Unit 1

Definition of Software Engineering, Software Development Life Cycle (SDLC) – Phases, Traditional vs Agile Models (Waterfall, Agile, DevOps), Scrum Basics – Roles, Sprint, Backlog, Version Control using Git and GitHub, Introduction to Project Tools (GitHub Projects, Jira, Trello).

Git and GitHub

1. Git Basics.
2. Git Installation
3. Git Basic Commands
4. Experiments

Version Control System using Git and GitHub

What is Version Control System (VCS)?

A Version Control System (VCS) is a tool used in software development and collaborative projects to track and manage changes to source code.
It allows developers to:

- Record and track every update to the codebase.
- Collaborate on code without overwriting each other's work.
- Revert to earlier states of the project if needed.
- Maintain a detailed and structured history of the project's evolution.

A Version Control System (VCS), also commonly referred to as a Source Code Management (SCM) system, is a software tool or system that helps manage and track changes to files and directories over time. The primary purpose of a VCS is to keep a historical record of all changes made to a set of files, allowing multiple people to collaborate on a project while maintaining the integrity of the codebase. There are two main types of VCS: centralized and distributed.

Centralized Version Control Systems (CVCS): In a CVCS, there is a single central repository that stores all the project files and their version history. Developers check out files from this central repository, make changes, and then commit those changes back to the central repository. Examples of CVCS include CVS (Concurrent Versions System) and Subversion (SVN).

Distributed Version Control Systems (DVCS): In a DVCS, every developer has a complete copy of the project's repository, including its full history, on their local machine. This allows developers to work independently, create branches for experimentation, and synchronize their changes with remote repositories. Git is the most well-known and widely used DVCS, but other DVCS options include Mercurial and Bazaar.

Key features and benefits of Version Control Systems include:

1. History Tracking: VCS systems maintain a complete history of changes, including who made the change, what was changed, and when it was changed. This makes it easy to review and understand the evolution of a project.
2. Collaboration: VCS allows multiple developers to work on the same project simultaneously. It provides mechanisms for merging changes made by different contributors and resolving conflicts when they occur.

3. Branching and Isolation: VCS systems support branching, allowing developers to create isolated environments for new features or bug fixes. This isolates changes and helps manage complex development tasks.
4. Revert and Rollback: If a mistake is made, it is possible to revert changes to a previous state or commit. This is essential for error correction and maintaining code quality.
5. Backup and Recovery: Project data is stored in multiple locations, providing redundancy and facilitating data recovery in case of accidental data loss or system failures.
6. Documentation: Commit messages and history serve as a form of documentation, explaining why a change was made, who made it, and when it was made.
7. Efficiency: VCS systems are designed to be fast and efficient. They typically store only the changes made to files, rather than entire file copies, which results in small repository sizes and faster operations.

VCS is a fundamental tool in software development and is used not only for source code but also for tracking changes in documentation, configuration files, and other types of text-based files. It is especially crucial for collaborative projects, allowing teams of developers to work together on the same codebase with confidence.

What is Git?

Git is a distributed version control system (VCS) that is widely used for tracking changes in source code during software development. It was created by Linus Torvalds in 2005 and has since become the de facto standard for version control in the software development industry.Git allows multiple developers to collaborate on a project by providing a history of changes, facilitating the tracking of who made what changes and when. Here are some key concepts and features of Git:

1. **Repository (Repo): A** Git repository is a directory or storage location where your project's files and version history are stored. There can be a local repository on your computer and remote repositories on servers.
2. **Commits**: In Git, a commit is a snapshot of your project at a particular point in time. Each commit includes a unique identifier, a message describing the changes, and a reference to the previous commit.
3. **Branches:** Branches in Git allow you to work on different features or parts of your project simultaneously without affecting the main development line (usually called the "master" branch). Branches make it easy to experiment, develop new features, and merge changes back into the main branch when they are ready.
4. **Pull Requests (PRs):** In Git-based collaboration workflows, such as GitHub or GitLab, pull requests are a way for developers to propose changes and have them reviewed by their peers. This is a common practice for open-source and team-based projects.
5. **Merging**: Merging involves combining changes from one branch (or multiple branches) into another. When a branch's changes are ready to be incorporated into the main branch, you can merge them.
6. **Remote Repositories:** Remote repositories are copies of your project stored on a different server. Developers can collaborate by pushing their changes to a remote repository and pulling changes from it. Common remote repository hosting services include GitHub, GitLab, and Bitbucket.
7. **Cloning:** Cloning is the process of creating a copy of a remote repository on your local machine. This allows you to work on the project and make changes locally.
8. **Forking:** Forking is a way to create your copy of a repository, typically on a hosting platform like GitHub. You can make changes to your fork without affecting the original project and later create pull requests to contribute your changes back to the original

repository.

Git is known for its efficiency, flexibility, and ability to handle both small and large-scale software projects. It is used not only for software development but also for managing and tracking changes in various types of text-based files, including documentation and configuration files. Learning Git is essential for modern software development and collaboration.

Why we need git?

Git is an essential tool in software development and for many other collaborative and version- controlled tasks. Here are some key reasons why Git is crucial:

1. Version Control: Git allows you to track changes in your project's files over time. It provides a complete history of all changes, making it easy to understand what was done, when it was done, and who made the changes. This is invaluable for debugging, auditing, and collaboration.
2. Collaboration: Git enables multiple developers to work on the same project simultaneously without interfering with each other's work. It provides mechanisms for merging changes made by different contributors and resolving conflicts when they occur.
3. Branching: Git supports branching, which allows developers to create isolated environments for developing new features or fixing bugs. This is essential for managing complex software projects and experimenting with new ideas without affecting the main codebase.
4. Distributed Development: Git is a distributed version control system, meaning that every developer has a complete copy of the project's history on their local machine. This provides redundancy, facilitates offline work, and reduces the reliance on a central server.
5. Backup and Recovery: With Git, your project's history is distributed across multiple locations, including local and remote repositories. This provides redundancy and makes it easy to recover from accidental data loss or system failures.
6. Code Review: Git-based platforms like GitHub, GitLab, and Bitbucket provide tools for code review and collaboration. Developers can propose changes, comment on code, and discuss improvements, making it easier to maintain code quality.
7. Open Source and Community Development: Git has become the standard for open-source software development. It allows anyone to fork a project, make contributions, and create pull requests, which makes it easy for communities of developers to collaborate on a single codebase.
8. Efficiency: Git is designed to be fast and efficient. It only stores the changes made to files, rather than entire file copies, which results in small repository sizes and faster operations.
9. History and Documentation: Git's commit history and commit messages serve as a form of documentation. It's easier to understand the context and reasoning behind a change by looking at the commit history and associated messages.
10. Customizability: Git is highly configurable and extensible. You can set up hooks and scripts to automate workflows, enforce coding standards, and integrate with various tools.

Git is essential for tracking changes in your projects, facilitating collaboration among developers, and ensuring the integrity and version history of your code. Whether you're working on a personal project or as part of a large team, Git is a fundamental tool for modern software development and version control.

1. **Setting Up and Basic Commands**
   Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

2. **Creating and Managing Branches**
   Create a new branch named "feature-branch." Switch to the "master" branch. Merge the"feature-branch" into "master."

3. **Creating and Managing Branches**
   Write the commands to stash your changes, switch branches, and then apply the stashed changes.

4. **Collaboration and Remote Repositories**
   Clone a remote Git repository to your local machine.

5. **Collaboration and Remote Repositories**
   Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

6. **Collaboration and Remote Repositories**
   Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

7. **Git Tags and Releases**
   Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
   /

8. **Advanced Git Operations**
   Write the command to cherry-pick a range of commits from "source-branch" to the current.

9. **Analyzing and Changing Git History**
   Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

10. **Analyzing and Changing Git History**
    Write the command to list all commits made by the author "JohnDoe" between "2023-01-01"and "2023-12-31."

11. **Analyzing and Changing Git History**
    Write the command to display the last five commits in the repository's history.

12. **Analyzing and Changing Git History**
    Write the command to undo the changes introduced by the commit with the ID "abc123".

Git Life Cycle

The Git lifecycle refers to the typical sequence of actions and steps you take when using Git to manage your source code and collaborate with others. Here's an overview of the Git lifecycle:

