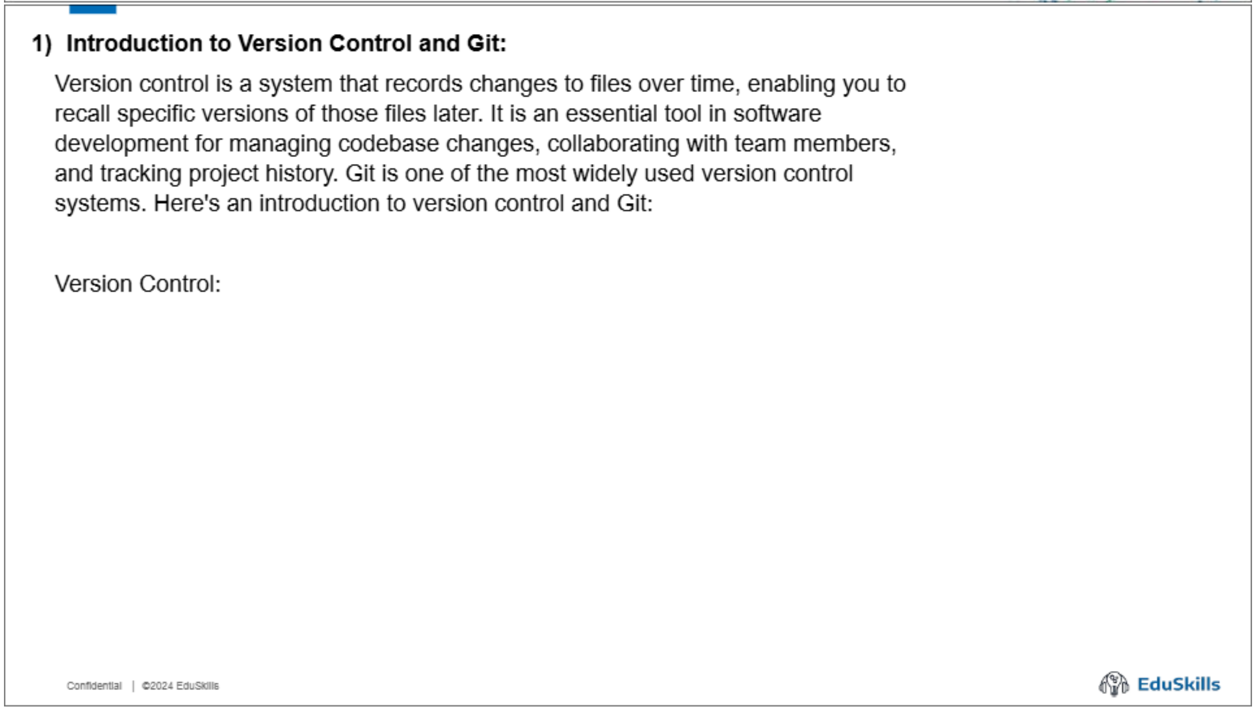
GIT & VERSION CONTROL



1) Introduction to Version Control and Git:

Version control is a system that records changes to files over time, enabling you to recall specific versions of those files later. It is an essential tool in software development for managing codebase changes, collaborating with team members, and tracking project history. Git is one of the most widely used version control systems. Here's an introduction to version control and Git:

Version Control:

EduSkills

1. Why Version Control?

- Version control allows you to track changes to files, revert to previous versions if needed, and collaborate with others efficiently.

- It helps manage complexity in software development by providing a structured way to organize code changes.

2. Key Concepts:

- **Repository:** A repository, or repo, is a storage location where version-controlled files are stored.
- **Commit:** A commit is a snapshot of changes to files at a specific point in time. Each commit has a unique identifier.
- **Branch:** A branch is a separate line of development within a repository. It allows for parallel development and isolation of features or fixes.
- **Merge:** Merging combines changes from one branch into another.
- **Conflict:** A conflict occurs when changes in one branch conflict with changes in another, and manual resolution is required.

3. Benefits:

- **Collaboration:** Multiple developers can work on the same codebase simultaneously, with changes tracked and merged seamlessly.
- **History Tracking:** Detailed history of changes, including who made each change and when, is maintained.
- **Rollback:** You can revert to previous versions of files or entire projects easily.
- **Experimentation:** Branching allows for experimentation without affecting the main codebase

Git:

1. What is Git?:

- Git is a distributed version control system designed for speed, data integrity, and support for distributed, non-linear workflows.
- It was created by Linus Torvalds in 2005 for managing the Linux kernel development.

