Git - Collaboration

Software Engineering - Tutorial

Dr. Antonio Bucchiarone - bucchiarone@fbk.eu

Academic year 2023/2024

Contents of today class

- Branching
- Collaboration
- Practical Exercises

Quiz Time :)

Exercise: Modifying a File Locally, Pushing, and Pulling

Scenario: You are a developer working on a team project hosted on GitHub. Your task is to modify an existing file, commit the changes locally, push them to the remote repository, and then pull any new changes made by your teammates.

Preparation:

- Make sure you have Git installed on your computer.
- Have access to a GitHub repository (either your own or a shared one).

Steps:

Clone the Repository:

- Open your terminal (or Git Bash on Windows).
- Navigate to the directory where you want to clone the repository.
- Clone the repository using the `git clone` command:

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

Navigate to the Repository:

- Use the `cd` command to change to the cloned repository directory:

```
cd <repository-name>
```

Modify a File:

- Open the file you want to modify in your code editor.
- Make changes to the file (e.g., update text, add new code).

Stage and Commit Changes:

- Stage your changes by running:

```
git add .
```

- Commit your changes with a meaningful commit message:

```
git commit -m "Update file: Describe the changes made"
```

Push Changes to GitHub:

- Push your local changes to the remote repository:

```
git push origin master
```

- Replace `master` with the name of the branch you're working on if it's different.

Pull Remote Changes:

- Simulate changes made by a teammate. You can ask a colleague to make a change to the same file in the remote repository, or you can do this step later after making your initial changes.

Pull Remote Changes to Your Local Repository:

- To fetch and merge the latest changes from the remote repository, run:

```
git pull origin master
```

- Replace `master` with the branch name as needed.

Resolve Conflicts (if any):

- If Git detects conflicts between your changes and your teammate's changes, you'll need to resolve them manually. Open the conflicted file in your code editor, resolve the conflicts, and then commit the resolved changes.

Push the Merged Changes:

- If you resolved conflicts, commit the changes and push them again:

```
git add .
git commit –m "Merge conflict resolution"
git push origin master
```

Verify the Synchronization:

- Ensure that your local repository is now up-to-date with the remote repository.

Create a Branch and Push a Change

Scenario: You are a developer working on a project hosted on GitHub. You want to create a new branch and push a code change to that branch.

Preparation:

1. Make sure you have a GitHub account.
2. Create a new GitHub repository or choose an existing one to work with.

Steps:

Clone the Repository:

- Open your terminal (or Git Bash on Windows).
- Navigate to the directory where you want to store your local copy of the repository.
- Clone the repository using the following command (replace `<repository-url>` with the actual repository URL):

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

Create a New Branch:

- Change to the newly cloned repository directory.

```
cd <repository-name>
```

- Create a new branch with a descriptive name for your change (e.g., `feature/add-new-feature`).

```
git checkout -b feature/add-new-feature
```

Make a Code Change:

- Open the project in your preferred code editor.
- Make a code change (e.g., update a file, add a new feature, or fix a bug).

Stage and Commit the Change:

- Stage your changes.

```
git add .
```

- Commit the changes with a meaningful commit message.

```
git commit -m "Add a new feature: <brief description>"
```

Push the Branch to GitHub:

- Push the newly created branch and your committed changes to GitHub.

```
git push origin feature/add-new-feature
```

Create a Pull Request (PR):

- Open your web browser and navigate to your GitHub repository.
- You should see a prompt to create a new pull request for the branch you just pushed. Click on it.
- Fill in the PR details, including a description of your changes.
- Review the changes and create the pull request.

Merge the Pull Request (Optional):

- If you have the necessary permissions, you can merge the pull request on GitHub once it's reviewed and approved.

Cleanup (Optional):

- You can delete the branch locally and on GitHub after it's merged or if it's no longer needed.

Locally:

```
git branch -d feature/add-new-feature
```

On GitHub (after merging):

```
git push origin --delete feature/add-new-feature
```

Exercise 1

Exercise Title: Collaborative Git Workflow

Exercise Description:

In this exercise, you'll simulate a collaborative Git workflow where two team members make separate commits to a shared Git repository. This exercise is designed to help you understand how multiple individuals can work together using Git.

