Lab Manual: Git Workflow Management Simulation

Objective

To simulate a structured Git workflow for collaborative software development. Students will use task descriptions from a Trello board to create feature branches, make text-based "simulated" changes, and practice the complete branch management, code review, and release process.

Prerequisites

- Git installed on local machine
- Access to a shared Git repository (GitHub/GitLab)
- Access to the course's Trello board with predefined tasks
- Basic knowledge of Git commands

Basic Terminology & Concepts Explanation

- **Repository (Repo):** The project's database. It contains all the files, history, and commits. It exists in two places: a **remote** (hosted on a server like GitHub) and **local** (a clone on your machine).
- **Clone:** The act of creating a full copy of a remote repository on your local machine. This is the first thing you do to get started. `git clone <url>`
- **Commit:** A snapshot of your project at a point in time. It's like a save point in a game. Each commit has a unique ID, a message describing what changed, and a reference to its parent commit(s).
- **Branch:** A lightweight, movable pointer to a specific commit. Think of it as an independent line of development. The default branch is usually `main`. You create new branches to develop features without messing up the main codebase.
  - `main/master`: The production-ready, stable branch.
  - `dev`: The integration branch where completed features are merged for testing.

- o `feature/*`, `hotfix/*`: Short-lived branches for specific tasks.
- **Checkout:** The command to switch between different branches or commits in your working directory. `git checkout <branch-name>`
- **Merge:** The process of combining the changes from one branch into another. For example, merging a `feature` branch into `dev`. This often creates a "merge commit."
- **Pull Request (PR) / Merge Request (MR):** A collaboration tool on platforms like GitHub/GitLab. It's a request to merge one branch into another. It initiates discussion, code review, and automated checks before the merge happens.
- **Pull:** The act of fetching the latest changes from a remote repository and merging them into your current local branch. `git pull origin main` is equivalent to `git fetch origin` + `git merge origin/main`.
- **Push:** The act of sending your committed changes from your local repository to a remote one. `git push origin <branch-name>`
- **Workflow (Gitflow in this case):** A branching model that defines how branches interact. It provides a rigid framework for managing features, releases, and hotfixes, which is perfect for collaboration.
  - o **Feature Branches:** Isolate new work. Born from `dev`, merged back into `dev`.
  - o **Hotfix Branches:** For critical bugs in `main`. Born from `main`, merged back into `main` *and* `dev`.
  - o **Release Branches:** For preparing a new production release. Born from `dev`, merged into `main` and `dev`.
- `origin`: The default conventional name for the remote repository you cloned from.

Workflow Overview

```
feature/* → dev → stage → main (production)
```

Simulation Concept

Since no actual code will be written, each "feature" will be simulated by creating or modifying a text file. The content of the change will be the description of the task from Trello.

- **A new feature** = Adding a new section to `PROJECT_FEATURES.md`
- **A bug fix** = Correcting a typo or error in an existing file.
- **A chore** = Updating the `CHANGELOG.md` or `README.md`.

The value is not in the content but in **correctly executing the Git commands and workflow steps**.

Step-by-Step Simulation Instructions

1. Repository Setup

This foundational task will be performed by one member of your 4-person group. The rest of the team will then clone the repository that this student creates.

Objective: To create the central "source of truth" for the project, initialize its basic structure, and establish the core branching strategy.

Step-by-Step Guide for the Setup Lead:

1. Create the Repository on GitHub/GitLab:

- Log in to your GitHub/GitLab account.
- Click the **"New repository"** button.
- **Repository name:** `project-simulator-[YourTeamName]` (e.g., `project-simulator-team-alpha`).
- **Description:** `A simulation repository for practising Git workflow management.`

- Set the repository to **Public** (or Private if your organization requires it).
- **DO NOT** initialize the repository with a README, .gitignore, or license. We want to start completely empty. This is crucial.
- Click **"Create repository"**.

