Git Quiz Answers by Aryan Kumar (12019397)

Q1.
A version control system (VCS) is a system that tracks changes to files over time, allowing developers to collaborate on software development and easily roll back to previous versions of the code. Examples of VCS include Git, Mercurial, and Subversion.

Q2.
A version control system was developed to help manage the complex and ever-changing process of software development. There were several key necessities that led to the development of VCS:

Collaboration: As software development increasingly became a team endeavor, it became necessary to have a way for multiple developers to work on the same codebase simultaneously.

History tracking: Developers needed a way to keep track of the changes made to the code over time, so that they could understand how the code evolved and easily revert to previous versions if needed.

Backup: With the increasing complexity of software, it became important to have a reliable backup system in place to protect against data loss due to hardware failure or other issues.

Branching: As the software development process became more complex, developers needed a way to create different branches of a codebase, so that they could work on different features or fix bugs in isolation, without affecting the main codebase.

Overall, the Version Control System was created to support the collaborative, incremental, and ever-changing nature of software development.

Q3.
There are two main types of version control systems: centralized version control systems (CVCS) and distributed version control systems (DVCS).

Centralized Version Control Systems (CVCS): In a CVCS, there is a central server that stores all the versions of the code and all the developers are connected to the central server and they can only access the latest version of the code. Examples of CVCS include Subversion (SVN) and Perforce.

Distributed Version Control Systems (DVCS): In a DVCS, there is no central server and every developer has a complete copy of the codebase, including its entire history. This allows for offline development and makes it easy for developers to collaborate without needing to be connected to a central server. Examples of DVCS include Git and Mercurial.

Q4.
Centralized Version Control Systems (CVCS):
There is a central server that stores all the versions of the code.
All developers are connected to the central server and can only access the latest version of the code.
Changes made by developers need to be committed to the central server before they can be seen by other developers.
In case of server failure or outage, developers may lose access to the codebase and its history.
Distributed Version Control Systems (DVCS):
There is no central server, every developer has a complete copy of the codebase, including its entire history.
Developers can work offline and commit changes locally, and then push or pull changes to or from other developers.
It allows for more flexible collaboration and workflows.
In case of server failure or outage, developers still have access to the codebase and its history.

Q5.
Git is a distributed version control system (DVCS) that was created by Linus Torvalds in 2005. It is designed to handle everything from small to very large projects with speed and efficiency. Git is widely considered to be one of the most popular and powerful version control systems currently in use, and is used by many large open-source projects and companies, such as the Linux kernel, Ruby on Rails, and Facebook.

With Git, developers have a full local copy of the entire codebase, including its entire history, allowing them to work offline and collaborate with others. When they are ready to share their changes, they can commit their changes locally and then push or pull changes to or from other developers. Git also supports branching and merging, which allows developers to work on multiple features or fix bugs in isolation, without affecting the main codebase.

Q6:
Here are a few key features of Git:

Distributed version control: Git is a distributed version control system, which means that every developer has a full local copy of the entire codebase, including its entire history. This allows for offline development and flexible collaboration.

Branching and merging: Git supports branching and merging, which allows developers to work on multiple features or fix bugs in isolation, without affecting the main codebase. This makes it easy to experiment with new ideas and collaborate with others on the same codebase.

Speed and efficiency: Git is designed to handle everything from small to very large projects with speed and efficiency. It is particularly well suited for large, complex codebases with multiple contributors.

Q7:
git init: This command is used to initialize a new Git repository. It creates a new directory called .git in the current directory, which contains all the necessary files for version control. This command is typically used when starting a new project and setting it up for version control with Git.

git add: This command is used to stage changes for commit. It adds changes made in the working directory to the staging area, which is a temporary holding area where changes are stored before they are committed to the repository. This command is used to prepare changes for commit, and it allows developers to include only the changes they want to commit.

git commit: This command is used to commit changes to the repository. It saves the changes that have been staged with the git add command to the repository's history. This command is used to create a new version of the codebase and it also allows developers to add a commit message to describe the changes made.

