1. Managing Source Code – Git and GitHub – 6 hrs.

Information

1. Overview of Version Control System git

- Git is a distributed version control system (VCS) that allows multiple developers to collaborate on a project while keeping track of changes made to the source code.
- It provides a structured way of managing code versions and enables teams to work simultaneously on different branches, merge changes, and track the history of the codebase.

Here is an overview of Git:

1. Repository: A Git repository is a central location where all the project's files and their history are stored. It contains all the versions of the code, branches, and metadata related to the project.

2. Commits: A commit is a snapshot of the code at a specific point in time. It represents a set of changes made to the files in the repository. Each commit has a unique identifier and includes a message describing the changes.

3. Branches: Branches allow for parallel development by creating separate lines of development. They enable multiple team members to work on different features
4. or bug fixes simultaneously. Branches can be created, switched between, merged, and deleted as needed.

5. Merge: Merging combines changes from different branches into a single branch. It integrates the changes made in one branch into another. Git provides various merging strategies, such as fast-forward merge, recursive merge, and squash merge, to handle different scenarios.

6. Remote: Remote repositories are copies of the repository hosted on a different server or location. They allow for collaboration among team members by enabling them to push and pull changes from a shared remote repository.

7. Pull and Push: Pulling retrieves the latest changes from a remote repository and merges them into the current branch. Pushing sends the local commits to a remote repository, making them available to other team members.

8. Tags: Tags are used to mark specific points in the commit history as important or significant, such as release versions. They provide a way to reference a specific commit by a meaningful name.

9. Conflict Resolution: Git helps resolve conflicts that occur when multiple developers make conflicting changes to the same file or code section. It provides tools to compare and merge conflicting changes manually.

- Git offers a powerful and flexible version control system that has become the industry standard for managing source code.
- It provides a range of features and commands to support collaborative development and streamline the software development process.

2. Central vs Distributed Version Control System generate the content related to git and github

- Central Version Control System (CVCS):
  - A central version control system, also known as a centralized version control system, is a type of version control system where there is a single central repository that stores the entire project's codebase and revision history.

Here are some key characteristics of CVCS:

1. Central Repository: In a CVCS, there is a central server that hosts the main repository. Developers check out the latest version of the code from this central repository, make changes locally, and then commit their changes back to the central repository.

2. Access Control: The central server manages access control, determining who can read from and write to the central repository. Developers typically have different levels of access privileges, such as read-only or read-write access.

3. Collaboration: Developers can collaborate by sharing changes with others through the central repository. They can update their local working copies with the latest changes from the central repository and resolve any conflicts that may arise.

4. Version History: The central repository maintains a complete history of all changes made to the codebase. Developers can view the revision history, track changes, and revert to previous versions if needed.

5. Dependency on Central Server: Since all operations are performed directly on the central server, developers need a stable network connection to access the central repository. If the central server goes down, developers may not be able to perform operations or access the codebase.

- Distributed Version Control System (DVCS):
  - A distributed version control system is a type of version control system where every developer has a complete copy of the code repository, including the entire revision history.

Here are some key characteristics of DVCS:

1. Local Repository: In a DVCS, each developer maintains a local repository on their machine, which contains the entire history of the project. This allows developers to work independently, even without a network connection.

2. Decentralized Collaboration: Developers can share changes directly with each other by pushing and pulling changesets between their local repositories. This enables decentralized collaboration, where developers can work on their own branches and merge changes when ready.

3. Flexibility and Speed: DVCS allows developers to perform most operations locally, such as committing changes, creating branches, and viewing revision history. This provides greater flexibility and faster performance compared to CVCS, as network access is not required for every operation.

4. Redundancy and Backup: Since every developer has a full copy of the repository, the risk of data loss is minimized. If one developer's machine fails, the repository can be recovered from another developer's local copy.

5. Enhanced Branching and Merging: DVCS systems typically have advanced branching and merging capabilities, making it easier to create and manage

branches for different features or experiments. Merging changes from different branches is also typically faster and more efficient.

Git and GitHub:

- Git is a popular distributed version control system that provides all the features mentioned above.
- It is widely used for managing source code and enables efficient collaboration among developers.
- GitHub is a web-based hosting service for Git repositories.
- It provides a platform for hosting Git repositories in the cloud, making it easy to share and collaborate on code with others.
- GitHub offers additional features such as issue tracking, pull requests, and project management tools, which enhance the collaborative development process.
- Using Git and GitHub together, developers can leverage the power of distributed version control and take advantage of the collaborative features provided by GitHub's platform.
- They can clone repositories from GitHub, make changes locally, and then push their changes back to GitHub for others to see and collaborate on.
- GitHub also provides features for code review, documentation, and integration with various CI/CD tools, making it a popular choice for software development projects.