2. Key Features:

Distributed: Each developer has a local copy of the entire repository, enabling offline work and faster operations.

Branching and Merging: Lightweight branching and efficient merging are core features of Git.

Data Integrity: Git uses cryptographic hashing to ensure data integrity.

Staging Area: Git has a staging area (index) for preparing commits, allowing for selective commit of changes.

Open Source: Git is open source and has a large community of contributors and users.

3. Basic Workflow:

- Initialize a Git repository (`git init`).
- Add files to the staging area (`git add`).
- Commit changes to the repository (`git commit`).
- Create and switch branches (`git branch`, `git checkout`).
- Merge branches (`git merge`).
- Push changes to a remote repository (`git push`).
- Pull changes from a remote repository (`git pull`).

4. Popular Git Hosting Platforms:

- GitHub: A web-based hosting service for Git repositories.
- GitLab: Another web-based Git repository manager with additional features like CI/CD pipelines.
- Bitbucket: Offers Git repository hosting and also supports Mercurial repositories.

Version control with Git is a fundamental skill for software developers and anyone involved in collaborative projects. Learning Git enables efficient collaboration, facilitates project management, and enhances code quality.

1) GIT Basics and Workflow:

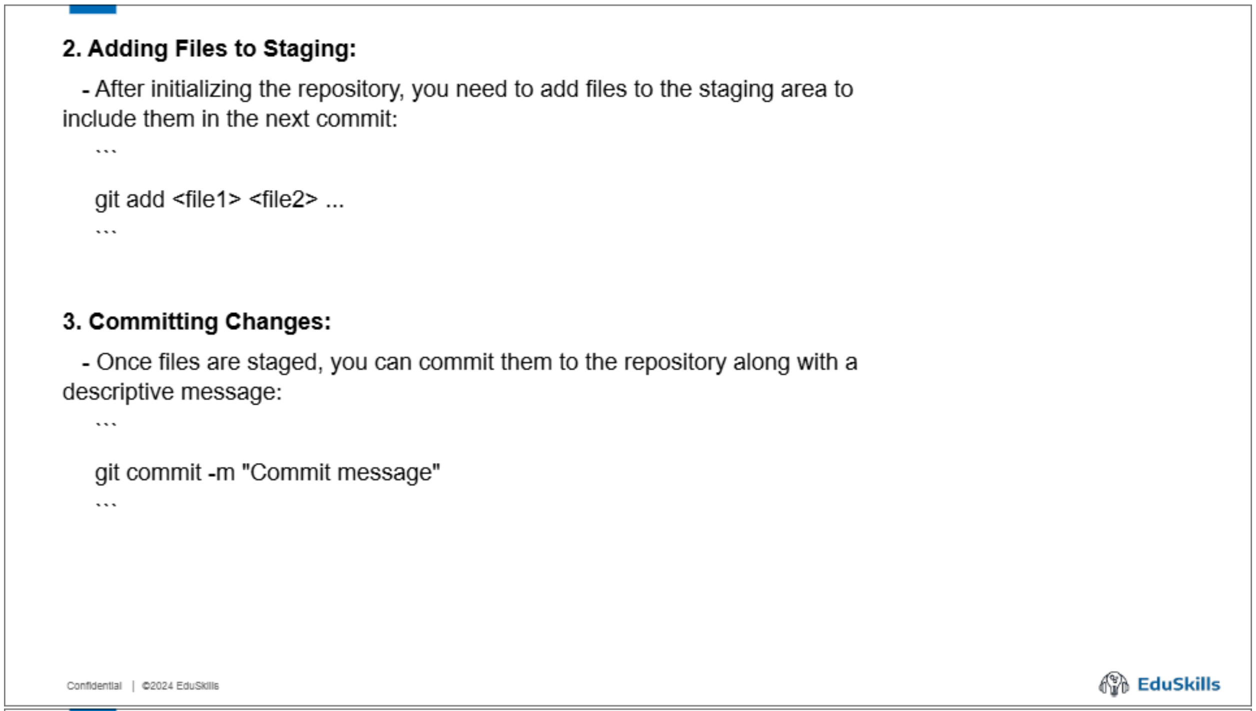
Git is a distributed version control system used by developers to manage and track changes to their codebase. It offers a powerful set of features for collaboration, tracking changes, and maintaining project history. Let's delve into Git basics and workflow:

