TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

Topics for today's class:

1. **What is Git**
2. **What are the different types of Git**
3. **What is VCS**
4. **What are the advantages of VCS**
5. **Functionalities of VCS**
6. **VCS vs Collaboration**
7. **Differences between Git, Github, Gitlab**

1. What is Git?

Git is a **distributed version control system (VCS)** that allows developers to track changes to files, collaborate with others, and manage multiple versions of a project over time. Git was created by **Linus Torvalds** in 2005 to handle the development of the Linux kernel.

Git tracks changes in source code, enables teams to collaborate seamlessly, and allows users to revert to previous versions if something goes wrong. It is highly efficient, even with large projects, and it operates locally (on a developer's computer) with the ability to sync with remote repositories.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

Key Features of Git:

- **Version Control**: Tracks the history of changes made to files.
- **Branching and Merging**: Allows developers to create branches for new features or bug fixes and merge them back into the main project.
- **Distributed**: Every user has a full copy of the repository, making it resilient to network failures and enabling offline work.
- **Fast and Efficient**: Git is optimized for speed and can handle large projects.

2.Git Types: (2 types)

1. Centralized Version Control System (CVCS)

CVCS is a type of version control system where a **central server** holds the entire project's version history. Developers **check out** files from the central server, work on them locally, and then **commit** their changes back to the server.

Key Characteristics of CVCS:

- **Centralized Repository**: All the source code and its history are stored in a central server. Developers retrieve the files from this server and update it with their changes.
- **Single Point of Failure**: If the central server goes down or experiences issues, no one can access the project or make any changes until the server is restored.
- **Local Work with Remote Updates**: Developers can make changes to their local copies of files but need to connect to the central repository to commit their changes. All changes are eventually synced with the server.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

- **No Full History Locally**: Developers can only see their changes and the project's history if they have access to the server. They don't have a complete copy of the project history on their local machine.

Common Examples of CVCS:

- **Subversion (SVN)**: A widely used centralized version control system.
- **CVS (Concurrent Versions System)**: One of the earliest version control systems, though now outdated.
- **Perforce**: A centralized VCS often used for large codebases, especially in game development.

Advantages of CVCS:

- **Simpler to Manage**: Easier to configure and manage for smaller teams or projects because everything is centralized in one place.
- **Fewer Copies**: Only one copy of the entire repository is maintained, reducing storage overhead.
- **Access Control**: Centralized control over the repository allows for easier enforcement of access permissions.

Disadvantages of CVCS:

- **Single Point of Failure**: If the central server crashes, development stops until the server is restored.
- **Network Dependency**: Developers need to be online and connected to the central server to commit changes or retrieve the latest versions of files.
- **Limited Offline Work**: You cannot view the full history or make commits unless connected to the central server.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

2. Distributed Version Control System (DVCS)

DVCS is a version control system in which every **developer's local machine** contains a **full copy of the repository**, including the project's history. Changes made on each local copy can be shared with others by **pushing** and **pulling** changes to and from other repositories.

Key Characteristics of DVCS:

- **Distributed Repositories**: Every developer has a full copy of the repository (including its entire history) on their local machine. The main repository (often called the **remote repository**) is only one of many, and all developers can work on their local copies of the project.
- **Offline Work**: Developers can work offline without needing to be connected to a central server. They can make commits, view history, and create branches locally.
- **Merging and Syncing**: Changes are made locally and can be pushed or pulled to/from the remote repository when needed. Developers can also merge their local changes with others' changes.
- **No Single Point of Failure**: Since every developer has a complete copy of the repository, if one developer's machine or the remote repository fails, others can continue working with their own local copies.

Common Examples of DVCS:

- **Git**: The most popular distributed version control system, known for its speed, flexibility, and ability to handle large projects.
- **Mercurial**: Another distributed VCS, similar to Git but with a simpler and more straightforward design.
- **Bazaar**: A distributed version control system that is simple to use and works well for small teams and individuals.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

Advantages of DVCS:

- **Offline Work**: Developers can work offline, make commits, and access the project history at any time.
- **No Single Point of Failure**: Every developer has a full copy of the project, so the system is more resilient to server failures.
- **Branching and Merging**: DVCS systems like Git make it easy to create branches for features, bug fixes, or experimentation, and merge them back into the main project.
- **Faster Operations**: Since most operations (like commits, viewing history, etc.) happen locally, they are faster than in CVCS.

Disadvantages of DVCS:

- **More Complex Setup**: DVCS systems can be more complex to set up and use, especially for beginners. The need to manage multiple repositories and branches can be confusing.
- **Storage Overhead**: Since every developer has a full copy of the repository, it can take up more storage space than a CVCS.
- **Synchronization**: Developers need to regularly push and pull changes to ensure they are working with the latest version of the project. This can lead to conflicts if changes are not properly merged.

3. What is VCS (Version Control System)?

A **Version Control System (VCS)** is a tool that helps software developers manage changes to source code and other files in a project. VCS allows developers to track, manage, and revert to previous versions of files, making collaboration easier and safer.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

There are two main types of VCS:

- **Local Version Control Systems (LVCS)**: Tracks changes to files locally, on a developer's own machine. Earlier systems like **RCS (Revision Control System)** were LVCS.
- **Distributed Version Control Systems (DVCS)**: Unlike LVCS, a DVCS keeps track of changes locally and also maintains a full repository on the server. Git is a widely used DVCS, as is **Mercurial**.

