) git log --oneline: This command displays a more condensed version of the commit history. Each commit is displayed on a single line, with its abbreviated commit hash and commit message. This can make it easier to quickly scan through the commit history and identify relevant changes.

In summary, both **git log** and **git log --oneline** are used to view lists of commits in Git. The **git log** command provides a detailed view of commits, while **git log --oneline** provides a more condensed view.

11. Differentiate centralized version control system from distributed version control. **(5marks)**

sol:

Centralized version control systems (CVCS) and distributed version control systems (DVCS) are two different ways of managing changes to code or other files in software development. Here are the key differences between them:

1. Centralized Version Control Systems (CVCS):
- In a centralized version control system, there is a central repository that stores all versions of the files.
- Developers working on the codebase check out files from the central repository to make changes.
- Changes are made locally, and then pushed back to the central repository for others to see and use.
- Examples of CVCS include Subversion (SVN) and Microsoft Team Foundation Server (TFS).
2. Distributed Version Control Systems (DVCS):
- In a distributed version control system, there is no central repository.
- Each developer has their own copy of the entire codebase, including the full version history of all files.
- Changes are made locally and then committed to a local repository on the developer's machine.
- Developers can then push their changes to other repositories, which can be either other developers' local repositories or a central repository.
- Examples of DVCS include Git and Mercurial.

In summary, the main difference between CVCS and DVCS is the way they manage repositories. In CVCS, there is a central repository where all files and version history are stored, while in DVCS, each developer has their

own copy of the entire codebase, and changes can be pushed and pulled between different repositories.

SECTION B: Attempt any three (3) questions (30 marks)

---

12. Explain Git version control system with its ten (10) explained features **(10marks)**

sol:

Git is a distributed version control system used for software development. It was created by Linus Torvalds in 2005 and has become one of the most widely used version control systems in the world. Here are ten features of Git and what they mean:

1. Distributed: Git is a distributed version control system, which means that every developer has a full copy of the codebase and its complete history. This allows developers to work on code offline, create their own branches, and merge changes without needing to connect to a central repository.
2. Branching and Merging: Git makes branching and merging easy, allowing developers to create their own branches to work on specific features or fixes. Branches can be easily merged back into the main branch when the changes are ready.
3. Lightweight: Git is a lightweight version control system, meaning that it has a small footprint and does not require a lot of resources to run. This makes it easy to install and use on any system.
4. Speed: Git is designed to be fast, even when dealing with large repositories. This is because it stores changes as incremental differences rather than complete file copies.
5. Security: Git has built-in security features that allow developers to sign and verify commits, ensuring that changes are made by authorized users and have not been tampered with.
6. Staging Area: Git uses a staging area, also known as the index, to allow developers to selectively choose which changes to commit. This allows for more fine-grained control over the codebase and helps prevent mistakes.
7. Revisions: Git tracks every change made to the codebase, allowing developers to easily revert to previous versions if needed. This provides a safety net in case something goes wrong.
8. Collaboration: Git makes collaboration easy, allowing multiple developers to work on the same codebase at the same time. Changes can be pushed and pulled between repositories, making it easy to share code and stay up-to-date.
9. Open Source: Git is an open-source project, which means that anyone can contribute to its development and improvement. This

has helped to make it one of the most widely used version control systems in the world.

10. Extensible: Git is highly extensible and can be customized to meet the needs of individual developers and teams. There are a variety of third-party tools and plugins available that can enhance its functionality and improve the development workflow.

13. Discuss on all Git basic concepts as follow: **(10marks)**

a) VCS
b) Repository
c) Commit
d) Branch
e) Merging
f) Pull request
g) Remote
h) Staging
i) Git push
j) Git push origin master

a) VCS: VCS stands for Version Control System, which is a software tool used for managing changes to code or other files in software development. VCS allows multiple developers to work on the same codebase and keep track of changes made over time.

b) Repository: A repository, also known as a repo, is a central location where all the code and other files for a project are stored. It is a container for all the branches, commits, and other data related to the project.

c) Commit: A commit is a snapshot of changes made to the codebase at a particular point in time. It includes a message that describes the changes made and provides context for other developers.

