Java Basics

Lesson 3: Merging, Rebasing, and Branching

Credits and Copyright

Copyright notices

Lesson 3: Merging, Rebasing, and Branching - OH MY!

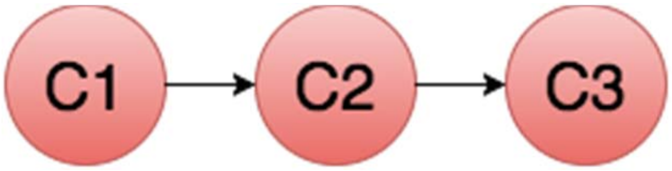
Merging

There are times when multiple developers may be working on a project together and they all create local commits without having pulled the previous guy's changes. In these cases the last one in starting with the second person will have to pull the changes from the server after committing and before pushing their changes. This process can be accomplished by taking the changes from the head of the branch being worked on and the current branch and merging them together to create a new commit, often referred to as a merge commit. Merging will keep the full history of the project and allow developers to resolve any conflicts they encounter where the same files may have been modified in both sets of changes. This conflict will appear in the file surrounded by the following setup:

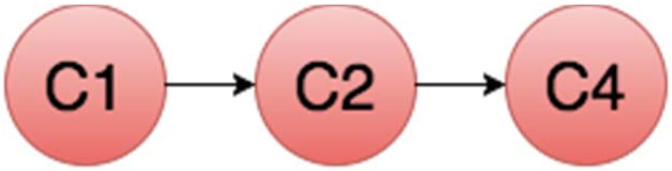
```
<<<<<<< HEAD
YOUR CODE
=======
THEIR CODE
>>>>>>> Commit ID
```

In this case you would remove the **<<<<, ===== and >>>>** lines and resolve the code to a working piece of code that incorporates the changes of one or the other or some elements of both. Once the code has been modified and compiles you can move on to the next conflict. Once all conflicts are resolved you can complete the commit and push your changes to the remote repository.

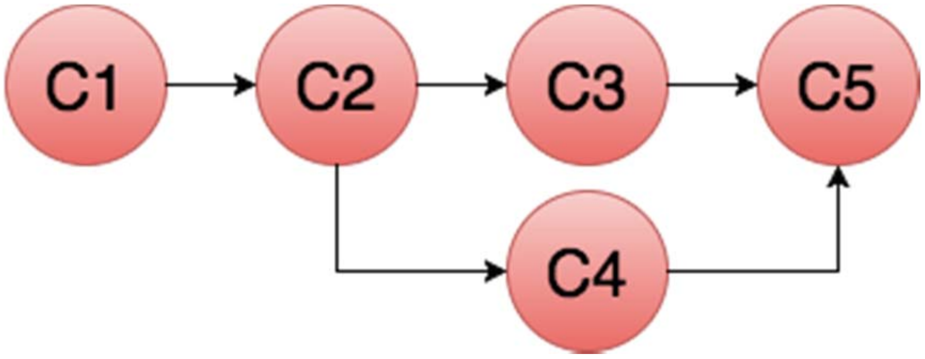
To view a merge we can start with a remote repository may look like:



Where the local repository is:



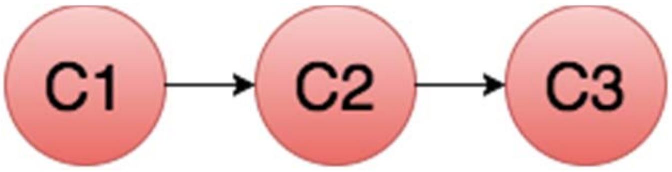
In this case we will leave both of the commits and merge them together into a new commit resulting in a history as follows:



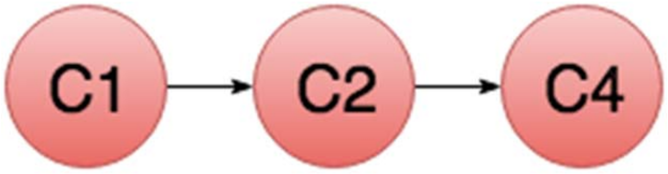
Rebasing

Another way to resolve conflicts between the remote repository and your local copy of the repository is to use rebase. While merging creates a new commit that joins the last commit of the remote repository with your current progress, rebasing will move the changes you have made to the end of the commits on the remote repository. This may sound like magic, but all it is really doing is attempting to replay the changes you made as if you made them to the remote branch's current state directly. Thus rebase will replay the changes you have made to the last commit on the history, which moves your commit(s) and maintains a linear history for the repository. In this case, conflicts can also occur and will be viewed the same as the merge. The difference in resolving rebase conflicts is that we will not—I repeat **not**—commit the changes when the conflict is resolved. Instead, the rebase is paused when conflicts occur and will be continued when they are resolved. This results in no additional commit for the rebase where merge created one. Let's take a look at this from a different perspective.

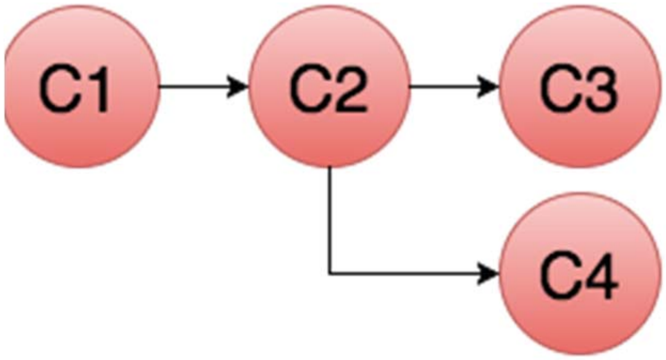
Looking at a rebase, we would see a remote repository such as this:



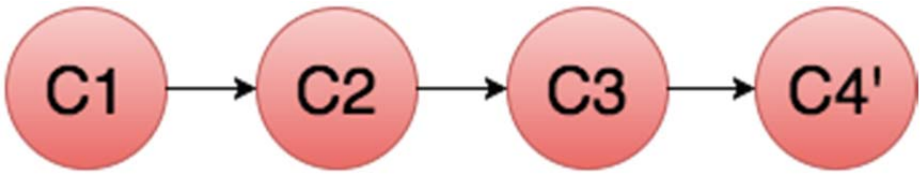
And our local repository could look like this:



We can already see that the history is different. We have a C3 present in the remote repository where our local one has C4. After pulling these changes from the remote repository, we would see something where we have two different paths in the history.:



Rather than merge the changes, we can move the changes we have made to be rebased off the latest change. In other words, move our changes to the end of the remote repository's changes that ended at C3. Thus, we would rewrite the history to move the commits as follows:



Considerations for Merge vs. Rebase

In the cohort, rebasing isn't a consideration as it is in your best interest to keep the full history of the project and use merging. In the field, consider rebasing if you require the linear history and the ability to replay the changes versus just creating an additional merge commits. In many cases, merging will be the way to go because it allows you to see the full history.

Branching

When working with larger enterprise projects or in groups of multiple developers, it may become necessary to use more than just the master branch. In the cohort, however, we can stick to the master branch.

In other cases, if we wish to create new branches, we can run the git checkout command with –b option to create a new branch, as follows:

```
git checkout –b <branch_name>
```

The `<branch_name>` is then replaced with the name of the branch we wish to create (ex. vpdev). This branch will be created locally by default and we do not need to push the branch to the remote unless we wish to have the remote repository track this new branch. In some cases, we may wish to keep the branch private and never push it to the remote repository. If we are having a hard time visualizing the branches of a solution, we can view master and origin/master as separate branches. The only difference between origin/master and a branch we create is that we would perform development directly on the created branch where origin/master is really just read-only copy of the repository and used with the remote repository.

Merging and rebasing both apply to the branches of a repository the same as previously explained because at some point, we may need to merge them with another branch of the repository, possibly the master branch. Or we can work in an isolated environment, a separate branch, and then merge back into a production branch or any other branch.