Q8:
Git and GitHub are not the same thing, although they are closely related.

Git is a version control system (VCS) that was created by Linus Torvalds in 2005. It is a command-line tool that is used to track changes to files over time, allowing developers to collaborate on software development and easily roll back to previous versions of the code.

GitHub, on the other hand, is a web-based platform that was founded in 2008. It is built on top of Git, and it provides a web interface for working with Git repositories. It also provides additional features such as issue tracking, pull requests, and wikis.

GitHub is a place where people can store their Git repository remotely, collaborate with other people, review and merge code changes, track issues, and much more. It is a web-based hosting service for Git repositories, but it also provides a wide range of additional functionality that makes it easy for developers to collaborate on software development.

In summary, Git is a command-line tool for version control, while GitHub is a web-based platform that provides additional functionality for working with Git repositories and for collaborating on software development.

Q9:
The command to get the installed version of Git is git --version.

Q10:
The command to add all files and changes of the current folder to the staging environment of the Git repository is git add -A.

Q11.
git status shows the current state of the repository, including any changes that have been made to the working tree but not yet committed, as well as the current branch and the status of any files that are being tracked.

git log shows a history of commits that have been made to the repository, including the author, date, and message for each commit. It shows the entire commit history of the repository.

In summary, git status shows the current state of the repository, while git log shows the history of commits that have been made to the repository.

Q12.
The command to initialize a Git repository in the current directory is git init.

Q13.
In Git, a file can be in one of three states:

Untracked: This is a file that is not being tracked by Git. It is not part of the repository and will not be included in commits. The command git add is used to start tracking an untracked file and include it in commits.

Modified: This is a file that has been tracked by Git but has been modified since the last commit. The command git diff can be used to see the differences between the modified file and the last committed version. The command git add is used to stage the changes, and git commit is used to commit the changes.

Staged: This is a file that has been modified and its changes have been staged using the git add command. The command git commit is used to commit the changes.

Q14.
False. Git does not automatically add new files to the repository and start tracking them.

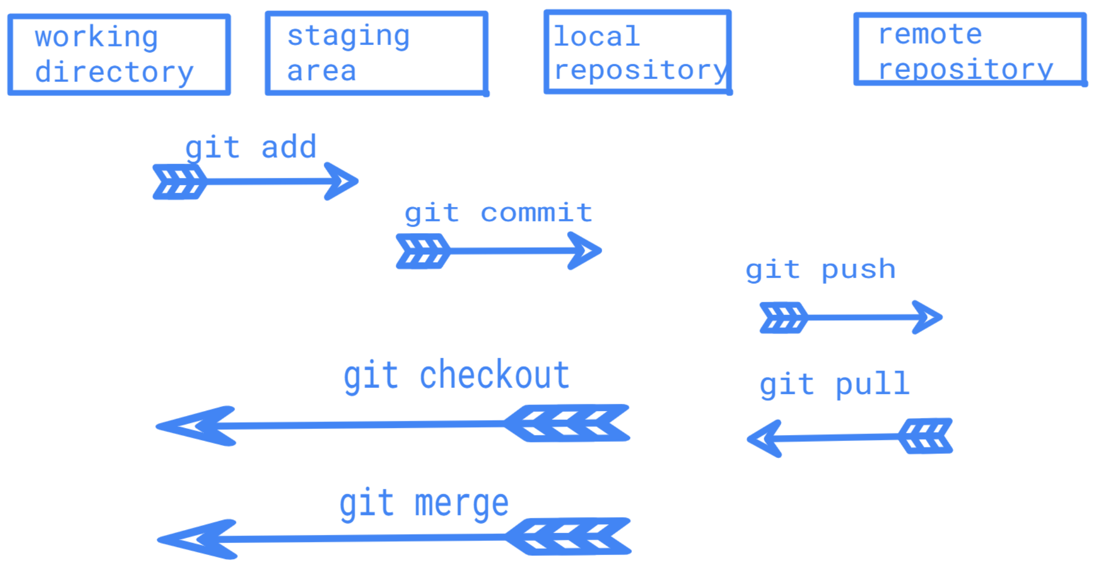
When a new file is created in a directory that is a Git repository, Git does not automatically recognize it as something to be tracked. This is because Git tracks changes to files, not the files themselves. If a new file is created, Git does not know that it should be tracked unless it is explicitly told to do so.