3. Introduction to Git

- Git is a distributed version control system designed to handle everything from small to large-scale projects with speed and efficiency.
- It allows multiple developers to collaborate on a project by providing a mechanism to track changes, manage versions, and coordinate work seamlessly.

Here are some key aspects of Git:

1. Version Control: Git tracks changes to files and directories in a project over time. It maintains a complete history of all changes made, allowing developers to revert to previous versions, view differences between versions, and understand the evolution of the project.

2. Distributed Architecture: Unlike centralized version control systems, Git follows a distributed architecture. Each developer has a complete local copy of the project, including the entire revision history. This enables developers to work independently and perform version control operations without relying on a central server.

3. Branching and Merging: Git provides powerful branching and merging capabilities. Developers can create multiple branches to work on different features or experiments. Branches can be easily created,

switched, merged, or deleted, allowing for parallel development and seamless collaboration.

4. Lightweight and Fast: Git is designed to be lightweight and fast, even with large codebases. Most operations are performed locally, requiring minimal network interaction. This results in quick response times and efficient handling of version control tasks.

5. Scalability and Performance: Git is highly scalable and can handle projects of any size. It efficiently manages and compresses data, making it suitable for large repositories with extensive histories.

6. Conflict Resolution: When multiple developers make changes to the same file, conflicts can occur during merging. Git provides tools to identify and resolve conflicts by allowing developers to manually review and modify conflicting sections.

7. Integration with Tools and Services: Git integrates with various tools and services to enhance the development workflow. It can be integrated with Continuous Integration (CI) systems, code review tools, project management platforms, and hosting services like GitHub, GitLab, and Bitbucket.

- By using Git, developers can effectively manage and track changes in their projects, collaborate with others, and maintain a reliable and organized version history.

- Its flexibility, performance, and rich feature set make it the de facto standard for version control in modern software development.

4. Installation and setting up Git

To install and set up Git, follow these steps:

1. Download Git: Visit the official Git website (https://git-scm.com/downloads) and download the appropriate version of Git for your operating system (Windows, macOS, or Linux).

2. Run the Installer: Once the download is complete, run the installer and follow the on-screen instructions. Make sure to review the options during the installation process and select any additional components or settings you require.

3. Configure Git: After the installation is complete, open a terminal or command prompt and configure your Git settings. Set your username and email address using the following commands, replacing "Your Name" and "your.email@example.com" with your actual name and email:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

- These settings are associated with your commits and will be used to identify you as the author.

4. Verify the Installation: To verify that Git is successfully installed, run the following command:

```
git --version
```

- It should display the installed Git version.

5. Optional Configuration: You may want to configure additional settings, such as your preferred text editor, line-ending handling, or credential storage. Refer to the Git documentation or online resources for more information on these configuration options.

- Once Git is installed and configured, you can start using it for version control in your projects.
- You can create new repositories, clone existing repositories, commit changes, and collaborate with others using Git commands and workflows.

5. Important Git Commands

Here are some important Git commands that you will commonly use:

1. git init: Initializes a new Git repository in the current directory.

2. git clone [repository URL]: Clones a remote Git repository to your local machine.

3. git add [file]: Adds a file to the staging area to be included in the next commit.

4. git commit -m "[commit message]": Creates a new commit with the changes in the staging area.

5. git status: Shows the current status of the repository, including modified files and untracked files.

6. git log: Displays a list of all commits in the repository, showing the commit ID, author, date, and commit message.

7. git pull: Fetches and merges changes from a remote repository into your current branch.

8. git push: Pushes your local commits to a remote repository.

9. git branch: Lists all branches in the repository.

10. git checkout [branch]: Switches to a different branch.

11. git merge [branch]: Merges changes from a specified branch into the current branch.

12. git remote: Shows the remote repositories associated with the local repository.

13. git fetch: Fetches changes from a remote repository without merging them into your local branches.

14. git diff: Shows the differences between the working directory and the staging area.

15. git reset [file]: Removes a file from the staging area.

- These are just a few of the many Git commands available.
- Git provides a powerful set of tools for version control, branch management, collaboration, and more.
- It's recommended to familiarize yourself with additional Git commands and their usage to make the most out of Git in your development workflow.

6. Creating and Managing git Repositories

To create and manage Git repositories, you can follow these steps:

1. Create a new repository:
    a. On your local machine, navigate to the desired directory where you want to create the repository.
    b. Open a terminal or command prompt in that directory.

   c. Run the command: git init. This initializes an empty Git repository in the current directory.

2. Add files to the repository:
   a. Copy or create the files you want to include in the repository into the repository's directory.

3. Run the command:
   a. git add [file] to stage the files for the next commit.
   b. Replace [file] with the specific file name or use . to add all files.

