UNIT I – INTRODUCTION TO SOFTWARE ENGINEERING [9 hours]

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

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

- Manage projects with Repositories
- Clone a project to work on a local copy
- Control and track changes with Staging and Committing
- Branch and Merge to allow for work on different parts and versions of a project
- Pull the latest version of the project to a local copy
- Push local updates to the main project Working with Git
- Initialize Git on a folder, making it a Repository
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered modified
- You select the modified files you want to Stage
- The Staged files are Committed, which prompts Git to store a permanent snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes

made in each

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

Define repository:

A repository is a directory (aka folder) where you are saving all the code files for any given project. We can make a folder and then use git init to make it into a git repository.

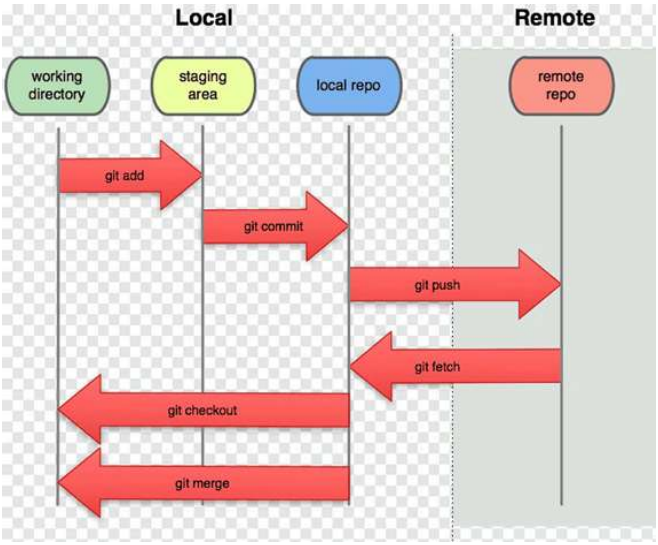
Branching

"On branch master" (means on the main project, not a side branch) to add to main branch use -m to add to side branch use –t

Git Repository Structure

It consists of 4 parts:

1. **Working directory:** This is your local directory where you make the project (write code) and make changes to it.
2. **Staging Area (or index):** this is an area where you first need to put your project before committing. This is used for code review by other team members.
3. **Local Repository:** this is your local repository where you commit changes to the project before pushing them to the central repository on Github. This is what is provided by the distributed version control system. This corresponds to the .git folder in our directory.
4. **Central Repository:** This is the main project on the central server, a copy of which is with every team member as a local repository.



Difference between Git and GitHub

Git is a version control tool (software) to track the changes in the source code. GitHub is a web-based cloud service to host your source code(Git repositories). It is a centralized system. Git doesn't require GitHub but GitHub requires Git.

Installation of Git

There are two ways of installing Git.

1. Install Git for using WSL (Windows Subsystem for Linux) Install Ubuntu – https://www.youtube.com/watch?v=X-DHaQLrBi8

a. Install Git in Ubuntu – https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-20-04

2. Install Git software for windows – https://git-scm.com/download/win

a. Go ahead with whichever method is comfortable for you. Continue further once you are done with the installation.

Git operations and commands

Before deep-diving into Git operations and commands, create an account for yourself on GitHub if you don't have it already.

Git Commands: Working With Local Repositories

1. **git init** The command git init is used to create an empty Git repository. After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the

repository is initialized, the process of creating other files begins

Syntax: $git init

2. **git add** Add command is used after checking the status of the files, to add those files to the staging area. Before running the commit command, "git add" is used to add any new or modified files.

   Syntax: $git add

3. **git commit** The commit command makes sure that the changes are saved to the local repository. The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

   Syntax: git commit -m “commit message”

4. **git status** The git status command tells the current state of the repository. The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

   Syntax: $git status

5. **git config** The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository. When git config is used with --global flag, it writes the settings to all repositories on the computer.

   Syntax: git config --global user.name “any user name” git config --global user.email <email id>

6. **git branch** The git branch command is used to determine what branch the local repository is on. The command enables adding and deleting a branch.

   Syntax: # Create a new branch

   git branch <branch_name>

   # List all remote or local branches

   git branch –a

   # Delete a branch

   git branch -d <branch_name>

7. **git checkout** The git checkout command is used to switch branches, whenever the work is to be started on a different branch. The command works on three separate entities: files, commits, and branches.

   Syntax:

   # Checkout an existing branch

   git checkout <branch_name>

   # Checkout and create a new branch with that name

   git checkout -b <new_branch>

8. **git merge** The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch. It is used to merge the changes in the staging branch to the stable branch.

   Syntax: git merge <branch_name>

Git Commands: Working With Remote Repositories

1. **git remote** The git remote command is used to create, view, and delete connections to other repositories.

   Syntax: $ git remote add origin <address>

2. **git clone** The git clone command is used to create a local working copy of an existing remote repository. The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

   Syntax: $ git clone <remote_URL>

3. **git pull** The git pull command is used to fetch and merge changes from the remote repository to the local repository. The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

   Syntax: $git pull <branch_name> <remote URL>

4. **git push** The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository. The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

   Syntax: $ git push -u origin master

5. **git branch** is a new/separate version of the main repository.

Accessing Github central repository via HTTPS or SSH

Here, transfer project means transfer changes as git is very lightweight and works on changes in a project. It internally does the transfer by using Lossless Compression Techniques and transferring compressed files. Https is the default way to access Github central repository.

- By git remote add origin http_url: remote means the remote central repository. Origin corresponds to your central repository which you need to define (hereby giving HTTPS URL) in order to push changes to Github.
- Via SSH: connect to Linux or other servers remotely.

INTRODUCTION TO PROJECT TOOLS

Project tools are software applications used to help teams plan, manage, track, and deliver projects effectively. They centralize all project-related information—tasks, documents, communication, schedules—so everyone knows what to do and when to do it. These tools support all phases of a project such as

- Requirements gathering
- PlanningExecution
- Monitoring and control
- Delivery
- Maintenance

Project tools typically offer several key features that help teams plan, organize, and manage their work effectively. They provide task management capabilities such as creating, assigning, and tracking tasks through lists, boards, or timelines. Most tools include collaboration features, allowing team members to communicate through comments, file sharing, and real-time updates. Scheduling and time management are supported through calendars, reminders, and deadline tracking. Many project tools also include progress tracking features like dashboards, status indicators, and reports that give visibility into project health. Additionally, they offer documentation support, enabling teams to store notes, requirements,

and project details in one place. Some tools provide integration capabilities, allowing them to connect with communication platforms, code repositories, or design tools. Advanced project tools may also feature resource management, automation, and workflow customization to streamline processes and improve productivity.

Why Project Tools Are Important — Summarized Sub-topics

Centralized Information:

Project tools store all project data—tasks, documents, deadlines, and communication—in a single place, which reduces confusion and helps everyone access the same information quickly.

Effective Collaboration:

These tools allow team members to communicate, share files, and update work in real time, ensuring smooth teamwork and reducing the need for long email chains or meetings.

Real-Time Tracking:

They provide dashboards and progress indicators that show what is completed, what is in progress, and what is delayed, helping managers make timely decisions.

Time and Resource Management:

Project tools help in setting schedules, estimating work, assigning responsibilities, and balancing workloads, which improves overall efficiency and prevents overload.

Improved Accountability:

Each task is assigned to a specific person with clear deadlines, ensuring responsibility is visible and making it easier to track who is doing what.

Documentation and Audit Trail:

Every change, update, and decision gets recorded automatically, which helps with future reference, knowledge sharing, and project audits.

Category of Project tools

1. Project Management Tools

These tools help teams plan, organize, and track project activities. They provide features like task assignment, schedules, boards, timelines, and progress tracking. They ensure the project stays on schedule and everyone knows what to do. **Examples:** Jira, Trello, Asana, Monday.com, and Microsoft Project.

2. Communication Tools

These tools enable smooth interaction among team members through messaging, voice/video calls, and channel-based discussions. They help teams share updates quickly and stay connected, even when working remotely. **Examples:** Slack, Microsoft Teams, Zoom, and Google Meet.

3. Collaboration & Documentation Tools

These tools allow teams to create, store, and edit documents, notes, designs, and ideas together in real time. They ensure everyone has access to updated information and support brainstorming and knowledge sharing. **Examples:** Confluence, Google Docs, Notion, Miro, and Figma.

4. Version Control & Code Management Tools

Used mainly by software teams, these tools help manage code changes, track updates, and collaborate on software development. They store code safely and make it easy to work on different parts of a project at the same time. **Examples:** GitHub, GitLab, and Bitbucket.

5. Testing & Quality Assurance Tools

These tools help teams test products, find bugs, and monitor quality. They support manual and automated testing, making sure the final product is stable and functions correctly before release. **Examples:** Selenium, Postman, TestRail, and Jira (for bug tracking).

6. Time Tracking & Productivity Tools

These tools track the time team members spend on tasks and help measure productivity. They support project estimation, workload planning, and time-based reporting, which is useful for billing and performance analysis. **Examples:** Toggl, Clockify, and Harvest.

