Java Basics

Exercise 4: Basic Rebase

Credits and Copyright

Copyright notices

The Learning House
427 S. 4th Street #300
Louisville KY 40202

Lesson 4: Git Quick Start

Java Cohort

Exercise 4 – Basic Rebase

Introduction

This exercise will be similar to the exercise found in the pre-work where you simulated two developers working on the same code base and performed a merge, but this time we will introduce rebase and show how it can keep the history a little cleaner. This exercise can be done solo by cloning two copies of the repository or with a partner where each of you takes on one of the roles.

Also note that the exercise serves as a guideline for investigating these concepts further, so feel free to expand on this and keep working with it. You can and should also run git status and git log frequently to see what is happening with the repository. This is the best way to get familiar with the commands, what they do, and how this all works.

Now, for the sake of clarity, Wise and Ward are at it again. I will use their names to distinguish which user should be doing what. If working on your own, feel free to name the directories as such. If working in pairs, one person can be Wise and the other can take the actions of Ward.

Steps

1. Create a repository on Bitbucket. (For clarity, the one who creates the repository will be Ward.)
2. Ward starts things off by creating the repository using the git init command. Remember that the URL can be found on Bitbucket and copied directly from there. Also make sure to specify the directory where the repository will be created.

   `git init`

   git remote add origin https://<user>@bitbucket.org/<user>/gitpractice.git
3. Ward will now add a file to the repository, commit the change, and push it up to Bitbucket.

   Example

   a. Create a text file (file1.txt) and add the text "**Ward: This is a text file.**" Save the file in the repository directory.

   b. Use the git add command to add the file: `git add --all`

   c. Use the git commit to save the change to the repository: `git commit -m "initial commit"`

   d. Then, git push to send those changes to Bitbucket: `git push -u origin master`

4 With that change pushed to Bitbucket, we can have Wise clone the repository and start where Ward left off.

```
git clone <URL>
```

Or, if you are simulating this on your own:

```
git clone <URL> <DIR>
```

In this case, <DIR> can equal Wise.

5 Wise can open the file and make a modification.

Example Text to Add: "Wise: This is definitely a text file."

Remember to run git add, git commit, and git push.

6 Ward can now add a line to the text file, but notice he does not see Wise's change... Continue anyway.

Example Text to Add: "Ward: another line of text" run git add, git commit, and wait there.

Now, Ward wants to push to Bitbucket, but Wise made a change to the file and we don't have that change. What can we do?

7 Run git fetch:

```
git fetch origin
```

This command checks the remote repository (Bitbucket) for any changes. A git status at this point will show us we have an issue:

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)
nothing to commit, working directory clean
```

We've diverged... Oh no!

No, this is okay. This just means we have to merge... Or what about rebase? In the pre-work, we used merge, which worked and can work again here. However, when we used merge, it left that fork in the history and it didn't look as clean as we hoped. Isn't there a way to clean this up? That is where rebase comes in.

8 Before Ward can push, we can proactively resolve the conflict we will have with the push and then send a clean result to Bitbucket. To do this, let's look at rebase. Now, at first glance, you may think rebase failed, but it didn't... it just found the conflict we anticipated.

```
$ git rebase origin/master
First, rewinding head to replay your work on top of it...
Applying: Added another line
Using index info to reconstruct a base tree...
M       file1.txt
Falling back to patching base and 3-way merge...
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Failed to merge in the changes.
Patch failed at 0001 Added another line
The copy of the patch that failed is found in:
   C:/_repos/ward/.git/rebase-apply/patch
```

When you have resolved this problem, run "`git rebase --continue`".

If you prefer to skip this patch, run "`git rebase --skip`" instead.

To check out the original branch and stop rebasing, run "`git rebase --abort`".

9 To resolve the conflict, open the text file, fix the file, and then save. Next, you can run git add. You only need to run git add, but no need to commit as the rebase will take care of this.

10 . With the file in good shape and staged (git add), you can go ahead and run the command to continue to rebase.

```
git rebase –continue
```

Ta-dah! You just rebased a change.

11 Go ahead and push the changes to Bitbucket.

Now you can continue by making a change on Wise. You could make several changes and then see how those rebase. The idea here is to not push anything until you check your issues ahead of time; this way you can resolve the conflicts, if any exist, before you encounter them in a push.

Many articles and books will discuss the difference between merge and rebase, and some will even engage in heated debates. My opinion is that rebase is great when you aren't dealing with anything that is public; in this context, public means any changes that have been pushed to Bitbucket. If you have not yet pushed, you can rebase and keep a nice, clean linear history. If you push something, someone pulls it down, and then you try to rebase, you can cause further issues as the person who had the original code base will not have your rebase when they try and push their changes. This can create extra work and headaches to resolve.