4. Commit changes:
   a. Run the command: git commit -m "[commit message]" to create a new commit with the changes.
   b. Replace [commit message] with a brief description of the changes you made.

5. Create a remote repository (optional):
   a. Sign up for an account on a Git hosting service like GitHub, GitLab, or Bitbucket.
   b. Create a new empty repository on the Git hosting service.

6. Connect the local repository to the remote repository:
   a. Run the command: git remote add origin [remote repository URL] to add the remote repository as the origin.

   b. Replace [remote repository URL] with the URL of your remote repository.

7. Push changes to the remote repository:
   a. Run the command: git push -u origin [branch] to push the committed changes to the remote repository.
   b. Replace [branch] with the name of the branch you want to push.

8. Manage the repository:
   a. Use commands like git branch, git checkout, and git merge to manage branches.
   b. Use commands like git pull, git fetch, and git push to synchronize changes with the remote repository.

- These are the basic steps to create and manage Git repositories.
- Git provides a lot more functionality and flexibility for version control, branching, collaboration, and more.
- It's recommended to explore further and learn additional Git commands and best practices to effectively use Git for your projects.

7. Branching, Merging, Stashing, Rebasing, Reverting and Resetting

- Branching, merging, stashing, rebasing, reverting, and resetting are important concepts in Git for managing changes and history in a repository.

Here's an overview of each concept:

1. Branching:
    a. Git allows you to create multiple branches to work on different features or versions of your project.
    b. Use the command: git branch [branch_name] to create a new branch.
    c. Use the command: git checkout [branch_name] to switch to a different branch.
    d. Branches help in isolating work, experimenting, and collaborating without affecting the main codebase.

2. Merging:
    a. Merging combines changes from one branch into another.
    b. Use the command: git merge [branch_name] to merge the changes from the specified branch into the current branch.
    c. Merging helps in integrating features and resolving conflicts between different branches.

3. Stashing:
    a. Stashing allows you to temporarily save your changes without committing them.
    b. Use the command: git stash to save your changes.

c. Use the command: git stash apply to apply the stashed changes back to your working directory.

4. Rebasing:
   a. Rebasing allows you to modify the commit history by moving, combining, or splitting commits.
   b. Use the command: git rebase [branch_name] to rebase your current branch onto another branch.
   c. Rebasing helps in keeping a clean and linear commit history.

5. Reverting:
   a. Reverting undoes a specific commit by creating a new commit that undoes the changes made in the original commit.
   b. Use the command: git revert [commit_id] to create a new commit that undoes the changes made in the specified commit.

6. Resetting:
   a. Resetting allows you to move the current branch pointer to a different commit, discarding some or all of the commits.
   b. Use the command: git reset [commit_id] to move the branch pointer to the specified commit.
   c. Resetting can be used to undo commits, unstage files, or remove commits from the branch history.

- These are essential operations in Git for managing changes and collaborating with others.

- Understanding and utilizing these concepts will help you effectively manage your repository and track the history of your code.
- It's important to use these commands with caution, especially when working with shared repositories, to avoid data loss or conflicts.

8. Introduction to GitHub

- GitHub is a web-based platform for version control and collaboration that utilizes Git as its underlying technology.
- It provides a centralized location for hosting and managing Git repositories, making it easier for individuals and teams to work together on software development projects.

Here's an introduction to GitHub and its key features:

1. Repository Hosting:
    a. GitHub allows you to create and host Git repositories for your projects.
    b. You can create both public and private repositories depending on your needs.
    c. Public repositories are visible to the public, while private repositories provide access control to restrict visibility and collaboration.

2. Collaboration and Social Features:

a. GitHub facilitates collaboration among developers by providing features such as pull requests, code reviews, and issue tracking.
b. Developers can submit pull requests to propose changes, and other team members can review and discuss the proposed changes before merging them into the main codebase.
c. Issue tracking allows you to create and manage tasks, bugs, and feature requests, making it easier to track progress and collaborate on problem-solving.

3. Version Control:
   a. GitHub uses Git for version control, allowing you to track changes, manage branches, and merge code changes seamlessly.
   b. Git's powerful branching and merging capabilities enable teams to work on different features or versions of a project concurrently and merge their changes efficiently.

4. Community and Open Source:
   a. GitHub has a large community of developers, making it an excellent platform for open-source projects.
   b. It provides a way for developers to contribute to existing projects by submitting pull requests and collaborating with project maintainers.
   c. GitHub also allows you to discover and explore a vast range of open-source projects, providing

opportunities for learning and contributing to the community.

5. Integration and Extensibility:
    a. GitHub offers various integrations and APIs that enable seamless integration with other tools and services.
    b. You can integrate GitHub with popular development tools like continuous integration (CI) systems, project management platforms, and deployment services to automate various aspects of your software development workflow.