Git Basics:

1. Initializing a Repository:

- To start using Git in your project, navigate to your project directory in the terminal and initialize a Git repository:

```
git init
```



2. Adding Files to Staging:

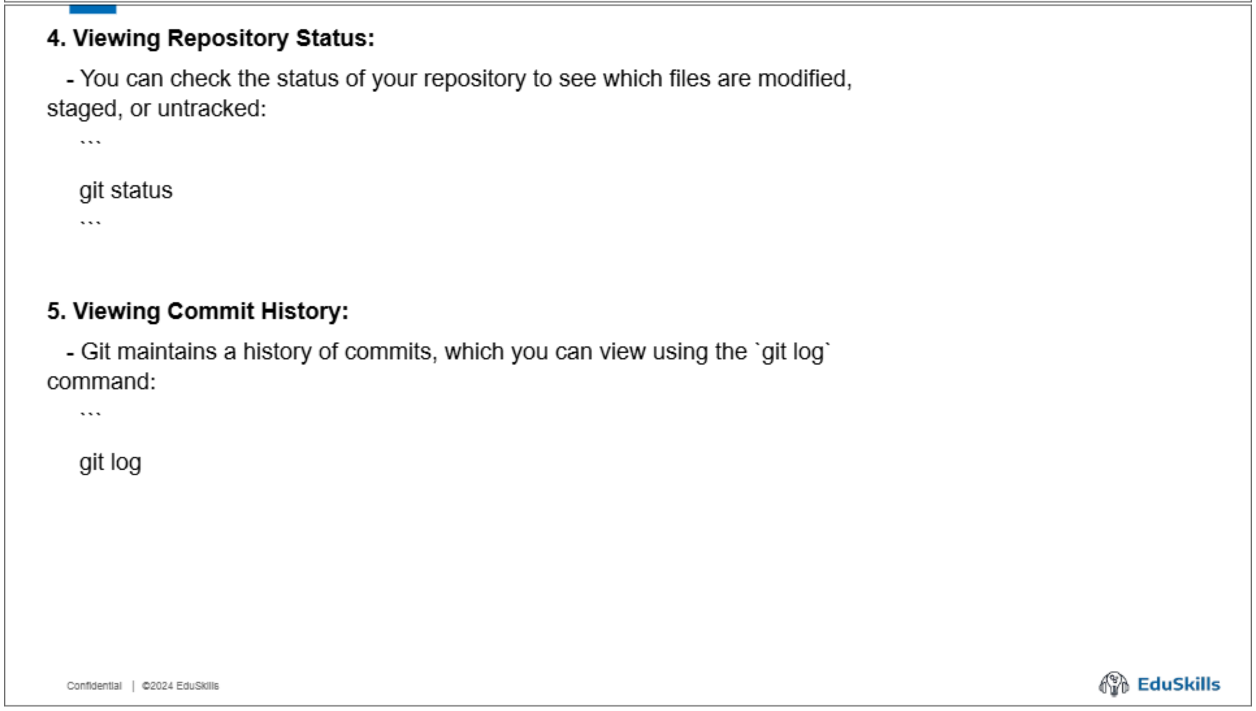
- After initializing the repository, you need to add files to the staging area to include them in the next commit:

```
git add <file1> <file2> ...
```

3. Committing Changes:

- Once files are staged, you can commit them to the repository along with a descriptive message:

```
git commit -m "Commit message"
```



4. Viewing Repository Status:

- You can check the status of your repository to see which files are modified, staged, or untracked:

```
git status
```

5. Viewing Commit History:

- Git maintains a history of commits, which you can view using the `git log` command:

```
git log
```

Git Workflow:

1. Working with Branches:

- Git allows you to work on different features or fixes simultaneously by creating branches:
  - Create a new branch:

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

  - Switch to a branch:

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

  - Create and switch to a new branch:

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

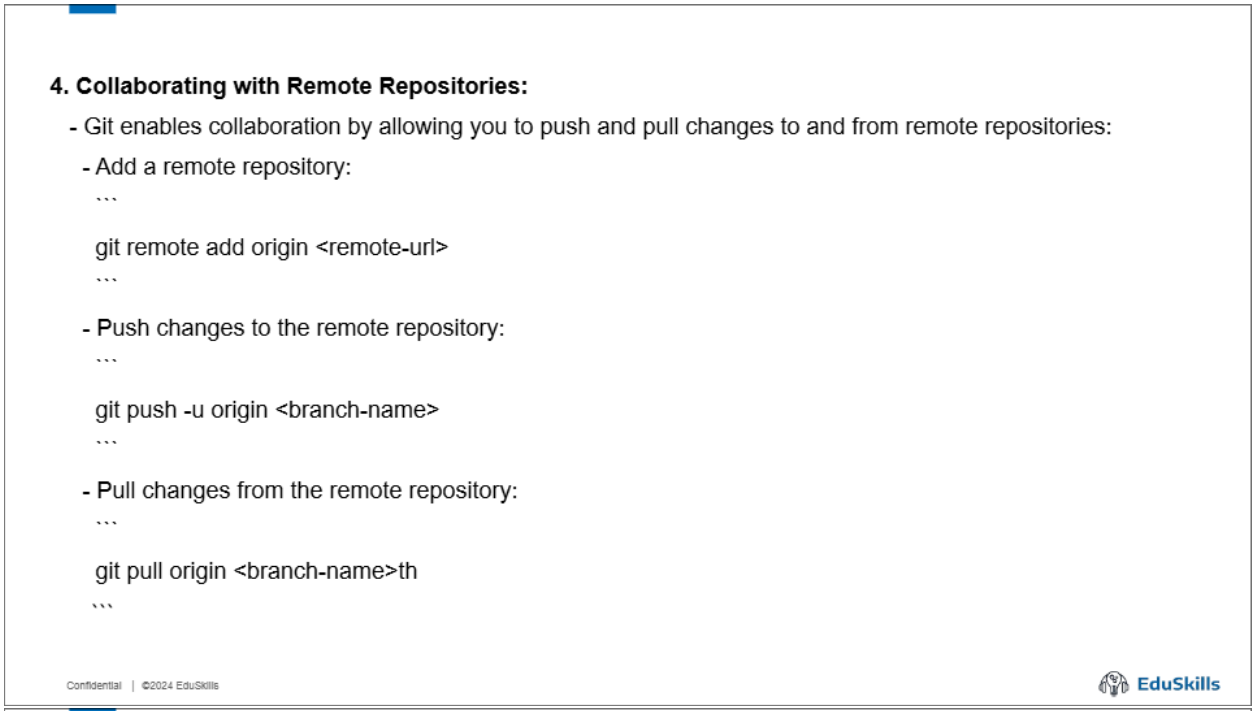
2. Merging Changes:

- After completing work on a branch, you can merge it back into the main branch:

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

3. Resolving Conflicts:

- Conflicts may occur during merging if changes conflict with each other. Git provides tools to resolve conflicts manually.



4. Collaborating with Remote Repositories:

- Git enables collaboration by allowing you to push and pull changes to and from remote repositories:
  - Add a remote repository:

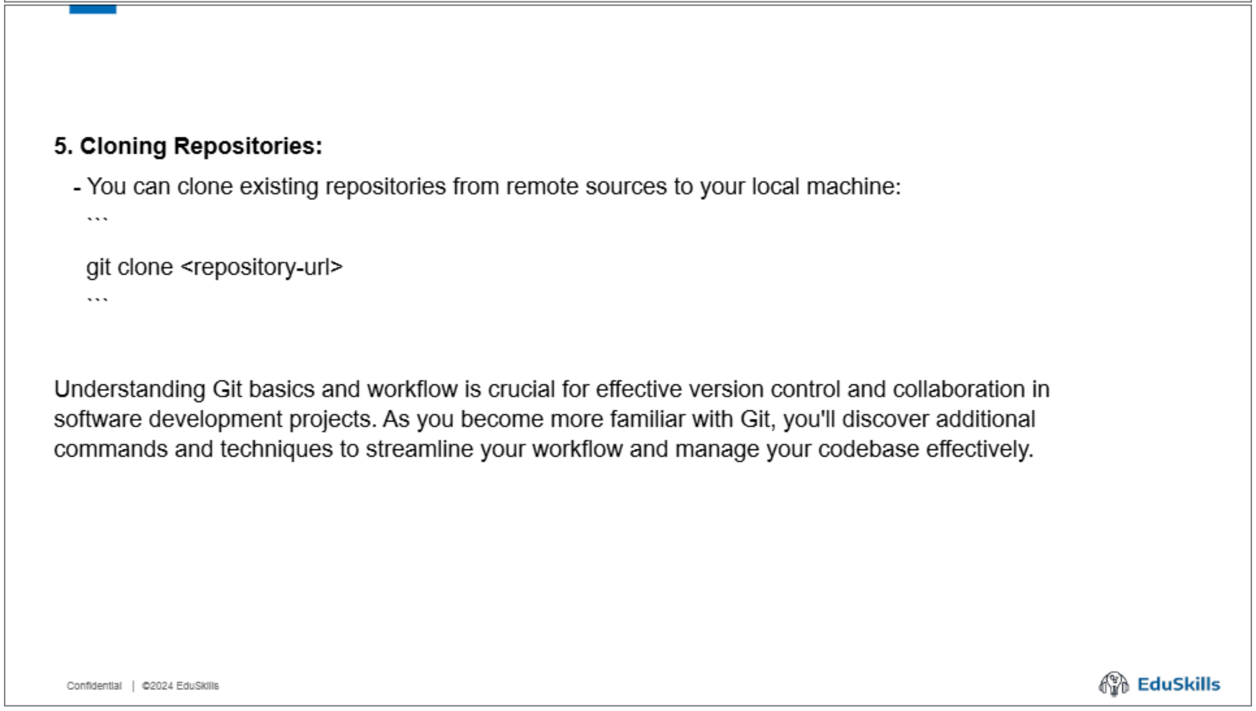
    ```
    git remote add origin <remote-url>
    ```

  - Push changes to the remote repository:

    ```
    git push -u origin <branch-name>
    ```

  - Pull changes from the remote repository:

    ```
    git pull origin <branch-name>th
    ```



5. Cloning Repositories:

- You can clone existing repositories from remote sources to your local machine:

  ```
  git clone <repository-url>
  ```

Understanding Git basics and workflow is crucial for effective version control and collaboration in software development projects. As you become more familiar with Git, you'll discover additional commands and techniques to streamline your workflow and manage your codebase effectively.

4)Branching and Pull Requests in GitHub:

Branching and pull requests are essential aspects of collaborating on GitHub, enabling developers to work on features or fixes independently and then merge their changes back into the main codebase. Here's how branching and pull requests work in GitHub:

Branching:

1. Creating a Branch:

- Before working on a new feature or fix, create a new branch in the repository. This isolates your changes from the main codebase.
- Use the "Branch" dropdown on the GitHub repository page or the `git branch <branch-name>` command locally.

2. Switching Branches:

- Use the "Branch" dropdown on the GitHub repository page or the `git checkout <branch-name>` command to switch to the branch you want to work on.

3. Making Changes:

- Make changes to the codebase in your branch locally using your preferred code editor or IDE.

4. Committing Changes:

- After making changes, stage them using `git add` and commit them using `git commit -m "Commit message"`.

5. Pushing Changes:

- Push your branch to the remote repository on GitHub using `git push origin <branch-name>`.

Pull Requests:

1. Creating a Pull Request:

- After pushing your branch to GitHub, navigate to the repository's GitHub page and click on the "Compare & pull request" button next to your branch.
- Review the changes and provide a descriptive title and comment for your pull request.

2. Requesting Reviews:

- Optionally, you can request reviews from specific individuals or teams by assigning them to your pull request.

3. Discussing Changes:

- Use the conversation tab on the pull request to discuss changes, ask questions, or provide feedback.

4. Making Changes Based on Feedback:

- If reviewers request changes, make the necessary adjustments to your branch locally, commit the changes, and push them to GitHub.

5. Reviewing Pull Requests:

- Collaborators and maintainers of the repository can review your pull request, leave comments, and approve or request changes.

6. Merging Pull Requests:

- Once the pull request is approved, a maintainer can merge it into the main branch by clicking the "Merge pull request" button.

7.Deleting Branches:

- After merging, you can delete the feature branch on GitHub to keep the repository clean.

GitHub Flow:

- The GitHub flow is a lightweight, branch-based workflow that leverages branching and pull requests for collaborative development.
- It emphasizes creating short-lived branches for each feature or fix, making changes, opening pull requests for review, and merging them into the main branch.

By following the branching and pull request workflow in GitHub, teams can collaborate effectively, review code changes thoroughly, and maintain a clean and stable codebase. It promotes transparency, code quality, and collaboration in software development projects.

5) GitHub Project Management and Automations:

GitHub offers several project management and automation features to streamline collaboration, track progress, and automate repetitive tasks. Here's an overview of GitHub's project management and automation capabilities:

Project Management:

1.GitHub Projects:

- GitHub Projects provide a flexible way to organize tasks, track progress, and manage workflows using boards.
- You can create custom project boards with columns, cards, and labels to represent different stages of your workflow (e.g., To Do, In Progress, Done).

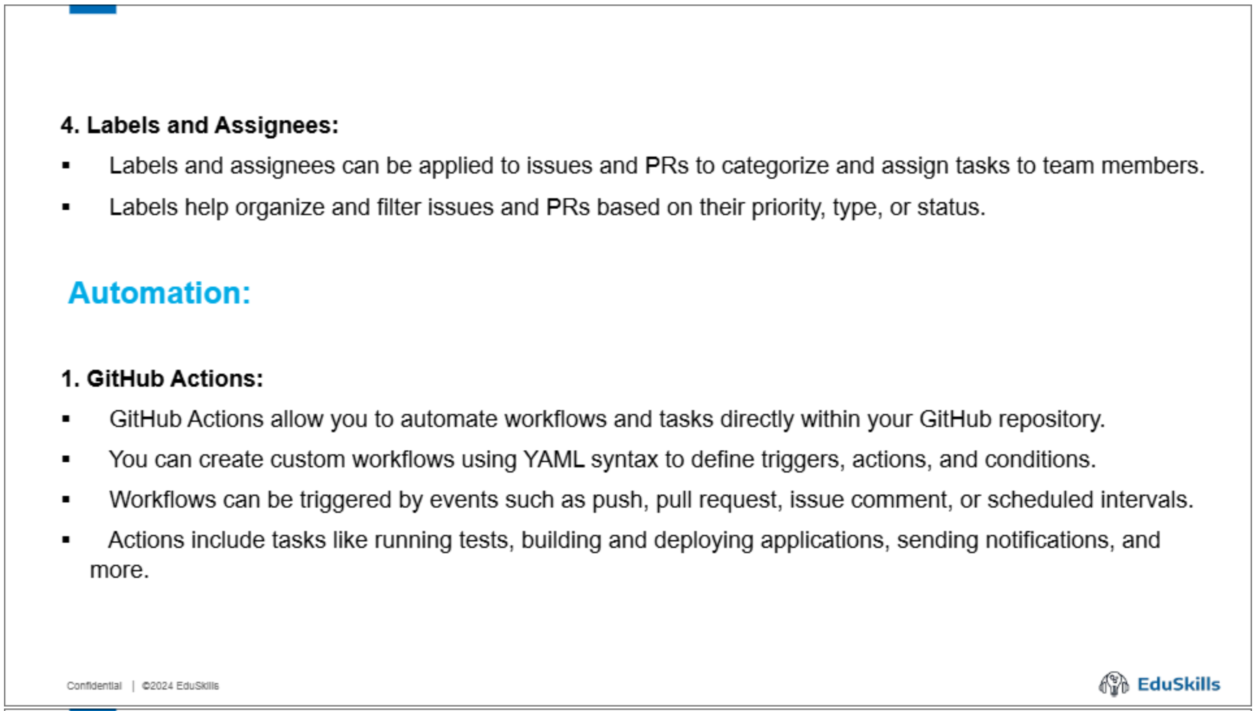
- Team members can drag and drop cards between columns to update the status of tasks.

2. Issues and Pull Requests:

- Issues are used to track bugs, feature requests, and other tasks in a project.
- Pull Requests (PRs) are used to propose changes to the codebase and facilitate code review.
- Issues and PRs can be linked to project boards, providing visibility into their status and progress.

3. Milestones:

- Milestones are used to group related issues and PRs together to represent a broader goal or feature.
- They help track progress towards project milestones and provide a high-level overview of project status.



