Maintaining GitHub Logs & Sharing the Project with the Team

Introduction

GitHub is an essential tool for collaborative software development, enabling teams to track changes, share code, and maintain an organized workflow. This guide outlines best practices for using GitHub in a **group web development project**, ensuring each team member can efficiently contribute, maintain logs, and resolve conflicts.

Setting Up a Shared GitHub Repository

Steps to Create a Repository

1. **One team member creates the repository**:
   - Log in to **GitHub** and click on **New Repository**.
   - Name the repository (e.g., `web-dev-project-group1`).
   - Set it as **Private** (to prevent unauthorized access).
   - Click **Create Repository**.
2. **Adding team members as collaborators**:
   - Navigate to **Settings → Manage Access**.
   - Click **Invite Collaborators** and add teammates using their GitHub usernames.

Setting Up Git Locally

Each team member must **clone the repository** to their local machine for development.

Cloning the Repository

- `git clone https://github.com/your-repo-name.git`
- `cd web-dev-project-group1`

Setting Up the Remote Repository

- `git remote add origin https://github.com/your-repo-name.git`
- `git branch -M main`
- `git pull origin main`

Branching Strategy for Collaboration

To prevent conflicts, each team member should work on a **separate branch** assigned to their tasks.

Team Member	Branch Name	Responsibilities
Student 1	`feature-homepage`	Homepage layout, navigation, styling
Student 2	`feature-services`	Services page, animations, JavaScript functionalities
Student 3	`feature-contact`	Contact page, form validation, LocalStorage integration

Creating a New Branch

- `git checkout -b feature-homepage`

Switching Between Branches

- `git checkout main`
- `git checkout feature-services`

Committing & Pushing Changes

To ensure proper tracking, team members must commit and push changes frequently.

Committing Changes

- `git add .`
- `git commit -m "Added responsive navigation bar"`

Pushing to GitHub

- `git push origin feature-homepage`

Merging a Branch into Main

1. Navigate to GitHub → **Open a Pull Request (PR)**.
2. Have another team member review the changes.
3. Merge the branch into `main` once approved.

Keeping Logs & Tracking Changes

GitHub automatically logs every change made by team members.

Viewing the Commit History

- `git log --oneline`

Checking Code Differences Between Branches

- `git diff feature-homepage feature-services`

Identifying Who Made Changes to a File

- `git blame filename.css`

Resolving Merge Conflicts

Merge conflicts occur when multiple people edit the same file. They must be manually resolved.

Steps to Resolve a Merge Conflict

1. Open the conflicting file (e.g., `index.html`).
2. Locate conflict markers (`<<<<<< HEAD`).
3. Edit and keep the correct version of the code.
4. Add and commit the resolved file:

- `git add .`
- `git commit -m "Resolved merge conflict in index.html"`
- `git push origin main`

Sharing the Repository for Submission

Once the project is complete, finalize the repository for submission.

Tagging the Final Version

- `git tag -a v1.0 -m "Final submission"`
- `git push origin v1.0`

Sharing the Repository Link

- Copy the GitHub repository link and submit it as per project guidelines.

Best Practices for GitHub Collaboration

Commit regularly – After completing each feature.

Write meaningful commit messages – E.g., "Fixed form validation issue".

Pull latest changes before starting work:

- `git pull origin main`

Use branches instead of working on `main` directly.

Track all issues using GitHub Issues – Assign tasks clearly.