- GitHub is widely used by developers and organizations for hosting, collaborating, and managing code repositories.
- It provides a robust and user-friendly interface for version control, making it easier to track changes, collaborate with others, and contribute to the open-source community.

9. Managing Remote Repositories

- Managing remote repositories in Git involves interacting with repositories hosted on remote servers, such as GitHub or GitLab.

Here are some common tasks and commands for managing remote repositories:

1. Cloning a Remote Repository:

a. To clone a remote repository to your local machine, use the git clone command followed by the repository's URL.
b. Example: git clone https://github.com/username/repo.git

2. Adding a Remote Repository:
   a. To add a remote repository to your local Git repository, use the git remote add command followed by a name for the remote and the repository's URL.
   b. Example: git remote add origin https://github.com/username/repo.git

3. Listing Remote Repositories:
   a. To list the remote repositories associated with your local Git repository, use the git remote command.
   b. Example: git remote -v

4. Fetching Changes from a Remote Repository:
   a. To fetch the latest changes from a remote repository, use the git fetch command followed by the remote name.
   b. Example: git fetch origin

5. Pulling Changes from a Remote Repository:
   a. To pull and merge the latest changes from a remote repository into your current branch, use the git pull command followed by the remote name and branch name.

b. Example: git pull origin master

6. Pushing Changes to a Remote Repository:
    a. To push your local commits to a remote repository, use the git push command followed by the remote name and branch name.
    b. Example: git push origin master

7. Renaming a Remote Repository:
    a. To rename a remote repository, use the git remote rename command followed by the old name and the new name.
    b. Example: git remote rename origin new-origin

8. Removing a Remote Repository:
    a. To remove a remote repository from your local Git repository, use the git remote remove command followed by the remote name.
    b. Example: git remote remove origin

- These commands allow you to interact with remote repositories, fetch and pull changes, push your local commits, and manage the remote repositories associated with your local Git repository.
- It enables collaboration and synchronization between your local development environment and the remote repository hosting platforms.

10. Practical
    1. Installation and Configuration of git

2. Creating Git Repositories
3. Demonstrating various Git repositories
4. Merging Branches and Managing merge conflicts
5. Stashing, Reverting, Rebasing and Resetting
6. Collaborating local and remote repositories

Title: Git Repository Management and Collaboration

Objective: The objective of this practical is to gain hands-on experience in installing and configuring Git, creating Git repositories, managing branches, resolving merge conflicts, and collaborating with local and remote repositories.

Practical Steps:

1. Installation and Configuration of Git:

Install Git on your local machine by downloading the latest version from the official Git website and following the installation instructions for your operating system. Configure your Git username and email using the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

2. Creating Git Repositories:

a. Create a new directory on your local machine for your Git repositories.
b. Open a terminal or command prompt and navigate to the directory.
c. Initialize a new Git repository using the following command:

```
git init
```

3. Demonstrating Various Git Operations:
   a. Create a new file in the repository directory and add some content to it.
   b. Use the following commands to add the file to the staging area and commit it to the repository:

```
git add <filename>
git commit -m "Initial commit"
```

c. Create a new branch using the following command:

```
git branch <branch-name>
```

d. Switch to the new branch using the following command:

```
git checkout <branch-name>
```

e. Make some changes to the file and commit them on the new branch.
f. Switch back to the main branch using the following command:

```
git checkout main
```

g. Merge the changes from the new branch into the main branch using the following command:

```
git merge <branch-name>
```

4. Merging Branches and Managing Merge Conflicts:
   a. Create another branch and make conflicting changes to the same file.
   b. Attempt to merge the conflicting branch into the main branch.
   c. Resolve any merge conflicts by editing the conflicting file.
   d. Use the following command to mark the conflicts as resolved:

```
git add <filename>
```

e. Complete the merge by committing the changes.
f. Verify that the merge was successful.

5. Stashing, Reverting, Rebasing, and Resetting:
   a. Make some changes to the file in the main branch.
   b. Stash the changes using the following command

```
git stash
```

   c. Create another branch and make some changes.
   d. Revert the changes in the new branch using the following command:
   e. Perform a rebase operation to incorporate the changes from the main branch into the new branch using the following command:

```
git rebase main
```

   f. Reset the branch to a previous commit using the following command:

```
git reset <commit-hash>
```

6. Collaborating with Local and Remote Repositories:

a. Create a new repository on a Git hosting platform such as GitHub or GitLab.
b. Connect your local repository to the remote repository using the following command:

```
git remote add origin <remote-repository-url>
```

c. Push the local repository to the remote repository using the following command:

```
git push -u origin main
```

d. Make some changes to the file in the remote repository using the online interface.
e. Pull the changes from the remote repository to your local repository using the following command:

```
git pull origin main
```

f. Collaborate with others by sharing the remote repository and performing push and pull operations.

StarAgile