To start tracking a new file, you need to use the command git add <file> to add the file to the staging area and make it part of the repository. Once the file is added, it will be included in the next commit, and Git will start tracking changes to it.

Q15.
The command to commit the staged changes in a Git repository is git commit.

Q16.
git commit -m "New email"

Q17.



Working directory: This is where you create and edit files. The files in the working directory are not under version control yet.

Staging area: The git add command is used to take changes from the working directory and bring them into the staging area. The git status command can be used to check the status of the files in the staging area and working directory.

Local repository: The git commit command is used to take changes from the staging area and bring them into the local repository. The git log command can be used to check the commit history of the local repository.

Remote repository: The git push command is used to take changes from the local repository and bring them into a remote repository (such as GitHub or GitLab). The git pull command can be used to bring changes from the remote repository into the local repository.

Q18.
A branch in Git is a pointer to a specific commit in the repository's history. When you create a new branch, it starts pointing to the same commit as the branch you created it from. When you make new commits, the branch pointer moves forward to the new commit, creating a new linear history of commits.

Branches are useful for developing new features or bug fixes in parallel with the main development branch. They allow you to work on multiple versions of your code at the same time without interfering with each other. Once a feature or bug fix is complete, you can merge the changes back into the main development branch.

In Git, the default branch is called master. Developers can create new branches to work on new features or bug fixes and merge them back to the master once they are done.

Q19.
```
git branch new-email
```

Q20.
```
git checkout new-email
```

Q21.
When moving to a branch in Git, the option to create the branch if it does not exist is the -b flag used in the git checkout command.

For example, to create and switch to a new branch named "new-email" if it does not exist, you would use the command:

```
git checkout -b new-email
```

Q22.
The git init command is used to initialize a new Git repository.

Q23.
A fork is a copy of a repository that is made by a user in order to make changes to the original code without affecting the original repository. A clone, on the other hand, is a copy of a repository that is made on a local machine in order to work on the code locally.

To fork a repository, you can go to the repository on GitHub and click the "Fork" button. This will create a new repository under your own account that is a copy of the original repository.
To clone a repository, you can use the command "git clone [repository url]" in your command line, where [repository url] is the url of the repository you want to clone. This will create a copy of the repository on your local machine.

Q24.
In Git, "push" refers to the action of uploading local commits to a remote repository. When you make changes to your code and commit them locally, you can use the "git push" command to upload those commits to the remote repository.

Q25.
Centralized Version Control System:
In a centralized version control system, there is a single central repository that contains all the versions of the code. All the developers work on their own working copies of the code and then push their changes to the central repository. The central repository acts as a single source of truth for the codebase.

Centralized Version Control System:

The central repository is the "master" copy of the code.
Each developer has their own working directory where they make changes to the code.
The developers use commands like "checkout" and "commit" to update their local working copy and the central repository.

Distributed Version Control System:
In a distributed version control system, each developer has their own copy of the entire code repository. This copy is called a "clone" of the repository. Developers can make

changes to their local copy and push those changes to a remote repository. Other developers can then pull those changes from the remote repository to their own local copy.

Distributed Version Control System:

Each developer has their own "clone" of the repository.
Developers can make changes to their local copy and push those changes to a remote repository.
Other developers can then pull those changes from the remote repository to their own local copy.
This allows for multiple remote repositories, and each developer can work on their own branches, which can be merged later on.

Diagrams:
CVCS:
DVCS: