**UWM - GIT**

**Create a repository in Git for your application and provide access to frontend and backend development teams, follow these steps:**

1. Create a new repository: Sign in to your chosen Git hosting platform and create a new repository. Give it an appropriate name and description that reflects your application.
2. Initialize the repository: Once the repository is created, you have two options:
   * Initialize an empty repository: If you want to start with an empty repository, you can skip this step and proceed to the next.
   * Initialize with starter code: If you already have existing code for your application, follow the instructions provided by your Git hosting platform to initialize the repository with the starter code. This will involve cloning the repository locally and pushing the code to the remote repository.
3. Set up access and permissions:
   * Invite team members: Identify the frontend and backend development team members who require access to the repository. On your Git hosting platform, navigate to the repository settings or access management section and invite the team members by their usernames or email addresses.
   * Assign appropriate access levels: Assign the appropriate access levels (e.g., read, write, admin) to the frontend and backend development teams based on their roles and responsibilities. This will control their ability to push changes, create branches, and perform other actions.
4. Communicate repository access: Inform the frontend and backend development teams about the repository's URL and the access privileges granted to them. Ensure they have the necessary credentials (e.g., username and password, SSH keys) to authenticate and access the repository.

Now, let's discuss the steps involved in common Git operations:

* Push: Pushing involves sending your local commits to the remote repository. To push changes, use the command **git push origin <branch-name>**, where **<branch-name>** represents the branch you want to push.
* Pull: Pulling retrieves the latest changes from the remote repository and merges them into your local branch. Use the command **git pull origin <branch-name>** to pull changes from the remote repository into your current branch.
* Merge: Merging combines changes from different branches into a single branch. To merge branches, use the command **git merge <source-branch>** while being on the target branch.
* Fork: Forking creates a copy of a repository under your GitHub account. This allows you to make changes to the code without directly affecting the original repository. You can fork a repository from the web interface of your Git hosting platform.
* Maintain the code: To maintain the code effectively, consider the following practices:
  + Use branches: Encourage developers to work on separate branches for new features or bug fixes. This allows for isolation and better code review.
  + Regularly commit changes: Encourage developers to commit their changes frequently with meaningful commit messages.
  + Code review: Establish a code review process where team members review each other's code before merging it into the main branch.
  + Use pull requests: Require developers to create pull requests for merging their changes into the main branch. This allows for a review process and ensures quality control.
  + Continuous Integration (CI): Set up a CI system that automatically builds and tests your code upon every push or pull request.

By following these steps and best practices, you can effectively collaborate with your frontend and backend development teams, maintain code quality, and ensure a smooth development process.

Maintaining code in Git during development and coordinating deployments involves several steps. Here is an end-to-end process that you can follow:

1. Setting up the Repository:
   * Create a new repository: Choose a Git hosting platform (e.g., GitHub, GitLab) and create a new repository for your project.
   * Clone the repository: Clone the remote repository to your local machine using the command **git clone <repository-url>**.
   * Set up the remote repository: Configure the remote repository as the upstream repository for your local repository using the command **git remote add upstream <repository-url>**.
2. Branching Strategy:
   * Define a branching strategy: Decide on a branching strategy that suits your project's needs (e.g., Gitflow, Trunk-Based Development).
   * Create branches: Create feature branches for new features, bug fixes, or other tasks using the command **git checkout -b <branch-name>**.
   * Switch branches: Switch between branches using the command **git checkout <branch-name>**.
3. Development Workflow:
   * Make changes: Write code, add new features, or fix bugs on the appropriate branch.
   * Stage changes: Stage your changes for commit using the command **git add <file-name>** or **git add .** to add all changes.
   * Commit changes: Commit your changes with a descriptive commit message using the command **git commit -m "Commit message"**.
   * Push changes: Push your committed changes to the remote repository using the command **git push origin <branch-name>**.
4. Pull Requests and Code Review:
   * Create a pull request: Create a pull request (PR) to merge your changes from the feature branch into the main branch or target branch.
   * Review code: Request team members to review your code and provide feedback.
   * Address feedback: Make necessary changes based on the code review feedback.
   * Merge the PR: Once the PR is approved, merge it into the main branch or target branch.
5. Continuous Integration and Deployment:
   * Set up a CI/CD pipeline: Configure a CI/CD pipeline that triggers automated builds, tests, and deployments for your application.
   * Configure deployment environments: Define different deployment environments (e.g., development, staging, production).
   * Define deployment steps: Specify the deployment steps, such as building the application, running tests, and deploying to the target environment.
   * Trigger deployments: Deploy the application to the respective environment either automatically after a successful build or manually through a deployment trigger.
6. Handling Conflicts:
   * Resolve conflicts: If conflicts occur during code merging or rebasing, manually resolve the conflicts by editing the affected files.
   * Commit resolved changes: After resolving conflicts, stage and commit the changes using **git add <file-name>** and **git commit -m "Merge conflict resolution"**.
7. Release Management:
   * Tag releases: Tag significant releases with version numbers or tags using the command **git tag <tag-name>**.
   * Publish releases: Publish tagged releases in the repository to indicate stable versions of your application.
8. Regular Maintenance:
   * Keep your local repository up to date: Regularly pull changes from the remote repository into your local repository using **git pull upstream <branch-name>**.
   * Periodic cleanup: Clean up your local repository by deleting merged branches using **git branch -d <branch-name>**.