1. **Initializing a Repository:**
   - To start using Git, you typically initialize a new repository (or repo) in your project directory. This is done with the command git init.
2. **Working Directory:**
   - Your project files exist in the working directory. These are the files you are actively working on.
3. **Staging:**
   - Before you commit changes, you need to stage them. Staging allows you to select which changes you want to include in the next commit. You use the git add command to stage changes selectively or all at once with git add ..
4. **Committing:**
   - After you've staged your changes, you commit them with a message explaining what you've done. Commits create snapshots of your project at that point in time. You use the git commit command to make commits, like git commit -m "Add new feature".
5. **Local Repository:**
   - Commits are stored in your local repository. Your project's version history is preserved there.
6. **Branching:**
   - Git encourages branching for development. You can create branches to work on new features, bug fixes, or experiments without affecting the main codebase. Use the git branch and git checkout commands for branching.
7. **Merging:**
   - After you've completed work in a branch and want to integrate it into the main codebase, you perform a merge. Merging combines the changes from one branch into another. Use the git merge command.
8. **Remote Repository:**
   - For collaboration, you can work with remote repositories hosted on servers like GitHub, GitLab, or Bitbucket. These repositories serve as a central hub for sharing code.
9. **Pushing:**
   - To share your local commits with a remote repository, you push them using the git push command. This updates the remote repository with your changes.
10. **Pulling:**
    - To get changes made by others in the remote repository, you pull them to your local repository with the git pull command. This ensures that your local copy is up to date.
11. **Conflict Resolution:**
    - Conflicts can occur when multiple people make changes to the same part of a file. Git will inform you of conflicts, and you must resolve them by editing the affected files manually.
12. **Collaboration:**
    - Developers can collaborate by pushing, pulling, and making pull requests in a shared remote repository. Collaboration tools like pull requests are commonly used on platforms like GitHub and GitLab.
13. **Tagging and Releases:**
    - You can create tags to mark specific points in the project's history, such as version releases. Tags are useful for identifying significant milestones.
14. **Continuous Cycle:**
    - The Git lifecycle continues as you repeat these steps over time to manage the

ongoing development and evolution of your project. This cycle supports collaborative and agile software development.

The Git lifecycle allows for effective version control, collaboration, and the management of complex software projects. It provides a structured approach to tracking and sharing changes, enabling multiple developers to work together on a project with minimal conflicts and a clear history of changes.

1. Git Commands List

Git is a popular version control system used for tracking changes in software development projects. Here's a list of common Git commands along with brief explanations:

1. git init: Initializes a new Git repository in the current directory.
2. git clone <repository URL>: Creates a copy of a remote repository on your local machine.
3. git add <file>: Stages a file to be committed, marking it for tracking in the next commit.
4. git commit -m "message": Records the changes you've staged with a descriptive commit message.
5. git status: Shows the status of your working directory and the files that have been modified or staged.
6. git log: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.
7. git diff: Shows the differences between the working directory and the last committed version.
8. git branch: Lists all branches in the repository and highlights the currently checked- out branch.
9. git branch <branchname>: Creates a new branch with the specified name.
10. git checkout <branchname>: Switches to a different branch.
11. git merge <branchname>: Merges changes from the specified branch into the currently checked-out branch.
12. git pull: Fetches changes from a remote repository and merges them into the current branch.
13. git push: Pushes your local commits to a remote repository.
14. git remote: Lists the remote repositories that your local repository is connected to.
15. git fetch: Retrieves changes from a remote repository without merging them.
16. git reset <file>: Unstages a file that was previously staged for commit.
17. git reset --hard <commit>: Resets the branch to a specific commit, discarding all changes after that commit.
18. git stash: Temporarily saves your changes to a "stash" so you can switch branches without committing or losing your work.
19. git tag: Lists and manages tags (usually used for marking specific points in history, like releases).
20. git blame <file>: Shows who made each change to a file and when.
21. git rm <file>: Removes a file from both your working directory and the Git repository.
22. git mv <oldfile> <newfile>: Renames a file and stages the change.

These are some of the most common Git commands, but Git offers a wide range of features and options for more advanced usage. You can use git --help followed by the command name to get more information about any specific command, e.g., git help commit.

Introduction to Project Tools

JIRA is a software development tool used for **project management** and **issue tracking**. It is a popular tool among software development teams to plan, track, and release software projects. JIRA provides a centralized platform for managing tasks, bugs, and other types of issues and it helps teams to organize and prioritize their work. The tool integrates with other software development tools and has a variety of customizable features and workflows that allow teams to adapt it to their specific needs. Additionally, JIRA also provides various reporting and dashboard features that help teams stay on top of their work and make data-driven decisions.

What is JIRA used for?

JIRA provides a centralized platform for managing tasks, bugs, and other types of issues, and it helps teams organize and prioritize their work. JIRA is designed for **agile software development teams** and it supports multiple methodologies such as **Scrum, Kanban**, and **custom workflows.**

JIRA is used for:

1. **Project Management:** JIRA provides a centralized platform for managing software development projects, with support for multiple projects and workflows.
2. **Task Management:** Teams can create, assign, and track tasks, bugs, and other types of issues.
3. **Agile Planning:** JIRA supports agile methodologies such as Scrum and Kanban and provides tools for planning and tracking sprints, backlogs, and releases.
4. **Reporting and Dashboards:** JIRA provides various reports and dashboards that help teams get a real-time view of their work and make data-driven decisions.
5. **Collaboration:** JIRA allows teams to collaborate and communicate effectively, with features such as comments, notifications, and alerts.