Benefits

- Increased Productivity: Automate tasks, reduce manual work.
- Better Visibility: Centralized view of project status for everyone.
- Streamlined Communication: Less miscommunication, clearer channels.
- Faster Decisions: Quick access to data supports prompt choices.
- Risk Reduction: Early identification of issues.

GITHUB PROJECTS

GitHub Projects is a project management tool integrated within GitHub, designed to help development teams organize, track, and manage work alongside their code repositories. It's particularly useful for software teams because it connects project planning with actual code work, making it easier to manage tasks, features, bugs, and releases in a single ecosystem.

GitHub Projects is part of the GitHub platform, which means you don't need to use a separate tool like Jira or Trello to track development work—you can manage code and projects in one place.

Key Features of GitHub Projects

1. **Kanban-style Boards**
    - Organize tasks visually using boards with columns such as **To Do**, **In Progress**, and **Done**.
    - Move tasks across columns as work progresses, giving a clear picture of the current status.
2. **Integration with Issues and Pull Requests**
    - GitHub Projects allows you to link **issues** and **pull requests (PRs)** directly to tasks on the board.
    - This ensures that project tasks are connected to actual development work, providing traceability from planning to implementation.
3. **Custom Fields & Filters**
    - Add metadata like priority, status, assignee, due date, or estimated effort to tasks.
    - Use filters to quickly view tasks assigned to specific team members, certain projects, or based on priority.
4. **Automation**
    - Automate common workflows, such as:
        - Moving a task to "In Progress" when a PR is opened.
        - Marking a task as "Done" when an issue is closed.
    - Automation reduces manual work and keeps boards updated in real time.
5. **Timeline & Roadmaps**

- Visualize project progress over time using timelines and milestones.
- Helps teams plan upcoming work, track deadlines, and monitor delivery schedules.

6. **Collaboration Tools**
   - Team members can comment on tasks, assign responsibilities, and update task status.
   - Provides a collaborative environment without leaving GitHub.
7. **Cross-repository Project Management**
   - You can manage tasks across multiple repositories in one project board.
   - This is useful for larger projects where work spans multiple repositories.

Purpose of GitHub Projects

The main purpose of GitHub Projects is to **bridge the gap between code and project management**. It allows software development teams to:

- Track work in a structured way while keeping it linked to the code.
- Plan sprints, releases, and features alongside the actual development process.
- Improve collaboration by centralizing task tracking, discussions, and updates in GitHub.
- Maintain traceability so every feature, bug fix, or improvement is linked to a specific issue or pull request.

Use Cases of GitHub Projects

1. **Feature Development Planning**
   - Track new features from ideation to deployment.
   - Link feature tasks to issues and pull requests for clear visibility.
2. **Bug Tracking**
   - Log bugs as issues and assign them to team members.
   - Monitor their progress directly on the project board.
3. **Sprint and Agile Management**
   - Organize sprints by creating columns or labels for each sprint.
   - Track progress and velocity within GitHub without switching tools.
4. **Release Management**

- Plan releases by grouping issues and tasks.
- Track completion rates and ensure all tasks for a release are completed.

5. **Cross-Team Collaboration**
   - Large teams working on multiple repositories can consolidate work in one central project board.
   - Everyone sees the overall project status in real time.

Benefits of Using GitHub Projects

- **Unified Workflow:** No need to switch between project management tools and code repositories.
- **Traceability:** Every task is linked to its code implementation, making audits and reviews easier.
- **Collaboration:** Teams can comment, assign, and update tasks in real time.
- **Efficiency:** Automation reduces manual updates and keeps boards in sync with actual work.
- **Flexibility:** Suitable for Agile, Scrum, Kanban, or hybrid workflows.

JIRA

Jira is a **project management and issue-tracking tool developed by Atlassian**. It is widely used in **software development** but also applicable for general project management. Jira is designed to help teams **plan, track, and manage work**, particularly in **Agile methodologies** like Scrum and Kanban.

Unlike simple task-tracking tools, Jira offers robust capabilities for **tracking complex workflows, managing tasks across teams, and generating detailed reports**.

Key Features of Jira

1. **Issue & Task Tracking**
   - Create and track work items such as **tasks, bugs, stories, and epics**.
   - Each item (called an "issue") can have detailed information: description, priority, assignee, due date, and attachments.
2. **Agile Boards**

- **Scrum Boards:** Organize sprints, backlog, and work items. Track progress during sprint cycles.
- **Kanban Boards:** Visualize workflow and manage continuous work in columns like “To Do,” “In Progress,” “Done.”