Setup:

1. Create a Git repository on GitHub.
2. Clone the repository to your local machine using `git clone`.

Team Member A's Tasks:

1. Team Member A should create a new branch using `git checkout -b feature-A` and switch to it.
2. Create a new text file named `featureA.txt` in the repository directory.
3. Add some content to `featureA.txt`.
4. Commit the changes with a meaningful commit message using `git commit`.
5. Push the branch to the remote repository using `git push origin feature-A`.

Team Member B's Tasks:

1. Team Member B should create a new branch using `git checkout -b feature-B` and switch to it.
2. Create a new text file named `featureB.txt` in the repository directory.
3. Add some content to `featureB.txt`.
4. Commit the changes with a meaningful commit message using `git commit`.
5. Push the branch to the remote repository using `git push origin feature-B`.

Team Member A's Tasks (Continued):

1. After Team Member B has pushed their changes, Team Member A should switch back to the `main` branch using `git checkout main`.
2. Pull the latest changes from the remote repository using `git pull origin main`.
3. Merge the changes from the `feature-B` branch into the `main` branch using `git merge feature-B`.
4. Resolve any conflicts if they occur during the merge.
5. Commit the merge changes with a meaningful commit message.
6. Push the `main` branch to the remote repository using `git push origin main`.

Team Member B's Tasks (Continued):

1. After Team Member A has merged their changes, Team Member B should switch back to the `main` branch using `git checkout main`.
2. Pull the latest changes from the remote repository using `git pull origin main`.
3. Merge the changes from the `feature-A` branch into the `main` branch using `git merge feature-A`.
4. Resolve any conflicts if they occur during the merge.
5. Commit the merge changes with a meaningful commit message.
6. Push the `main` branch to the remote repository using `git push origin main`.

Exercise 2

In this exercise, you will practice a collaborative Git workflow involving two team members who will use Git issues, create pull requests, and perform merges. This exercise will help you understand how these Git features facilitate teamwork and code collaboration.

Setup:

1. Create a Git repository on a platform like GitHub.
2. Clone the repository to your local machines using `git clone`.

Team Member A's Tasks:

1. **Issue Creation:**
    - Create a new issue on the repository's issue tracker. Name it "Feature A Implementation" and describe it briefly.
2. **Branch Creation:**
    - Create a new branch named `feature-A` and switch to it.

Team Member A's Tasks (Continued):

3. **Work on Feature A:**
   - Create a new text file named `featureA.txt` and Add some content to `featureA.txt`.
   - Commit the changes with a meaningful commit message.
4. **Push the Branch:**
   - Push the `feature-A` branch to the remote repository using `git push origin feature-A`.

Team Member B's Tasks:

1. **Issue Creation:**
    - Team Member B should create a new issue on the repository's issue tracker. Name it "Feature B Implementation" and describe it briefly.
2. **Branch Creation:**
    - Create a new branch named `feature-B` and switch to it.

Team Member B's Tasks: (Continued):

3. **Work on Feature B:**
   - Create a new text file named `featureB.txt` in the repository directory.
   - Add some content to `featureB.txt`.
   - Commit the changes with a meaningful commit message.
4. **Push the Branch:**
   - Push the `feature-B` branch to the remote repository using `git push origin feature-B`.

Team Member A's Tasks (Continued):

1. **Create a Pull Request (PR):**
   - Team Member A should create a pull request (PR) from the `feature-A` branch to the `main` branch. In the PR description, reference the "Feature A Implementation" issue.
2. **Review and Merge:**
   - Team Member B should review the PR, add comments if necessary, and approve it if satisfied.
   - Team Member A can merge the PR into the `main` branch once it's approved.

Team Member B's Tasks (Continued):

1. **Create a Pull Request (PR):**
    - Team Member B should create a pull request (PR) from the `feature-B` branch to the `main` branch. In the PR description, reference the "Feature B Implementation" issue.
2. **Review and Merge:**
    - Team Member A should review the PR, add comments if necessary, and approve it if satisfied.
    - Team Member B can merge the PR into the `main` branch once it's approved.