d) Branch: A branch is a separate line of development that diverges from the main codebase. It allows developers to work on features or fixes without affecting the main codebase until they are ready to merge their changes.

e) Merging: Merging is the process of combining changes made on one branch with another branch. It is often used to integrate features or bug fixes into the main codebase.

f) Pull request: A pull request is a feature in Git that allows developers to submit their changes to the main codebase for review and possible merging. It provides a way for developers to collaborate and review each other's work before it is merged into the main codebase.

g) Remote: A remote is a version of a repository that is hosted on a server or other location outside of the local machine. It allows

developers to collaborate on code and keep multiple copies of the codebase in sync.

h) Staging: Staging, also known as the index, is a feature in Git that allows developers to selectively choose which changes to commit. It allows for more fine-grained control over the codebase and helps prevent mistakes.

i) Git push: Git push is a command used to upload changes made to a local repository to a remote repository. It allows developers to share their changes with other team members and collaborate on code.

j) Git push origin master: Git push origin master is a command used to upload changes made to the master branch of a local repository to a remote repository named "origin". It is a common command used to share changes with other team members and update the main codebase.

14. Indicate brief explanations and usage of the git clone and git remote commands. **(10marks)**

1. git clone: The git clone command is used to create a copy of an existing Git repository on a local machine. It is used when a developer wants to start working on a project that already exists in a remote repository. The syntax of the command is as follows:

   git clone <repository URL>

   The **<repository URL>** can be a URL to a remote repository hosted on a Git server like GitHub or GitLab. The command will create a new directory on the local machine with the same name as the remote repository and copy all the files and history from the remote repository into the local directory.

2. git remote: The git remote command is used to manage connections to remote repositories. It is used to add, list, or remove remote repositories that the local Git repository can connect to. The syntax of the command is as follows:

   git remote [command] [remote name]
   Some common subcommands for git remote are:

- **add**: Used to add a new remote repository to the local Git repository. The syntax is as follows:

  git remote add <remote name> <repository URL>
  The **<remote name>** is a name for the remote repository that is easy to remember and use, and **<repository URL>** is the URL to the remote repository.

- **remove**: Used to remove a remote repository from the local Git repository. The syntax is as follows:

  git remote remove <remote name>
  The **<remote name>** is the name of the remote repository to be removed.

- **show**: Used to list the names of all the remote repositories connected to the local Git repository. The syntax is as follows:

`git remote show`
This will list all the remote repository names.

Overall, git clone and git remote are essential Git commands that are commonly used in software development workflows. The git clone command is used to create a local copy of a remote repository, while the git remote command is used to manage the connections to remote repositories.

15. Using Git commands and brief explanations, how would you create a new branch in your Git repository? **(10marks)**

sol:

To create a new branch in a Git repository, you can use the **`git branch`** command. Here are the steps to create a new branch:

1. First, make sure that you are in the local Git repository that you want to create a new branch for.
2. Use the following command to create a new branch:

git branch <branch name>

Replace **`<branch name>`** with the name you want to give your new branch. This creates a new branch with the given name but does not switch to it. The branch is created based on the current branch that you are on.

3. To switch to the new branch, use the following command:

git checkout <branch name>
Replace **`<branch name>`** with the name of the branch that you created in step 2. This command switches to the new branch and sets the HEAD pointer to the new branch.

4. You can now start making changes to your codebase in the new branch. When you are ready to commit your changes, use the **`git add`** and **`git commit`** commands as usual.
5. Once you have made changes and committed them to the new branch, you can merge the changes back into the main branch using the **`git merge`** command.

`git checkout main`
`git merge <branch name>`
Replace **`<branch name>`** with the name of the branch that you created in step 2. This merges the changes from the new branch back into the main branch.

Overall, creating a new branch in Git is a simple process that involves using the **`git branch`** and **`git checkout`** commands.

