Git Recovery and Push to Remote Repository: Step-by-Step Guide

This guide will help you recover lost commits after a `git pull` and push your changes to a remote repository, including handling errors like "non-fast-forward" and "refusing to merge unrelated histories."

Table of Contents

1. Recovering Lost Commits
2. Merging with Unrelated Histories
3. Pushing Changes to Remote Repository
4. Handling Merge Conflicts

1. Recovering Lost Commits

If you lost your local commits after pulling the latest changes from the remote repository, you can use Git's **reflog** to recover them.

Step 1: Check the Reflog for Lost Commits

The `git reflog` command shows the history of changes to references in your local Git repository, including commits that might have been lost during a pull or rebase.

```
git reflog
```

This will output a list of recent changes to your Git HEAD:

```
e.g., 1234567 HEAD@{1}: pull origin dev: Merge made by the 'recursive' strategy.
```

Step 2: Checkout Your Lost Commit

Once you find the commit hash from the reflog that represents your lost changes (before the pull), you can check out that commit.

```
git checkout <commit-hash>
```

Replace `<commit-hash>` with the actual commit hash from the reflog.

Step 3: Create a New Branch to Save the Recovered Commit

To save your recovered commit(s) safely, create a new branch:

```
git checkout -b recovered-branch
```

---

2. Merging with Unrelated Histories

If you are trying to merge branches that have no common history (for example, the branch with your lost commits and the branch you pulled from), Git will refuse the merge. You can resolve this by allowing unrelated histories.

Step 1: Merge with `--allow-unrelated-histories`

When performing the merge, you need to tell Git to allow merging unrelated histories by using the `--allow-unrelated-histories` flag:

```
git merge recovered-branch --allow-unrelated-histories
```

This forces Git to merge the two branches even though they have no common history.

---

3. Pushing Changes to Remote Repository

After merging or resolving any conflicts, you'll want to push your local changes to the remote repository.

Step 1: Push Your Changes to the Remote Repository

Once the merge is complete and conflicts (if any) are resolved, push your changes to the remote `dev` branch:

```
git push origin dev
```

---

4. Handling Merge Conflicts

If you encounter merge conflicts during the merge or rebase, follow these steps to resolve them.

Step 1: Identify the Conflicted Files

Git will list the files with conflicts. Open the files and look for conflict markers:

```
<Your changes>
```

Step 2: Resolve the Conflicts

Manually edit the file to resolve the conflicts. Delete the conflict markers and choose the changes you want to keep.

Step 3: Stage and Commit the Changes

Once all conflicts are resolved, stage the files:

```
git add <file-name>
```

Then commit the changes:

```
git commit
```

After committing, proceed to push your changes to the remote repository.

Summary of Commands

1. **Check the reflog to find lost commits:**

   ```
   git reflog
   ```

2. **Checkout the lost commit:**

   ```
   git checkout <commit-hash>
   ```

3. **Create a new branch to save the commit:**

   ```
   git checkout -b recovered-branch
   ```

4. **Merge the branches (with unrelated histories):**

   ```
   git merge recovered-branch --allow-unrelated-histories
   ```

5. **Push the changes to the remote `dev` branch:**

   ```
   git push origin dev
   ```

6. **Handle merge conflicts**:
    - Resolve conflicts manually.
    - Stage the resolved files:

      ```
      git add <file-name>
      ```

    - Commit the changes:

      ```
      git commit
      ```
