**[1]**

**Empty Commit ==========================================================**

By making an empty commit (git commit --allow-empty) without modifying any files, you can push the current state of one branch to another. This is useful when you want to copy the entire codebase from one branch to another without making actual code changes.

Git Command: ----------------------

git checkout main

git checkout -b new-feature-branch

git commit --allow-empty -m "Syncing code from main branch"

git push origin new-feature-branch

OR

git checkout main

git checkout -b new-feature-branch

git push origin new-feature-branch

**[2]**

**Detached HEAD State in Git =================================================**

A **detached HEAD** occurs when Git’s HEAD points directly to a specific commit instead of a branch. This usually happens when you checkout a specific commit, tag, or remote branch without creating a new branch.

Essentially, you can jump to a commit, make some code changes, and then save those changes by creating a new branch.  
  
Git Command: -------------------------

git checkout <commit-id>

You can also return to the head of a branch by using the following Git command: -------------

git checkout <branch-name>

**[3]**

**Deleting Latest Commits and Pushing New Ones =========================**

If you want to delete the most recent commits from a branch and push new changes instead, you can use the git reset --hard command. This is helpful when you’ve made a mistake in the latest commits and want to start fresh from an earlier commit.

Steps: ------------------------------------------------------------

1. Go to the earlier commit you want to keep:

git reset --hard <commit-id> (jump to that commit) (you can revert locally by git pull)

1. Make new changes in your code.
2. Add and commit the new changes:
   1. git add .
   2. git commit -m "New changes after reset"
3. Force push to update the branch on remote:

git push origin <branch-name> --force (force because the branch is not updated locally, it will suggest to git pull)

**[4]**

**A feature was not included when using git merge, but when I did a git rebase, the feature was correctly included.**

Let's suppose there is a branch **A**, and you create a new branch **B (Child Branch)** from **A (Parent Branch)**. Now, you make new commits on both **A** and **B**.  
If you want to copy the commits from **branch B** to **branch A**, the best practice is to use **git merge B** while on **branch A**.  
If you want to copy the commits from **branch A** to **branch B**, the best practice is to use **git rebase A** while on **branch B**.