Unit-3 Introduction to Project Management

The need for Source Code Control:

- Every Software Project has its own code structure in different files and folders.
- These project files must be stored somewhere so that it is available to all the teams within an organization, to work on them, constantly.
- And, in case incorrect/faulty code is written, there must be a feature which allows it to roll back the faulty changes.
- Version Control Systems, such as Git, serve those features.
- A common repository to a project is created and hosted in some web-based platforms such as GitHub, so that all of the members of the organization can access and work on the codebase, remotely.
- Changes to this common repository are made wherever the developers team pushes new code or changes the existing code.
- Hence, it is vital to maintain and manage the common repositories (projects).
- This process of controlling the source code repositories to maintain the integrity, reliability and scalability of the codebase, is known as Source Code Control.

History of Source Code Management:

Roles and Code:

- The Common repositories is the whole product and asset for an organization.
- The main Products of the organizations are hosted from these repositories.
- Hence, any organization wouldn't go easy on management of these common repositories.
- Organizations maintain strict rules and guidelines for those who are contributing to the repositories, to maintain the quality, integrity, flexibility and scalability of the code.
- Hence, it is impossible for one person/a team to manage the whole codebase alone.
- Hence, In the continuous and ever ending process of source code management, there can be various roles.
- These roles play crucial parts in ensuring the effective and collaborating development of code.
- Every role has their own responsibilities to contribute to the source code management system.
- Some of the common roles are:

1. Developers:
    - Responsibilities: Write and modify code, collaborate on features or bug fixes, and ensure that their changes align with project goals.
    - Key Actions: Committing code changes, creating branches for new features or bug fixes, and participating in code reviews.

2. Administrators:
    - Responsibilities: Manage the source code management system, configure access controls, and ensure the integrity and availability of the repositories.
    - Key Actions: Setting up repositories, managing user access, and performing system maintenance

3. QA team:
    - Responsibilities: Test and verify that the code changes meet specified requirements and do not introduce regressions or bugs.

- Key Actions: Creating and executing test cases, reporting and tracking issues, and collaborating with developers to resolve defects.

4. Project Managers:
    - Responsibilities: Oversee the overall progress of the project, manage timelines, and ensure that the development aligns with the project goals.
    - Key Actions: Planning and scheduling development tasks, tracking project milestones, and communicating with stakeholders.

5. Release Managers:
    - Responsibilities: Oversee the release process, coordinate versioning, and ensure a smooth deployment of software releases.
    - Key Actions: Tagging releases, managing release branches, and coordinating with development and operations teams.

6. Security Analysts:
    - Responsibilities: Assess the security aspects of the code, identify vulnerabilities, and propose measures to enhance the security of the software.
    - Key Actions: Conducting security reviews, identifying potential risks, and collaborating with developers to address security concerns.

7. Documentation Team:
    - Responsibilities: Create and maintain documentation related to code, APIs, and system architecture to facilitate understanding and usage.
    - Key Actions: Writing documentation, updating documentation with code changes, and ensuring documentation accuracy.

Source Code Management System and Migrations:

Shared Authentication:

Hosted Git Servers:

- Git, being a distributed version control system, allows users to host their local repositories on remote servers.
- This is achieved through the use of remote repositories, which are copies of your local repository stored on a server.
- There are several types of Git servers available on the market, including Github, GitLab, Bitbucket, etc.
- A basic Git server offers the following functionalities:

1. Web Interfaces: Web interfaces provide users with a graphical user interface (GUI) to interact with Git repositories via a web browser. These interfaces typically offer functionalities such as browsing repositories, viewing file history, managing branches, and performing Git operations like commits, pushes, and pulls. Web interfaces make Git repositories more accessible to users who may not be familiar with command-line interfaces.
2. Documentation Facility with an Inbuilt Wiki: This feature enables users to create and maintain documentation directly within the Git repository. It often includes a built-in wiki system where users can create, edit, and organize documentation pages. This wiki functionality is useful for documenting project guidelines,

development processes, coding standards, and other project-related information. By storing documentation within the repository, it becomes version-controlled and easily accessible to all team members.
3. Issue Trackers: Issue trackers are tools integrated into Git servers to manage project tasks, bugs, feature requests, and other work items. They allow users to create, assign, prioritize, and track issues throughout their lifecycle. Issue trackers typically support features like customizable issue fields, labels, milestones, comments, attachments, and search/filter capabilities. Integration with Git repositories enables referencing and linking issues to specific commits, branches, or pull requests, providing context and traceability to project changes.
4. Commit Visualization: Commit visualization tools provide graphical representations of the commit history within a Git repository. They often display the commit graph as a tree or graph structure, showing the relationships between commits, branches, and merges. Commit visualization helps users understand the project's development history, identify branching patterns, visualize merge conflicts, and track the flow of changes over time. Some Git servers offer built-in commit visualization features, while others integrate with third-party tools or provide APIs for custom visualization solutions.
5. Branch Visualization: Branch visualization tools display the branching structure of a Git repository, illustrating how branches diverge and merge over time. They visualize branch relationships, including parent-child relationships, merge commits, and branch labels. Branch visualization aids in understanding branching strategies, identifying branch dependencies, tracking feature development, and resolving merge conflicts. Git servers may include branch visualization as part of their web interfaces or integrate with external visualization tools for more advanced branching analysis.
6. The Pull-Request Workflow: The pull-request workflow is a collaborative development process commonly used in Git-based projects, particularly in distributed teams and open-source communities. It involves the following steps:

- Feature Branching: Developers create feature branches from the main development branch (e.g., master) to work on specific features or fixes.
- Commits and Changes: Developers make changes to the code in their feature branches, committing their work locally as they progress.
- Pull Request Creation: When ready, developers submit pull requests to merge their feature branches into the main development branch. Pull requests include a summary of changes, description of the feature, and any related issues or documentation updates.
- Code Review: Other team members review the pull request, providing feedback, suggestions, and approvals. Code review ensures code quality, adherence to coding standards, and consistency with project goals.
- Merge and Deployment: Once approved, the pull request is merged into the main branch, triggering automated tests and deployments. Merged changes become part of the project's history and are available to other team members.

DIfferent Git Server Implementation:

- All the Git Servers are just the servers which offer additional services on top of Git.
- Hence, A local storage hosted on a git server can be hosted on multiple Git Servers and can be manages parallelly.
- Addition of multiple remote servers can be done as follows:

```
git remote add origin1 <URL1>
git remote add origin2 <URL2>
```

- The changes can be pushed to all added remote servers, as follows:

```
git push --all
```

- A new change from one server can be updated on another server as follows:

```
git fetch origin1
git push --all origin2
```

Docker Intermission:

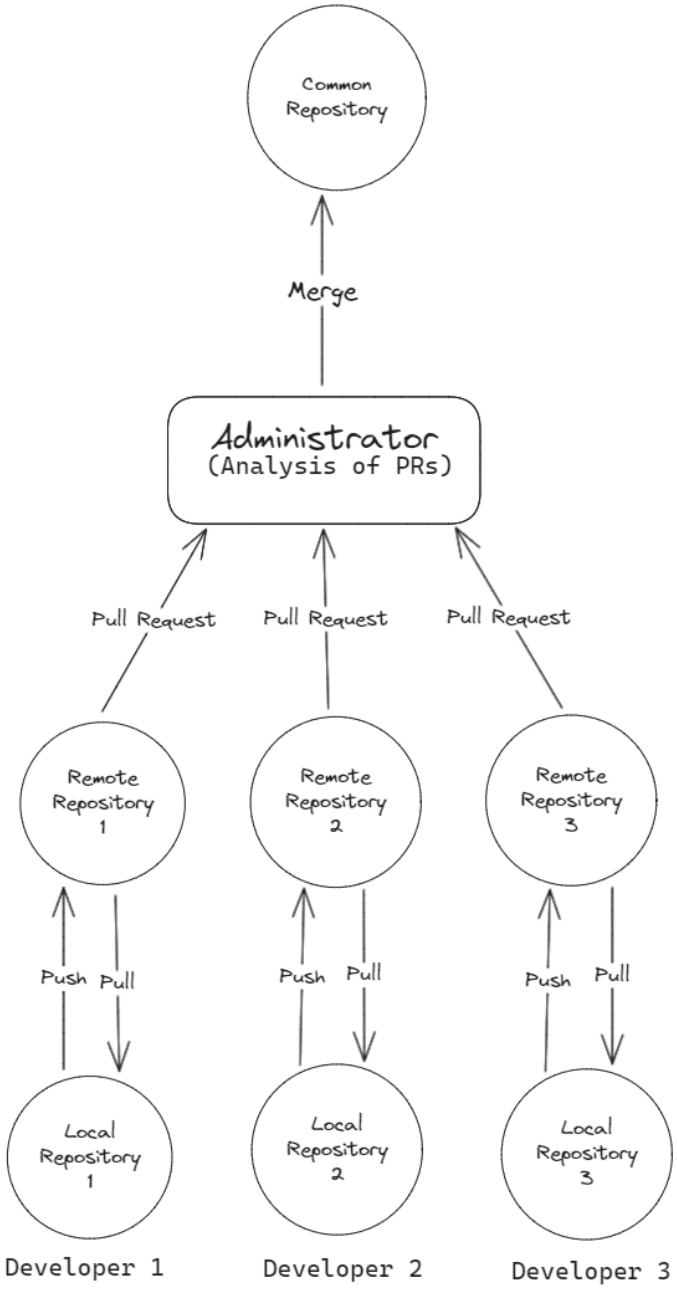
- A long-term application is built with many dependencies with specific versions and many external API links.
- When it is hosted on a repository, the sensitive links are not made available due to security reasons.
- If a new developer wants to pull the repository into a local repository and work on it, there will be lots of formalities and tasks to be done, which takes lots of time and effort.
- For Example, a Developer has to make sure that correct versions of dependencies are installed and all api links are mentioned and ensure that there are no errors while running it locally. Then and only then, the application runs successfully.
- Docker solves this problem.
- Docker is a platform that enables developers to automate the deployment of applications inside lightweight and portable containers.
- Containers are standalone and executable software packages that include everything needed to run an entire software application.
- Basically, an entire application with specific dependencies and APL links are encapsulated into a single and independent container which is both lightweight and portalble.
- And the docker container works platform independently.
I.e, if a code was developed on a computer with (x) OS, then not only new developers with the same (x) OS, but also the developers with (y) OS can run these containers.
- As a result, a developer using docker containers to run an application locally, may have no worries about ensuring dependencies and API links.
- Docker makes developers to be focused on writing great code, rather than worrying about initialization formalities(i,e, setting up the environment correctly).
- When a docker container is run, it runs a set of commands that are required to build, install and run an application.
- Those commands/instructions are written in a file called “dockerfile”.
- If an application has multiple services to run (ex: frontend, backend, database, etc), then there is a concept called “docker compose”.
- “Docker Compose” is a method used when multiple containers of an application are need to be run.
- Each container is for a single service.

Gerrit:

Pull Request Model:

- An organization has large no.of developers and teams that individually contribute to a common production repository.
- If the code works on developers’ workstation, that doesn't necessarily mean it does work on production servers too, because system configurations of server and developers’ are different.

- Hence, the code must go through various tests, before merging it into a common repository.
- Having these testing teams as an intermediate b/w developers and common repository, helps make better decisions before making it public.
- Hence, the version control system (VCS), such as Git, does the same.
- A pull request (PR) is a method of proposing changes to a codebase, reviewing those changes, and then integrating them into the main code repository.
- The PR model helps teams manage code changes in a controlled and collaborative manner.



