Name : Ayesha Fathima M
RRN : 200171601010
Dept : B.Tech AI & DS

Study on Git

Git

Git is a powerful distributed version control system used by developers to manage source code and collaborate effectively in software development projects. It allows you to track changes to files, manage different versions of your codebase, and work seamlessly with others in a team. In this study guide, we will explore the fundamental concepts of Git and learn about various essential commands for version control.

Creating and Cloning Repositories

git init

- **Description:** This command initializes a new Git repository in the current directory. It creates a hidden .git folder that holds all the necessary files and configurations for version control.
- **Syntax:** git init

git clone [repository_url]

- **Description:** This command creates a local copy of a remote repository. It allows you to start working on an existing project and collaborate with other developers who have access to the remote repository.
- **Syntax:** git clone [repository_url]

Working with Changes

git add [file(s)]

- **Description**: This command adds file changes to the staging area, which prepares them to be included in the next commit. Staging allows you to control which changes should be included in the commit.

- **Syntax**:
    - To add a single file: git add [file_path]
    - To add multiple files: git add [file1_path] [file2_path] ...

git commit -m "[commit_message]"

- **Description**: This command records the changes made to the files in the staging area and creates a new commit in the Git history. A commit is like a snapshot of the project at a specific point in time, and it includes the changes you made along with a descriptive commit message.
- **Syntax**: git commit -m "[commit_message]"

Branching and Merging

git branch

- **Description**: This command lists all branches in the repository. Branches allow you to work on different features or fixes independently from the main codebase. The branch with an asterisk (*) indicates the currently checked-out branch.
- **Syntax:**
    - To list all branches: git branch
    - To create a new branch: git branch [new_branch_name]

git checkout [branch_name]

- **Description**: This command switches to the specified branch. When you switch branches, your working directory will be updated with the files from that branch, and you can start working on it separately from other branches.
- **Syntax**: git checkout [branch_name]

git merge [branch_name]

- **Description**: This command merges the changes from the specified branch into the currently checked-out branch. It combines the development efforts from two different branches into one.

- **Syntax**: First, ensure you are on the branch where you want to merge the changes (the destination branch), then execute: git merge [branch_name_to_merge]

Viewing History

git log

- **Description**: This command displays the commit history for the current branch. It shows a list of commits in reverse chronological order, starting with the most recent commit. The log includes information such as the commit hash, author, date, and commit message.
- **Syntax**: git log

git log [file]

- **Description**: This command shows the commit history specifically for a given file. It displays all the commits that affected the specified file, along with their details like the commit hash, author, date, and commit message.
- **Syntax**: git log [file_path]

Collaborating with Remote Repositories

git remote -v

- **Description**: This command lists the remote repositories linked to your local repository. A remote repository is a copy of the project hosted on a server (like GitHub or GitLab), where you and your team can share changes and collaborate.
- **Syntax**: git remote -v

git push [remote_name] [branch_name]

- **Description**: This command sends the committed changes from your local branch to the remote repository. It updates the remote branch with your changes.
- **Syntax**: git push [remote_name] [branch_name]
- Example: git push origin main

git pull [remote_name] [branch_name]

- **Description**: This command fetches the changes from the remote repository and automatically merges them into the current branch. It is used to synchronize your local repository with the remote repository and keep it up-to-date.
- **Syntax**: git pull [remote_name] [branch_name]
- Example: git pull origin main

Study on GitHub

GitHub

GitHub is a web-based platform and hosting service that uses Git for version control. It is one of the most popular and widely used platforms for software development and collaboration. GitHub provides developers and teams with a range of features that enhance the Git experience, making it easier to host, review, and manage code repositories. In this study guide, we will explore the key features and functionalities of GitHub and how to effectively use it for version control and collaboration.

Working with Repositories

Cloning repositories

Cloning a repository means creating a local copy of a remote repository from GitHub onto your computer. You can use the git clone command to achieve this. Cloning allows you to work on the codebase locally, make changes, and push them back to the remote repository.

Pushing changes to repositories

After making changes to your code locally, you can use the git push command to send those changes to the remote repository on GitHub. This makes your changes visible to others who have access to the repository.

Pulling changes from repositories

To keep your local repository in sync with the changes made by others, you can use the git pull command. It fetches the latest changes from the remote repository and merges them into your local branch.

Managing Issues and Pull Requests

Creating and managing issues

Issues in GitHub are used to track tasks, bugs, enhancements, or any other topic related to the development of a project. You can create issues, assign them to team members, add labels, and manage the progress of the project using issues.

Opening pull requests for code review

Pull requests (PRs) are a way to propose changes from a branch in your repository to be merged into another branch, usually the main branch. They are typically used for code review and collaboration. When you open a pull request, others can review your changes, leave comments, and suggest improvements.

Reviewing and approving pull requests

GitHub provides a built-in code review system for pull requests. Reviewers can see the changes introduced by the pull request, comment on specific lines of code, and discuss the proposed changes with the author. After thorough review, the pull request can be approved and merged into the target branch.

Merging pull requests

When a pull request is approved and all discussions are resolved, it can be merged into the target branch. Merging combines the changes from the source branch into the destination branch, incorporating the new code into the project.

Using GitHub Project Boards

Creating and managing project boards

GitHub Project Boards help you organize and prioritize work items (e.g., issues, pull requests, and notes) into customizable boards. You can create multiple

boards for different aspects of your project, such as feature development, bug tracking, or team tasks.

Organizing tasks with project boards

With project boards, you can create columns (e.g., "To Do," "In Progress," "Done") to represent the stages of your workflow. Move tasks across columns to track their progress easily. Project boards provide a visual overview of the project's status and help teams stay organized.

Collaborating with team members on projects

Project boards facilitate collaboration among team members by providing a shared view of the project's tasks and progress. Everyone can see the status of various tasks, contribute to discussions, and take ownership of specific items.

GitHub Actions

Setting up continuous integration and continuous deployment (CI/CD)

GitHub Actions allows you to automate various workflows, including continuous integration (CI) and continuous deployment (CD). CI/CD pipelines automate the process of building, testing, and deploying code, ensuring that changes are thoroughly tested before being deployed to production.

Automating workflows with GitHub Actions

GitHub Actions use YAML-based configuration files to define workflows. You can create custom workflows to suit your project's needs, combining various actions and steps to automate repetitive tasks.

Creating custom workflows for your projects

Using GitHub Actions, you can set up workflows for different scenarios, such as running tests on pull requests, deploying applications to different environments, sending notifications, and more. Custom workflows can significantly improve your development and deployment processes.

GitHub Pages

Hosting a static website with GitHub Pages

GitHub Pages allows you to host static websites directly from a GitHub repository. By pushing your HTML, CSS, and JavaScript files to a specific branch (often named gh-pages or docs), GitHub will automatically publish your website at a specific URL.

Custom domain setup for GitHub Pages

You can also use a custom domain for your GitHub Pages site, allowing you to use your domain name instead of the default GitHub Pages URL. By configuring the necessary DNS settings and adding a CNAME file to your repository, you can associate your custom domain with your GitHub Pages site.

GitHub APIs and Integrations

GitHub's API for programmatic interactions

GitHub provides a RESTful API that allows developers to interact with GitHub programmatically. You can use this API to perform various tasks, such as retrieving repository information, creating issues, managing pull requests, and more.

Integrating GitHub with third-party tools and services

GitHub integrates seamlessly with a wide range of third-party tools and services. These integrations can enhance your development workflow by connecting GitHub with tools for project management, continuous integration, code analysis, deployment, and more.