Day 3 Git and GitHub Assignment

Solution

Question 1: What is Git?

Answer 1: Git is an open-source distributed version control system. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

- Git is foundation of many services like GitHub and GitLab, but we can use Git without using any other Git services. Git can be used privately and publicly. Git was created by Linus Torvalds in 2005 to develop Linux Kernel. It is also used as an important distributed version-control tool for the DevOps. Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase
- Features of Git Some remarkable features of Git are as follows:
    - Open Source
    - Scalable
    - Distributed
    - Security
    - Speed
    - Supports non-linear development
    - Branching and Merging
    - Data Assurance
    - Staging Area
    - Maintain the clean history
- Benefits of Git: A version control application allows us to keep track of all the changes that we make in the files of our project. Every time we make changes in files of an existing project, we can push those changes to a repository. Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.
- Some significant benefits of using Git are as follows:
    - Saves Time
    - Offline Working
    - Undo Mistakes
    - Track the Changes

Question 2: What do you understand by the term ‘Version Control System’

Answer 2: Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

- A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled, they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.
- Benefits of the version control system:
    - Enhances the project development speed by providing efficient collaboration,
    - Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
    - Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
    - Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,
    - For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,
    - Helps in recovery in case of any disaster or contingent situation,
    - Informs us about Who, what, When, why changes have been made.
- Types of Version Control Systems:
    - Local Version Control Systems
    - Centralized Version Control Systems
    - Distributed Version Control Systems

Question 3: What is GitHub?

Answer 3: GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

- GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.
- It offers both distributed version control and source code management (SCM) functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.
- Features of GitHub: GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open-source projects and codes of various programming languages.
- Some of its significant features are as follows.
    - Collaboration
    - Integrated issue and bug tracking

    - Graphical representation of branches
    - Git repositories hosting
    - Project management
    - Team management
    - Code hosting
    - Track and assign tasks
    - Conversations
    - Wikisc
- Benefits of GitHub
    - GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.
    - The key benefits of GitHub are as follows.
    - It is easy to contribute to open-source projects via GitHub.
    - It helps to create an excellent document.
    - You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
    - It allows your work to get out there in front of the public.
    - You can track changes in your code across versions.

Question 4: Mention some popular Git hosting service.

Answer 4: There are several popular Git hosting services where you can host and manage your Git repositories. Some of the most widely used Git hosting platforms as of my last knowledge update in September 2021 include:

- GitHub: GitHub is one of the most popular Git hosting platforms. It offers both free and paid plans, and it's known for its user-friendly interface and a wide range of features for collaboration and continuous integration.
- GitLab: GitLab is a web-based Git repository manager that provides a platform for DevOps and collaboration. You can use the GitLab Community Edition for free or opt for the more feature-rich Enterprise Edition.
- Bitbucket: Bitbucket, provided by Atlassian, offers both Git and Mercurial repository hosting. It's known for its integration with other Atlassian products like Jira and Confluence.
- Azure DevOps (formerly Visual Studio Team Services): Azure DevOps, now part of Microsoft Azure, is a comprehensive platform that offers version control, continuous integration, and project management tools. It supports both Git and Team Foundation Version Control (TFVC).
- AWS CodeCommit: Amazon Web Services' CodeCommit is a managed source control service that's tightly integrated with other AWS services. It supports Git repositories and is designed for easy integration into AWS environments.

- SourceForge: SourceForge has been around for a long time and offers Git repository hosting along with a variety of tools for open-source software development and collaborative coding.
- Gitea: Gitea is an open-source self-hosted Git service that provides a lightweight, easy-to-install platform for managing Git repositories. It's a good option for those who prefer to host their own Git server.
- GitBucket: GitBucket is another open-source alternative for self-hosted Git repository management, written in Scala. It aims to provide a GitHub-like experience.
- Beanstalk: Beanstalk is a hosted version control service that supports both Git and Subversion. It's designed for simplicity and ease of use.
- RhodeCode: RhodeCode is an enterprise-grade platform for managing code repositories, and it supports both Git and Mercurial. It offers features for code review, access control, and more.

Question 5: Different types of Version control system?

Answer 5: There are three type of Version control system are:

- Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.
- Centralized Version Control Systems: Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.
    - Two things are required to make your changes visible to others which are:
        - You commit
        - They update
    - The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.
    - It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.
- Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in

order to make them visible on the central repository. Similarly, when you update, you do not get others' changes unless you have first pulled those changes into your repository.

- To make your changes visible to others, 4 things are required:
  - You commit
  - You push
  - They pull
  - They update
- The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.

Question 6: What benefits of come with using GIT?