- The workflow of PR Model is:
    1. Branching: Developers work on feature branches rather than directly on the main or development branch. This allows them to isolate their changes from the rest of the codebase until the feature or bug fix is ready.
    2. Code Changes: Developers make changes to the code on their feature branches. These changes could be new features, bug fixes, or other improvements.

3. Push to Remote Repository: Once the developer is satisfied with the changes on their local branch, they push these changes to a remote repository, typically hosted on a platform like GitHub, GitLab, or Bitbucket.
4. Pull Request Creation: The developer then creates a pull request on the platform hosting the repository. This is a request to merge their changes from the feature branch into the main or target branch.
5. Code Review: Other team members or collaborators review the code changes in the pull request. This involves checking for code quality, adherence to coding standards, potential bugs, and alignment with project requirements.
6. Discussion and Iteration: Discussions may occur within the pull request, addressing questions or concerns raised during the code review. The developer may need to make additional changes to address feedback.
7. Automated Tests: Continuous Integration (CI) systems often run automated tests on the proposed changes to ensure that the new code doesn't introduce regressions or break existing functionality.
8. Approval: Once the code review and tests pass, one or more team members approve the pull request. Some projects have specific rules about the number of approvals required before merging.
9. Merge: After approval, the changes are merged into the target branch. This can be done manually by a team member or automatically by a CI/CD (Continuous Integration/Continuous Deployment) pipeline.
10. Deployment: The merged changes can then be deployed to a staging or production environment, depending on the project's deployment practices.

GitLab:

- GitLab is a web-based platform for version control, continuous integration, and collaboration.
- It provides a complete DevOps platform that helps teams manage their software development life cycle efficiently
- Some Features of GitLab are:
    1. Version Control:
        - Git Repository: GitLab supports Git as its version control system, allowing teams to manage source code, track changes, and collaborate on projects.
        - Branching and Merging: GitLab supports branching strategies, allowing developers to work on different features or fixes concurrently. Merging changes back into the main branch is also straightforward.
    2. Collaboration:
        - Merge Requests (MRs): Similar to pull requests in other platforms, Merge Requests in GitLab allow developers to propose changes, review code, and collaborate on new features or bug fixes.
        - Code Review: GitLab provides tools for code review, including inline comments, discussions, and the ability to review and approve code changes.
    3. Continuous Integration/Continuous Deployment (CI/CD):
        - GitLab CI/CD: GitLab includes built-in CI/CD pipelines that automate the testing, building, and deployment of applications. CI/CD pipelines can be defined in a .gitlab-ci.yml file within the repository.
        - Runners: GitLab Runners are agents that execute CI/CD jobs. They can be installed on different environments, allowing flexibility in building and deploying applications.
    4. Issue Tracking:
        - Issue Boards: GitLab provides issue tracking with boards that can be customized to match your team's workflow. Issues can be organized in lists, and their status can be tracked visually.
    5. Wiki and Documentation:

- GitLab includes a built-in Wiki feature that allows teams to create and collaborate on documentation. It's a useful space for storing project-related information and guidelines.

6. Code Quality and Security:
    - Code Quality: GitLab offers tools to analyze code quality and enforce coding standards. It provides insights into code complexity, duplication, and adherence to best practices.
    - Security Scanning: GitLab includes security scanning tools for identifying vulnerabilities, dependency scanning, and container scanning to enhance the security of the development process.
7. Container Registry:
    - GitLab Container Registry allows you to store and manage Docker images within GitLab. It integrates seamlessly with GitLab CI/CD, enabling teams to build, test, and deploy containerized applications.
8. GitLab Pages:
    - GitLab Pages enables the creation of static websites directly from the repository. It's useful for hosting documentation, personal pages, or any static content associated with the project.
9. License Management:
    - GitLab provides tools to manage and enforce licenses for the dependencies used in your project, helping teams maintain compliance with open-source licenses.
10. GitLab Community Edition (CE) and Enterprise Edition (EE):
    - GitLab is available in two editions: CE (Community Edition) and EE (Enterprise Edition). While CE is open-source and free to use, EE offers additional features and support suitable for larger enterprises.