**Lab 1: Team Collaboration – Missing Changes**

**Scenario:**  
You and your teammate are working on the same project. Your teammate pushed an update to the README.md file directly to GitHub. You already made local changes to the same file but haven’t committed yet.

**Tasks:**

1. Clone the repository from GitHub.
2. Edit the README.md locally (don’t commit yet).
3. Try to run git pull and observe the error.
4. Use git stash to save your local changes.
5. Run git pull again to fetch the teammate’s changes.
6. Apply your stashed changes and resolve any conflicts.
7. Commit and push the final version.

**Lab 2: Hotfix on Production Branch**

**Scenario:**  
A bug is reported in production. You need to fix it urgently without disrupting ongoing feature development.

**Tasks:**

1. Clone the repo and checkout the main branch (production).
2. Create a new branch called hotfix/urgent-bug.
3. Make a fix in app.js and commit it.
4. Push the hotfix branch to GitHub.
5. Merge the hotfix branch into main locally.
6. Push the updated main to GitHub.
7. Also merge the fix into the develop branch to keep branches in sync.

**Lab 3: Feature Development with Merge Conflict**

**Scenario:**  
Two developers work on the same feature but update the same line of code in config.json. You must handle the conflict when merging.

**Tasks:**

1. Create a branch feature/payment-gateway.
2. Edit config.json to add payment settings and commit.
3. Switch to main and also edit the same line in config.json. Commit changes.
4. Try merging feature/payment-gateway into main.
5. Resolve the merge conflict manually.
6. Commit the resolution and push to GitHub.

**Lab 4: Undoing Mistakes**

**Scenario:**  
You accidentally committed a password in a file and pushed it to GitHub. Now you must undo it correctly.

**Tasks:**

1. Create a file secrets.txt with dummy password.
2. Commit and push it to GitHub.
3. Realize it’s a mistake.
4. Use git revert to undo the commit safely.
5. Push the revert commit to GitHub.
6. Add secrets.txt to .gitignore to prevent future mistakes.
7. Commit and push the .gitignore change.

**Lab 5: Working with Multiple Branches**

**Scenario:**  
Your project uses Git Flow. New features are developed on feature branches, merged into develop, and only stable code is merged into main.

**Tasks:**

1. Clone the repository.
2. Create a branch feature/user-profile.
3. Add a new file profile.html and commit.
4. Push the branch to GitHub.
5. Merge feature/user-profile into develop.
6. Push the updated develop branch.
7. After testing, merge develop into main.
8. Tag the new release as v1.1.0 and push the tag.

**Lab 6: Stash for Quick Fix**

**Scenario:**  
You’re working on a half-done feature when your manager asks for a quick fix on main. You don’t want to lose your unfinished changes.

**Tasks:**

1. On the feature/new-dashboard branch, make some unfinished edits.
2. Run git stash to save work temporarily.
3. Switch to main and apply the fix in bugfix.js. Commit and push it.
4. Switch back to feature/new-dashboard.
5. Apply your stashed changes.
6. Continue coding and commit when ready.

**Lab 7: Syncing Forked Repository**

**Scenario:**  
You forked an open-source project but your fork is outdated compared to the original repository. You need to sync it.

**Tasks:**

1. Fork a repo on GitHub.
2. Clone your fork locally.
3. Add the original repo as an upstream remote.
4. Fetch updates from upstream.
5. Merge upstream main into your fork’s main.
6. Push the updated code to your fork on GitHub.