3. **Backlog Management**
   - Maintain a list of tasks or issues for future work.
   - Prioritize items to decide what to work on next.
4. **Custom Workflows**
   - Define workflows for different project types.
   - Specify stages like “Open → In Progress → Review → Done,” and automate transitions.
5. **Reporting & Analytics**
   - Generate reports such as burndown charts, velocity charts, cumulative flow diagrams, and sprint reports.
   - Helps teams analyze progress, predict timelines, and improve processes.
6. **Integrations**
   - Integrates with tools like Confluence, Bitbucket, GitHub, Slack, and more.
   - Helps link documentation, code, and communication with tasks.
7. **Automation**
   - Automate repetitive tasks like assigning issues, sending notifications, or transitioning issues based on triggers.
8. **Permissions & Roles**
   - Control who can view, edit, or move issues.
   - Supports different roles like Admin, Project Manager, Developer, and Viewer.

Purpose of Jira

Jira’s main purpose is to **track work efficiently, organize tasks, and provide visibility into project progress**, particularly for software development teams using Agile practices. It helps teams:

- Manage sprints and backlog items.

- Track bugs and features from start to finish.
- Visualize workflows and task progress.
- Improve productivity with reporting and automation.

Use Cases of Jira

1. **Software Development**
    - Track features, bugs, and enhancements.
    - Organize tasks in sprints for Agile teams.
2. **Agile & Scrum Management**
    - Plan and execute sprints.
    - Monitor team velocity and progress using burndown charts.
3. **Kanban Workflow Management**
    - Visualize work in progress.
    - Identify bottlenecks in the process.
4. **Project Tracking Across Teams**
    - Large organizations use Jira to coordinate multiple teams across projects.
    - Provides transparency on who is working on what.
5. **Bug & Issue Tracking**
    - Log bugs, assign developers, and track fixes.
    - Ensure quality control before product release.

Benefits of Using Jira

- **Centralized Work Tracking:** All tasks, bugs, and features in one place.
- **Supports Agile Practices:** Scrum and Kanban boards, sprint planning, and backlog management.
- **Customizable Workflows:** Adapt Jira to any project or team process.
- **Powerful Reporting:** Detailed analytics to track progress and team performance.
- **Collaboration & Transparency:** Team members can comment, share updates, and track tasks easily.

TRELLO

Trello is a **visual project management and collaboration tool** that helps individuals and teams organize tasks and projects using **boards, lists, and cards**. It is widely used for both personal productivity and professional project management because of its **simplicity and flexibility**. Trello is based on the **Kanban methodology**, where tasks move across stages (columns) to track progress.

Key Features of Trello

1. **Boards**
    - Represent a project or workspace.
    - Example: “Website Redesign Project” board.
2. **Lists**
    - Organize tasks into stages of work, such as **To Do, In Progress, Done**.
    - You can add as many lists as needed to match your workflow.
3. **Cards**
    - Each task is represented by a card.
    - Cards can contain descriptions, checklists, attachments, due dates, labels, and comments.
4. **Drag-and-Drop Workflow**
    - Move cards across lists to show progress visually.
    - Makes tracking simple and intuitive.
5. **Collaboration**
    - Assign team members to cards, add comments, and tag teammates.
    - Supports team collaboration and transparency.
6. **Checklists & Subtasks**
    - Break down tasks into smaller steps for better tracking.
7. **Labels & Prioritization**
    - Color-coded labels help categorize and prioritize tasks.
8. **Power-Ups / Integrations**
    - Add features like calendars, analytics, or integration with tools like Slack, Google Drive, and Jira.

9. **Notifications & Reminders**
    - Keeps team members updated on deadlines, changes, and comments.

Purpose of Trello

Trello is designed to **help teams and individuals visualize work, track progress, and collaborate easily**. It is especially useful for:

- Simple task tracking
- Project planning
- Agile workflows (Kanban)
- Personal productivity

Use Cases of Trello

1. **Software Development**
    - Track feature development, bugs, and releases on a board.
2. **Marketing Campaigns**
    - Manage content creation, deadlines, and approvals.
3. **Event Planning**
    - Organize tasks, vendors, and schedules for an event.
4. **Personal Productivity**
    - Use Trello for to-do lists, goals, or habit tracking.

Benefits of Using Trello

- **Easy to Use:** Drag-and-drop interface is simple for beginners.
- **Visual & Intuitive:** Boards and cards make task status immediately visible.
- **Flexible:** Can be adapted for any workflow or project type.
- **Collaborative:** Keeps the whole team on the same page.
- **Integrations:** Works with many apps for enhanced functionality.