16. Step by step, guide on how to create a new Git repository for an empty project with brief explanations. **(10marks)**

Here's a step-by-step guide on how to create a new Git repository for an empty project:

1. First, create a new directory on your local machine where you want to store the project. You can do this using the **mkdir** command in your terminal or command prompt. For example, **mkdir my_project**.
2. Navigate into the new directory using the **cd** command. For example, **cd my_project**.
3. Initialize a new Git repository in the directory using the **git init** command. This command creates a new **.git** directory inside your project directory and sets up the necessary Git files and folders to start tracking changes.
4. Create a new file in your project directory to start building your project. You can use any text editor to create a new file. For example, **touch index.html**.
5. Add the new file to the Git repository using the **git add** command. For example, **git add index.html**. This stages the file to be committed in the next step.
6. Commit the new file to the Git repository using the **git commit** command. This command creates a new commit with the changes you made to the file and saves it to the Git repository history. For example, **git commit -m "Initial commit"**.
7. Optionally, create a new remote Git repository to push your local Git repository to. This step is not necessary if you just want to keep the Git repository on your local machine. You can create a new remote Git repository on services like GitHub, GitLab, or Bitbucket.
8. Add the new remote Git repository to your local Git repository using the **git remote add** command. For example, **git remote add origin https://github.com/your_username/your_project.git**. This sets up a remote called "origin" that points to the URL of the new Git repository.
9. Push the local Git repository to the remote Git repository using the **git push** command. For example, **git push -u origin master**. This pushes the commits you made locally to the remote Git repository and sets up the "master" branch to track the remote "origin" branch.

That's it! You now have a new Git repository set up for your empty project, with a new file added and committed to the repository. You can continue working on your project, adding new files and making changes, and commit and push your changes to the Git repository as needed.

SECTION C: Attempt only one (1) question (15 marks)

17. a) How do you initialize a new Git repository? **(3marks)**

To initialize a new Git repository, you can follow these steps:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to create your new Git repository. You can use the **cd** command to change to the desired directory.

3. Once you are in the desired directory, use the **git init** command to initialize a new Git repository. This command creates a new empty repository in the current directory.
4. You can then add files to the repository using the **git add** command and commit changes using the **git commit** command.

Here's an example of how to initialize a new Git repository:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to create your new Git repository. For example, if you want to create the repository in your "Documents" folder, you can type the following command: **cd Documents**
3. Once you are in the "Documents" directory, type the command **git init** to initialize a new Git repository.
4. Git will create a new empty repository in the current directory.
5. You can now add files to the repository using the **git add** command and commit changes using the **git commit** command.

c) How do you revert to a previous commit in Git? **(3marks)**

To revert to a previous commit in Git, you can follow these steps:

Open your terminal or command prompt.
Navigate to the local repository that contains the commit you want to revert to using the **cd** command.
Use the **git log** command to get the SHA hash of the commit you want to revert to. You can copy the SHA hash from the log output.
Use the **git checkout** command followed by the SHA hash of the commit to revert to that commit. This command puts your repository in a "detached HEAD" state, which means that you are no longer on a branch. For example, if the SHA hash of the commit you want to revert to is "abcdefg", you can use the following command: **git checkout abcdefg**
If you want to make changes to the code, you can make the changes and commit them as usual.
Once you have made the changes and committed them, use the **git log** command again to get the SHA hash of the commit that represents the new state of the code.
Use the **git checkout** command followed by the SHA hash of the new commit to return to the branch. For example, if the SHA hash of the new commit is "hijklmn", you can use the following command: **git checkout hijklmn**

d) How do you perform code review in Git? **(3marks)**

Performing code review in Git typically involves the following steps:

1. Checkout the branch that contains the changes you want to review. For example, if the branch is named "feature-branch", you can use the command **git checkout feature-branch**.
2. Examine the changes in the branch using a diff tool such as Git's built-in diff tool, or a third-party tool such as Beyond Compare or KDiff3. You can use the **git diff** command to see the changes made to the files in the branch.

3. Provide feedback on the changes made in the branch. You can leave comments in the code or use a tool such as GitHub's pull request review feature to provide feedback on specific lines of code.
4. If changes are needed, the developer can make the changes and push them to the branch again. If the changes are satisfactory, the reviewer can approve the pull request and merge the changes into the main branch.

e) What is the difference between a merge and a rebase in Git? **(3marks)**

Both merge and rebase are used in Git to integrate changes from one branch into another. However, they differ in how they apply changes and how they affect the commit history.