JIRA is widely used by software development teams of all sizes and is known for its ease of use and customization capabilities. The tool offers a free trial and several pricing plans, including a **cloud-based version** and an **on-premise version**.

Four main sections are there:

- **Roadmap:** It is an action plan for how our project will evolve.
- **Backlog:** It is usually a list of issues describing what your team will be doing on the project.
- **Board:** It shows work to be done, work in progress, and work done.
- **Code:** In this, we can create an automated DevOps workflow and minimize context switching between Jira Software and Bitbucket, GitHub, GitLab, and other source code management tools.

Generally, a project is divided into epics and one epic can contain multiple sprints. Each sprint has multiple issues like user stories. A sprint is a fixed period (it can vary from project to project but in a single project, the duration of each sprint should be the same) in a development cycle where the project team completes work from the product backlog. JIRA's user stories are concise representations of the system's evolving capabilities, written by people who need that new functionality.

Now we have to create an epic first. So click on **create** select the issue type as epic fill in other requirements and create your first epic. Now create another issue "user story" to implement the first functionality for example register functionality of your project. Now move this issue to the already made sprint and start the sprint. Now your first sprint has started. We can also assign epic to the different issues.

Trello

Trello is an extremely useful project management tool that allows users to streamline

their projects. It does so with a user-friendly interface that makes task organization simple and intuitive. At its core, Trello uses boards to represent individual projects. Within these boards you can create lists that signify different stages, giving you a clear path for progression. Every task or idea is then represented as a card, allowing you to easily see what components make up your project.

While the cards themselves help track progress, Trello also lets users add comments, attachments, and deadlines to them. This level of collaboration ensures everyone is on the same page while also fostering accountability in team members.

Components

Trello is made up on 4 main components:

- Workspaces
- Boards
- Lists
- Cards

And the bonus, 5th component is the Board Menu

Features of Trello

Project management requires many steps, but Trello has plenty of features to keep things organized. Here are some of its key components:

1. Boards, Cards, and Lists:

Trello employs boards, cards, and lists to graphically represent projects. This provides an easy way to organize or track tasks. The users can therefore quickly understand the status of various components.

2. Subtasks and Checklists:

Within a card, some subtasks and checklists enable it to be divided into small steps. In this regard, clarity is improved while making complex tasks attainable. Thus each part may be sequentially completed creating efficiency.

3. Task Allocation and Notifications:

Multiple team members can be assigned tasks on Trello, allowing for different assignments. Once shared among all recipients when these cards get updated in any way they receive notifications as well. It also promotes cooperation by keeping everybody informed of what is happening within the group. Additionally, this activity generates ownership within a given unit.

4. Deadline Management:

Deadlines are important because they help us plan our time better thus ensuring we meet our targets on time at all times. Your group has to finish milestones within schedule hence you will notice that Trello allows you to set deadlines for assignments to facilitate this process. This ability makes sure every person remains attentive to details resulting in the completion of projects before deadlines arrive.

5. Activity Log:

An activity log keeps track of changes made on cards so that your employees are aware of the progress of work being done towards project completion which encouraging teamwork and open lines of communication between people working together at any level in the organization.

6. Attachments and Integrations:

Trello’s attachment and integration options make it easy to organize resources. It is possible to simply add documents and files to cards. Moreover, Trello has a seamless connection with tools like Google Drive, Slack, and GitHub which is very convenient; this diversity in the tool enhances productivity in various workflows.

7. Power-Ups:

These are Trello’s plugins, which enable users to improve their boards by adding more features and capabilities. Power-ups power up Trello by allowing users to integrate it with

other tools by adding custom fields.

8. Calender View:

The calendar view of Trello gives an all-encompassing display of due dates and timelines for tasks or projects as a help for prioritization purposes. This feature helps task prioritization, enabling the user to efficiently manage his or her workload.

9. Search and Filtering:

Through the use of its powerful search feature, Card searching can be done fast without any hassle like boards, labels, or even teammates. Furthermore, cards that meet specific criteria such as labels, due dates among others can be filtered which allows for an organized information retrieval system.