Common VCS Tools:

- **Git**
- **Mercurial**
- **Subversion (SVN)**
- **CVS (Concurrent Versions System)**

4. What Are the Advantages of VCS?

Using a VCS provides several advantages:

- **Version History**: Every change made to the code is recorded, allowing developers to track changes, identify bugs, and revert to previous versions if necessary.
- **Collaboration**: Developers can work on the same project simultaneously, merge their changes, and resolve conflicts effectively.
- **Backup and Redundancy**: With VCS, your project history is stored in multiple places. In distributed systems like Git, everyone has a copy of the entire project and history, ensuring no single point of failure.
- **Branching and Merging**: Branching allows developers to create isolated environments for new features or bug fixes. When the work is complete, it can be merged back into the main codebase.
- **Auditing**: You can see who made each change, when it was made, and why (through commit messages). This makes it easier to trace issues or understand the rationale behind certain code changes.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

- **Better Code Quality**: VCS allows for code reviews, peer collaboration, and structured workflows that encourage better coding practices.

5. Functionalities of VCS

A VCS provides the following core functionalities:

- **Commit Changes**: Save changes to the local repository, including messages that explain what was done.
- **Branching**: Create separate branches for development to avoid disrupting the main project while new features or bug fixes are being worked on.
- **Merging**: Combine changes from different branches into the main branch or other branches.
- **Revert Changes**: Undo changes to the codebase, either by resetting to an earlier state or discarding specific commits.
- **Cloning**: Create a local copy of a remote repository.
- **Pull/Pull Requests**: Fetch changes from a remote repository and integrate them into the local copy. Pull requests are used to review code before merging.
- **Conflict Resolution**: Handle conflicts when two developers modify the same part of a file simultaneously.
- **Tagging**: Mark specific points in history, such as releases or stable versions.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

6. VCS vs Collaboration

ASPECT	VERSION CONTROL SYSTEM(VCS)	COLLABORATION
Definition	A software tool that tracks and manages changes to source code or files.	The act of working together on tasks and sharing resources.
Main Purpose	To keep track of changes, versions, and history of files.	To enable multiple individuals to work together towards a common goal.
Core Focus	Managing versions, history, and ensuring consistency across codebases.	Coordinating team efforts, communication, and resource sharing.
Key Activities	Commit, push, pull, branch, merge, rebase, revert, tag.	Discussion, decision-making, task assignment, sharing of ideas and resources.
Scope of Use	Primarily used for code or file versioning, backups, and change tracking.	Used in a broader context like team projects, business collaboration, document sharing, etc.
Tools	Git, SVN, Mercurial, CVS, Perforce.	Slack, Microsoft Teams, Zoom, Google Docs, Confluence.
Access Control	VCS controls access to files through permissions and authentication.	collaboration often relies on project managers or team leaders for managing access.
Backup and History	Provides a backup of the project at every commit, making it easy to revert to previous versions.	Collaboration relies on shared understanding, but does not inherently store history unless backed by VCS.
Error Recovery	VCS allows easy recovery of previous versions or fixes through commits and branches.	Collaboration may require additional effort to identify and fix errors if there's no VCS in place.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

7. Difference between Git, Github, Gitlab.

Feature	Git	GitHub	GitLab
Type	Distributed Version Control System (DVCS)	Git repository hosting platform with social and collaboration features	Git repository hosting platform with integrated DevOps tools
Primary Purpose	Version control for tracking changes in source code and files.	Hosting Git repositories, collaboration, code sharing, and version control.	Hosting Git repositories, CI/CD pipelines, code reviews, and DevOps integration.
Ownership	Open-source, self-hosted, managed by the user or organization.	Owned by Microsoft (acquired in 2018).	Owned by GitLab Inc., an open-source platform.
Hosting	Requires hosting on your server or on platforms like GitHub, GitLab, etc.	Cloud-hosted, no need to manage servers.	Can be self-hosted or used on GitLab's cloud platform.
Version Control System	Core version control system (manages code history and versions).	Built on Git, with additional tools for collaboration.	Built on Git, with tools for CI/CD, issue tracking, and automation.
Collaboration Features	None, purely version control.	Pull requests, issues, discussions, code reviews, and project boards.	Merge requests, issues, wikis, continuous integration, project boards, and code reviews.
User Interface	Command-line interface (CLI), or third-party graphical tools (e.g., Sourcetree, GitKraken).	Web interface with easy-to-use UI for repository management, issues, and pull requests.	Web interface with built-in tools for CI/CD, issues, and monitoring.

TASKS ASSIGNED FOR TODAY

05-03-2025
Suneetha.V

Pricing	Free (open-source).	Free with limitations; paid plans for additional features.	Free for basic features; paid plans for additional features and self-hosted options.
Integration with CI/CD	Does not provide any built-in CI/CD features.	GitHub Actions (for CI/CD) and integrations with external CI/CD tools.	Native CI/CD integration (GitLab CI) for building, testing, and deploying code.
Issue Tracking	Not available natively, but can be integrated with other tools.	GitHub Issues for bug tracking, task management, and project management.	GitLab Issues for bug tracking, agile planning, and project management.
Access Control / Permissions	Managed through the repository hosting service (e.g., GitHub, GitLab).	GitHub teams, organizations, and granular permissions for repos.	Granular permissions, with the ability to manage access at various levels (group, project, etc.).
Code Review	Not supported directly in Git; done using external tools.	Code review through pull requests, inline comments, and discussions.	Comprehensive code review features with inline comments and approval workflows.
API	Not applicable (Git doesn't provide API directly).	GitHub API for interacting with repos, issues, pull requests, etc.	GitLab API for interacting with repos, issues, CI/CD pipelines, and more.