Java Basics

Exercise 5: Basic Branching

SOFTWARE GUILD

Credits and Copyright

Copyright notices

Lesson 4: Git Quick Start

Java Cohort

Exercise 5 – Basic Branching

Introduction

Another git topic to introduce is branching. Branching is essentially changing the pointer for commits without affecting the master branch. This means that if I want to experiment with something, or test something out, I can create my own branch. On that branch, I can do my development and see how things work while everyone else works at the same time with the master branch making updates. When I am done, I can determine if I want to save my changes and merge or rebase them back into the master branch or I can delete the branch and all the commits associated to only that branch will not be visible anymore.

To perform our git branching, we will use a new repository and perform a few modifications, concluding with a rebase back to master.

Steps

1. Create a new repository on Bitbucket.
2. Create the repository locally.
3. Add a file to the directory to initialize the repository.

   In this case, you can just add a new text file. Once you add the text file, use git add and git commit to save a snapshot.

4. Create a branch to do your work.

   ```
   git branch <branchname>
   git checkout <branchname>
   ```

   You can also do this in a single step using git checkout:

   ```
   git checkout -b <branchname>
   ```

5. Now, you can make modifications to the file on your branch without affecting the master.

   Make changes to text file. Run git add, git commit.

   Next, check the master branch out and review the file:

   ```
   git checkout master
   ```

   In reviewing the file, you will see that your changes are gone. Go back to your branch and they reappear… Magic, huh? This just demonstrates how your branch really does isolate the development. When you are done comparing the file on master and your branch, make sure you return to your branch.

   ```
   git checkout <branchname>
   ```

6. At this point, you can repeat Step 5 a few times or move on to merging your changes.

7 Because we have a commit, it’s on our dev branch we created, and no one else has worked on this repository, we can merge this into master pretty easily:

```
git checkout master
git merge <branchname>
```

That’s it! No conflicts, no problems. We don’t even have an additional merge commit.

No merge commit, but…

This is a case where rebase and merge would have the exact same result. You see, no one else is working on this project and no one modified master, so master was in the same state we left it when we created the branch. Therefore, there are no conflicts — our commit just comes over to master, master fast-forwards to the latest commit, and now both branches are at the same spot.

8 Notice what happens if we did not push the repository to Bitbucket:

```
git push -u origin master
```

Bitbucket only sees the master branch. We didn't push the development branch we created and therefore Bitbucket doesn't know about it.

9 Now, we can delete our branch if we are done developing on it.

```
git branch -d <branchname>
```

You may get into situations where you don’t delete branches. This would be where Bitbucket is also tracking the branch you are working on and it needs to be available to others. In cases where you just create the branch locally and you don’t push it up, you can clean the branch up when you are done and delete it. Just remember to merge or rebase your changes if you want to keep them!