Answer 6: Using Git, a distributed version control system, provides several benefits for individuals and teams working on software development and collaborative projects. Here are some of the key advantages of using Git:

- Distributed Version Control: Git is a distributed version control system, which means that every developer working with a Git repository has a complete copy of the entire project's history. This allows for offline work, faster branching, and greater flexibility.
- Branching and Merging: Git makes it easy to create branches for new features, bug fixes, or experimental work. Merging branches is straightforward and results in a clean and well-documented history.
- Data Integrity: Git uses a cryptographically secure hash function to ensure data integrity. Each commit is identified by a unique SHA-1 hash, making it nearly impossible for changes to be corrupted without detection.
- Collaboration: Git is designed for collaboration. It enables multiple developers to work on the same project concurrently without interfering with each other. Merge conflicts are manageable, and features like pull requests (in platforms like GitHub and GitLab) facilitate code review.
- History and Versioning: Git maintains a complete history of changes, which allows you to revisit any point in time and understand how and why the code evolved. You can easily revert to previous versions or compare differences between commits.
- Speed and Efficiency: Git is known for its speed and efficiency. Most operations are performed locally, reducing the need to communicate with a central server. This makes Git faster than centralized version control systems.
- Support for Large Projects: Git handles large projects with ease. Whether you're working on a small script or a massive codebase, Git scales well and manages files efficiently.
- Customizable Workflows: Git is highly flexible and allows you to define custom workflows that suit your development process. Whether you follow a specific branching model or use Git for tasks beyond code, it can be adapted to your needs.

- Open Source and Wide Adoption: Git is an open-source system with a vast and active community. It's widely adopted in the software development industry and has numerous tools, extensions, and resources available.
- Backup and Disaster Recovery: With Git, your code and project history are distributed across multiple repositories. This redundancy serves as a form of backup and disaster recovery.
- Integration: Git integrates well with many other tools and services, including issue tracking systems, continuous integration platforms, and code review tools.
- Cross-Platform Compatibility: Git is available on multiple platforms (Windows, macOS, Linux), making it accessible to developers using different operating systems.
- Security: Git allows you to control access to repositories, manage user permissions, and keep sensitive data secure.
- Ecosystem: Git is part of a broader ecosystem that includes popular platforms like GitHub, GitLab, and Bitbucket, which offer additional features for collaboration, issue tracking, and continuous integration.

Question 7: What is Git repository?

Answer 7: A Git repository, often referred to as a "repo," is a data structure that stores the metadata and content for a set of files and directories, as well as the entire history of changes made to that data. In other words, a Git repository contains your project's source code, and it keeps track of all the changes made to that code over time. Here are some key aspects of a Git repository:

- Version Control: The primary purpose of a Git repository is to provide version control for your project. It tracks every change made to the files and directories within the repository, creating a history of these changes.
- Metadata: Git stores metadata about each commit, including the author, timestamp, and a unique identifier (a hash) for the commit. This information is essential for tracking and managing changes.
- Branches: Git repositories can have multiple branches, which represent different lines of development. Each branch can contain different sets of changes and modifications to the project.
- Commits: Commits are the individual snapshots of the project at a specific point in time. Each commit represents a set of changes made to the files and directories. Commits are linked together in a chronological order to create a history of the project.
- Working Directory: The working directory is a directory on your local machine where you can make changes to the files in your repository. These changes can be staged and committed to the repository when you're ready.
- Staging Area (Index): The staging area is an intermediate step between your working directory and the repository. It allows you to select and prepare changes for the next commit. This gives you control over which changes are included in a commit.

- Remote Repositories: In addition to a local Git repository on your machine, you can also have remote repositories hosted on servers. Remote repositories enable collaboration and code sharing with other developers. Popular remote repository hosting platforms include GitHub, GitLab, and Bitbucket.
- History: The repository's history is a complete record of all changes made to the project. You can view the history, compare different versions, and even revert to previous states if needed.

Question 8: How can you initialize a repository in git?

Answer 8: To initialize a Git repository, you need to follow these steps:

- Open a Terminal or Command Prompt: You'll need to use your computer's command-line interface to run Git commands.
- Navigate to the Directory Where You Want to Create the Repository:** Use the `cd` command to navigate to the directory (folder) where you want to create your Git repository. For example, if you want to create a repository in a directory named "my_project," you would navigate to that directory like this:
    - Git bash
        - cd path/to/my_project
- Run the `git init` Command: This command initializes a new Git repository in the current directory. It sets up the necessary Git files and folders to start tracking your project.
    - GitBash
        - git init
- Optional: Create or Add Files: At this point, your repository is initialized, and you can start adding files to it. You can create new files in the directory or copy existing files into it.
- Optional: Stage and Commit Files: If you've added files, you'll need to stage and commit them to start tracking changes. Use the following Git commands to stage and commit your changes:
    - `git add <file>`: This command stages a specific file for the next commit.
    - `git add .` or `git add -A`: These commands stage all changes in the current directory.
    - `git commit -m "Your commit message"`: This command commits the staged changes with a descriptive message.
- Here's a complete example:

```
Gitbash

# Navigate to your project directory

cd path/to/my_project

# Initialize a Git repository
```

```
git init

# Create or copy files into the directory

# Stage and commit your changes

git add .

git commit -m "Initial commit"
```

- Your Git repository is now initialized, and you can continue to work on your project, make changes, and use Git for version control. If you plan to collaborate with others or use a remote repository hosting service like GitHub or GitLab, you can connect your local repository to a remote one following their respective instructions.