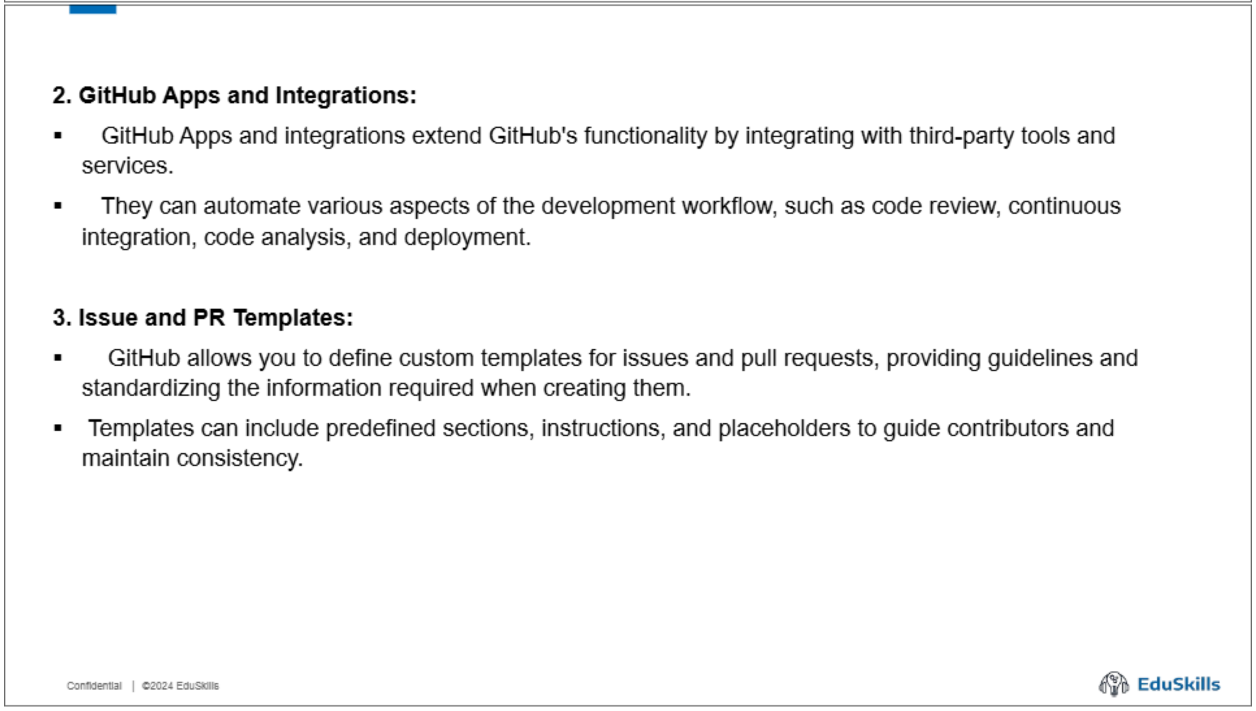
4. Labels and Assignees:

- Labels and assignees can be applied to issues and PRs to categorize and assign tasks to team members.
- Labels help organize and filter issues and PRs based on their priority, type, or status.

Automation:

1. GitHub Actions:

- GitHub Actions allow you to automate workflows and tasks directly within your GitHub repository.
- You can create custom workflows using YAML syntax to define triggers, actions, and conditions.
- Workflows can be triggered by events such as push, pull request, issue comment, or scheduled intervals.
- Actions include tasks like running tests, building and deploying applications, sending notifications, and more.



2. GitHub Apps and Integrations:

- GitHub Apps and integrations extend GitHub's functionality by integrating with third-party tools and services.
- They can automate various aspects of the development workflow, such as code review, continuous integration, code analysis, and deployment.

3. Issue and PR Templates:

- GitHub allows you to define custom templates for issues and pull requests, providing guidelines and standardizing the information required when creating them.
- Templates can include predefined sections, instructions, and placeholders to guide contributors and maintain consistency.

4. Webhooks:

- GitHub Webhooks allow you to receive real-time notifications about events that occur in your repository.
- You can configure webhooks to trigger custom actions or integrate with external services based on specific events such as pushes, pull requests, or issue updates.

Examples of Automation:

- Automatically running tests on every push or pull request using GitHub Actions.
- Automatically deploying a new version of an application to a staging environment when changes are merged into the main branch.

Automatically assigning labels or notifying team members when specific keywords are mentioned in an issue or PR comment.

GitHub's project management and automation features empower teams to collaborate more effectively, automate repetitive tasks, and streamline the development workflow. By leveraging these tools, teams can improve productivity, code quality, and project delivery.

END MODULE-13 GIT & VERSION CONTROL