2. Initialize the Repository Locally and Create the Structure:

Open your terminal/command prompt and run the following commands:

```
# 1. Create a new directory for the project and navigate into it
mkdir project-simulator
cd project-simulator

# 2. Initialize a new Git repository in this directory
git init

# 3. Create the initial, basic files that every project needs
echo "# Project Simulator" > README.md
echo "# Changelog" > CHANGELOG.md
echo "# Implemented Features" > PROJECT_FEATURES.md

# 4. Stage all the newly created files to be committed
git add README.md CHANGELOG.md PROJECT_FEATURES.md

# 5. Make the first commit to the repository's default branch (often 'master'
)
git commit -m "chore: initial project setup and documentation"
```

3. Rename the Default Branch and Create the Workflow Branches:

The industry standard is to use `main` as the primary branch instead of the historical `master`.

```
# 1. Rename the current branch from 'master' to 'main'
git branch -M main

# 2. Create the 'dev' (development) branch
git branch dev

# 3. Create the 'stage' (staging/pre-production) branch
git branch stage
```

```
# 4. Verify all branches have been created. You should see: main, dev, stage
git branch
```

4. Connect the Local Repository to the Remote and Push All Branches:

Now, link your local repo to the empty one you created on GitHub/GitLab.

```
# 1. Add the remote repository URL. Copy this URL from your GitHub/GitLab page.
git remote add origin https://github.com/your-username/project-simulator-team-alpha.git

# 2. Push the main branch to the remote 'origin' and set it as the upstream
git push -u origin main

# 3. Push the dev branch to the remote
git push -u origin dev

# 4. Push the stage branch to the remote
git push -u origin stage
```

5. Verify Success:

- Refresh the page of your repository on GitHub/GitLab.
- You should now see the three files
  (`README.md`, `CHANGELOG.md`, `PROJECT_FEATURES.md`).
- Click on the branch selector dropdown. You should see three branches: `main`, `dev`, and `stage`.
- The default branch should be `main`.

6. Onboard Teammates:

- Share the repository URL with your teammates.
- They can now follow the "Student Setup - Clone and Configure" steps from the main manual.

2. Student Setup - Clone and Configure

Each student (except the first student who created step one successfully) should:

```
# 1. Clone the team's repository
git clone <your-team's-repository-url>
cd project-simulator

# 2. Configure your identity for this project
git config user.name "Your Name"
git config user.email "your@email.edu"

# 3. Verify all branches are available and up-to-date
git checkout main
git pull origin main
git checkout dev
git pull origin dev
git checkout stage
git pull origin stage

# 4. Return to the development branch to start working
git checkout dev
```

3. Feature Development Simulation Workflow

Step 3.1: Claim a Task and Create a Feature Branch

1. Go to the team's Trello board.
2. Find a task in the **"To Do"** column that is not assigned (e.g., "T-14: Implement User Login Page").
3. Assign the task to yourself and move it to the **"In Progress"** column.
4. Create a feature branch. **The branch name must match the task ID.**

```
# Always start from an updated dev branch!
git checkout dev
git pull origin dev

# Create your feature branch. Use the task ID for the name.
git checkout -b feature/T-14
```

Step 3.2: Simulate the "Development"

Simulate working on the task by adding your task's description to the `PROJECT_FEATURES.md` file.

```
# Edit the file to add your "feature"
echo "## T-14: Implement User Login Page" >> PROJECT_FEATURES.md
echo "This task involves creating a login form component with email and passw
ord fields, including validation and error handling." >> PROJECT_FEATURES.md
echo "**Status: Implemented**" >> PROJECT_FEATURES.md
echo "" >> PROJECT_FEATURES.md

# Stage and commit your changes. Follow the message convention!
git add PROJECT_FEATURES.md
git commit -m "feat: implement user login page (T-14)

- Created login form component with fields
- Added input validation
- Implemented error handling for failed login attempts"
```

Step 3.3: Push and Create a Pull Request (PR)

```
# Push your feature branch to the remote repository
git push -u origin feature/T-14
```

1. Go to your repository on GitHub/GitLab.
2. You will see a prompt to create a Pull Request for your newly pushed branch. Click it.
3. **Base:** Set to dev. **Compare:** Set to feature/T-14.
4. **Title:** feat: implement user login page (T-14)
5. **Description:** Paste the full body of your commit message here.
6. Assign a teammate as a reviewer.
7. Create the PR and move your Trello card to the **"In Review"** column.

4. Code Review and Merge to Dev (Reviewer's Task)

Another student will:

1. Review the PR online. Check for:
   - Correct branch naming.
   - Proper commit message format.
   - Meaningful PR description.
   - (Optional) The simulated change in the file is correct.

2. Approve the PR or request changes.

3. Once approved, the reviewer will merge it using the "Squash and Merge" option to keep the history clean.

4. The reviewer will then delete the `feature/T-14` branch from the remote repository.

5. The original student (and everyone else) should then update their local `dev` branch.
   ```
   git checkout dev
   git pull origin dev # This fetches the merged changes and updates your
   local dev
   git branch -d feature/T-14 # Delete your local feature branch now that
   it's merged
   ```

6. Move the Trello card to the **"Done"** column.