Group Exercise: Git Flow Simulation

Objective:

The objective of this exercise is to practice working in a team using **Git Flow**. Each team member will play a specific role in a simulated software development project, following the Git Flow branching model. The team will collaborate on developing a feature, fixing a bug, and releasing the project.

Git Flow Overview:

Git Flow uses a branching model where the repository has the following main branches:

- **`main`**: The stable production-ready branch.
- **`develop`**: The branch where the latest development work is done.

In addition, there are supporting branches:

- **`feature/*`**: Used to develop new features.
- **`release/*`**: Used to prepare releases.
- **`hotfix/*`**: Used to fix urgent bugs on production.

Team Roles:

1. **Feature Developer**: Responsible for developing a new feature.
2. **Bug Fixer**: Responsible for fixing a bug that is discovered during development.
3. **Release Manager**: Oversees the release process and ensures the project is ready for deployment.
4. **Reviewer**: Reviews pull requests, ensures code quality, and merges branches.

Exercise Steps:

Step 1: Setup the Project Repository

1. **Initialize the repository**:
   - One team member creates a new Git repository (e.g., `project-repo`) and pushes an initial version of the codebase to GitHub or another Git hosting service.
   - The repository should have a basic structure with the `main` and `develop` branches set up.

   Example:

   ```
   bash
   Copy code
   git init
   git checkout -b develop
   echo "# Project Title" > README.md
   ```

```
git add .
git commit -m "Initial commit on develop"
git push origin develop
```

2. **Cloning the repository**:
   - Each team member clones the repository to their local environment.

```
bash
Copy code
git clone <repository-url>
cd project-repo
```

Step 2: Feature Development

1. **Feature Developer** creates a new feature branch from `develop`:

```
bash
Copy code
git checkout -b feature/new-feature
```

2. **Feature Development**:
   - Implement a simple feature (e.g., add a new function in the project). The feature should be significant enough to involve meaningful changes (e.g., a new module or functionality).
3. **Commit Changes**:
   - The Feature Developer commits the changes to the `feature/new-feature` branch.

```
bash
Copy code
git add .
git commit -m "Add new feature: XYZ"
```

4. **Push the feature branch**:
   - Push the feature branch to the remote repository.

```
bash
Copy code
git push origin feature/new-feature
```

Step 3: Code Review and Merge

1. **Reviewer** reviews the feature branch:
   - The Reviewer checks out the feature branch and reviews the code.

```
bash
Copy code
git checkout feature/new-feature
```

2. **Pull Request**:

    - The Feature Developer creates a pull request (PR) from `feature/new-feature` to `develop`.
3. **Reviewer** leaves comments, and once satisfied, approves the PR and merges the branch into `develop`.
4. **Merge the PR**:
    - After approval, the Reviewer merges the branch into `develop` and deletes the feature branch if desired.

```
bash
Copy code
git checkout develop
git merge feature/new-feature
git push origin develop
```

Step 4: Bug Fix (Hotfix)

1. **Bug Fixer** identifies a bug on the `main` branch and creates a hotfix branch:

```
bash
Copy code
git checkout -b hotfix/fix-critical-bug main
```

2. **Bug Fix**:
    - The Bug Fixer implements the necessary bug fixes and commits the changes.

```
bash
Copy code
git add .
git commit -m "Fix critical bug"
```

3. **Push the hotfix branch**:
    - Push the hotfix branch to the remote repository.

```
bash
Copy code
git push origin hotfix/fix-critical-bug
```

4. **Merge the Hotfix**:
    - Once the Reviewer approves the changes, the Bug Fixer merges the hotfix into both `main` and `develop`.

```
bash
Copy code
git checkout main
git merge hotfix/fix-critical-bug
git push origin main

git checkout develop
git merge hotfix/fix-critical-bug
git push origin develop
```

Step 5: Release

1. **Release Manager** creates a new release branch from `develop`:

   ```
   bash
   Copy code
   git checkout -b release/v1.0 develop
   ```

2. **Preparation for Release**:
   - The Release Manager prepares the project for release (e.g., updates the version number, adds release notes).
3. **Push the release branch**:
   - Push the release branch to the remote repository.

   ```
   bash
   Copy code
   git push origin release/v1.0
   ```

4. **Final Review and Merge**:
   - After the final review, the Release Manager merges the release branch into both `main` and `develop`.

   ```
   bash
   Copy code
   git checkout main
   git merge release/v1.0
   git push origin main

   git checkout develop
   git merge release/v1.0
   git push origin develop
   ```

5. **Tagging the Release**:
   - After merging, tag the release in the `main` branch.

   ```
   bash
   Copy code
   git checkout main
   git tag -a v1.0 -m "Release version 1.0"
   git push origin v1.0
   ```

Step 6: Wrap-up and Retrospective

- Once all steps are completed, the team should reflect on the process:
  - Were there any challenges in managing the branches?
  - Did all team members communicate effectively?
  - What improvements could be made for future Git Flow projects?

Deliverables:

1. A GitHub (or similar platform) repository showing the following:
   - Feature branch workflow.
   - Hotfix branch workflow.
   - Release branch workflow.
   - Tags for the release.
2. A markdown file summarizing the Git Flow process used, the roles of each team member, and any challenges encountered.

Exercise Goals:

- Practice using **Git Flow** in a team environment.
- Understand the importance of feature branching, hotfixes, and releases.
- Learn to manage pull requests and code reviews in a collaborative project.

By the end of the exercise, the team will have a good understanding of Git Flow and how to manage a project using this branching model in a real-world scenario