Collaborative Issue Resolution

- Scenario: You and your colleague are working on a project hosted on GitHub. Your colleague has opened an issue, and you need to collaborate to resolve it.
- This exercise simulates a typical GitHub collaboration scenario, where team members work together to resolve issues, review each other's code, and maintain a clean and organized codebase. It's a great way to practice real-world collaborative development using Git and GitHub.

Steps:

Setup

- Create a GitHub repository for this exercise.
- Add your colleague as a collaborator to the repository.

Issue Creation

- Your colleague should open an issue in the repository, describing a problem or a task. This could be a bug report, a feature request, or any other project-related task.

Assigning the Issue

- The colleague who opened the issue assigns it to the other collaborator.

Branch Creation

- The collaborator who is assigned the issue creates a new branch from the main branch with a descriptive name related to the issue.

```
git checkout main
git pull origin main
git checkout -b issue-<issue-number>
```

Work on the Issue

- The collaborator works on resolving the issue in their branch. This might involve coding, documentation changes, or any other necessary tasks.

Commit and Push Change

- The collaborator commits their changes locally and pushes them to the remote branch.

```
git add .
git commit -m "Fixes #<issue-number>: Describe the changes made"
git push origin issue-<issue-number>
```

Pull Request (PR) Creation:

- The collaborator who worked on the issue opens a Pull Request from their branch to the main branch.

Review and Discussion

- Both collaborators review the changes in the Pull Request. They can leave comments, request changes, or approve the PR.

Merge the PR:

- Once the PR has been reviewed and approved, it can be merged into the main branch.

Close the Issue:

- After the PR is merged, the collaborator who opened the issue can close it, referencing the PR that resolved it.

Cleanup:

- Delete the branch associated with the resolved issue (if no longer needed).

```
git branch -d issue-<issue-number>
```

Sync Up:

- Make sure both collaborators have the latest changes by pulling from the main branch.

```
git checkout main
git pull origin main
```

Exercise: GitHub Issue Management with Tags and Priorities

Scenario: You are a project manager responsible for organizing tasks in a GitHub repository for a software development project. You need to create and manage issues with tags and priorities to ensure efficient project management.

Preparation:

1. Create a GitHub repository for this exercise.
2. Familiarize yourself with GitHub's issue and label system.

Repository Setup:

- Create a new GitHub repository or use an existing one.
- Add at least two collaborators to the repository.

Create Labels:

- Define a set of labels that represent different task categories or components of your project (e.g., "Bug," "Feature," "Documentation," "Enhancement," "Refactor," "Critical," "High Priority," "Low Priority").

Create Issues:

- As the project manager, create a set of issues in the repository, each representing a specific task or feature related to the project.
- Assign different labels to these issues based on their type (e.g., "Bug," "Feature") and their priority (e.g., "Critical," "High Priority," "Low Priority").

Tagging and Prioritizing:

- Go through the issues you've created and ensure that they are tagged with the appropriate labels based on their type and priority.
- For example, a critical bug might be tagged with "Bug" and "Critical."

Assign Issues:

- Assign some of these issues to your collaborators. Choose issues with different priorities and types to simulate real project management scenarios.

Review and Prioritize:

- As a project manager, periodically review the issues and their labels to ensure they are correctly tagged and prioritized.
- Use the "Projects" tab on GitHub to create a project board for better visualization.

Modify Labels and Priorities:

- During the exercise, simulate changes in project priorities. For example, decide that a "High Priority" feature needs to be addressed immediately.
- Update the label accordingly to reflect the change in priority.

Collaboration and Workflow:

- Collaborate with your teammates to address and close the issues. They should follow the priority and labels you've set to determine which issues to work on first.
- As collaborators complete tasks, they should mark the issues as "Closed."

Reevaluate and Reorganize:

- Periodically reevaluate the state of the project by checking the open issues and the progress made.
- Adjust labels and priorities as necessary to adapt to changing project requirements.

Reflect:

- After some time working on the issues, reflect on the effectiveness of your label and priority system. Discuss what worked well and what could be improved.

Questions?

bucchiarone@fbk.eu