A merge creates a new commit that combines changes from two or more branches. When you merge a branch into another branch, Git creates a new merge commit that has two or more parent commits. The merge commit has a commit message that describes the changes being merged. Merging is a simple and safe way to combine changes from different branches, but it can lead to a messy commit history when there are many merge commits.

A rebase, on the other hand, applies changes from one branch onto another branch by reapplying each commit in the branch onto the target branch. This creates a linear history with all changes applied in sequence, as if they were made on top of the target branch from the beginning. Unlike merging, a rebase does not create a new merge commit, and it changes the commit history of the target branch. Rebasing can be useful for keeping the commit history clean and easy to follow, but it can be more complex and risky than merging, especially when there are many conflicting changes.

In summary, merging creates a new commit that combines changes from different branches, while rebasing applies changes from one branch onto another branch by reapplying each commit in the branch. Merging is simple and safe, but can lead to a messy commit history, while rebasing keeps the commit history clean, but can be more complex and risky. The choice of whether to use merge or rebase depends on the specific situation and preferences of the team.

f) Why is version control important? **(3marks)**

Version control is important because it provides a way to manage and track changes to files and code over time. Here are some reasons why version control is important:

1. Collaboration: Version control allows multiple people to work on the same codebase and track changes made by different team members. It enables

collaboration and makes it easier for team members to work together on the same project.
2. History tracking: Version control keeps a record of every change made to a file or codebase, along with who made the change and when. This makes it possible to track the history of changes, identify issues, and revert to previous versions if necessary.
3. Backup and recovery: Version control systems store code and files in a centralized location, which makes it easy to backup and recover data in case of data loss, hardware failure, or other issues.
4. Testing and experimentation: Version control allows developers to create new branches to experiment with new features or code changes, without affecting the main codebase. This makes it possible to test changes and revert them if they don't work as expected.
5. Code reviews: Version control makes it easy to review code changes made by team members, which helps ensure that code quality is maintained and that best practices are followed.

Overall, version control is essential for managing software development projects, ensuring collaboration and quality, and keeping track of changes made over time.

18. AGABE Adolphe is a Senior Developer of Innovate company Ltd located at Musanze District, he assigned a project to 5 different developers to design web application that contains different forms such as: login form, Student Registration form, Entertainment form, courses registration form and book registration form, using html language and PHP, but due to Covid-19 pandemic developers were not able to work together on the given task and become difficult to control them. Senior developer decided to assign tasks to each developer respectively and remotely. He recommended them to work individually on a given task.

He told them to create their own repository related to a given task,

Task: As a one of 5 developers you are hired to choose one task from above, work on it and create Pull Request of your task to be merged on the main branch, use GitHub as version control platform. Submit Pull Request link to the senior developer **(15marks)**

sol:

As a developer assigned to the project, here are the steps I would take to complete my task and create a pull request for merging on the main branch using GitHub as a version control platform:

1. First, I would choose one task from the given list of forms to work on. Let's say I choose the Student Registration form.
2. I would create my own repository on GitHub related to the task. To do this, I would log in to my GitHub account and click on the "New" button to create a new repository.

3. I would give my repository a name related to the task, such as "Student-Registration-Form".
4. I would initialize the repository with a README file and choose to add a .gitignore file for PHP to ignore unnecessary files in my repository.
5. Next, I would clone the repository to my local machine using the **git clone** command in my terminal or command prompt.
6. I would then start working on the task by creating the HTML and PHP files needed to build the Student Registration form.
7. Once I have completed the task and tested it to ensure it is working properly, I would commit my changes using the **git add** and **git commit** commands in my terminal or command prompt.
8. After committing my changes, I would push the changes to my repository on GitHub using the **git push** command.
9. Once the changes are pushed to my repository, I would create a pull request by going to the "Pull Requests" tab in my repository on GitHub and clicking on the "New Pull Request" button.
10. In the pull request, I would select the main branch of the project as the base branch and my own branch as the compare branch.
11. I would add a description of the changes I made to the form in the pull request and submit it.
12. Finally, I would submit the pull request link to the Senior Developer as instructed, so they can review my changes and merge them into the main branch of the project.

END OF ASSESSMENT!