COMP3111: Software Engineering

Advanced Git – conflict resolution

Learning Outcomes

- To resolve and prevent having conflict in Git
- Writing Javadoc

This lab is your final lab. You are required to summarize all the material you have learned in your previous lab. There will be less steps and references shown in this lab work. However, you shall be able to find the answer from your previous lab exercise if you truly understand what they were about. Please feel free to contact your TA if you have any doubt.

Preparation

Please prepare an empty remote repo, clone the remote repo in two clients (they can be on the same machine just different folders). You should be able to compile a Java program on those clients using Gradle. Make sure your command line git is properly installed

Install command line git for Windows: (https://git-scm.com/download/win)
(ps. Our lab machine has installed command line git for you already)

Task 1 – Creating Conflict

(image src: https://www.enago.com/academy/should-journal-editors-avoid-conflict-of-interest/)

Step 1. Clone the remote repo to two clients. Let’s call those client machine Trump and Kim.

Step 2.1. On Trump, create a Java Project or a Java Gradle Project with a simple executable, like Lab 1 for example.
Step 2.2 Build and run it.
Step 2.3. Stage everything including the build folder and class files. Commit and push to the remote repo.
Step 2.4 Edit the source code (e.g. write an comment), and build it again.

Step 3. On Kim, pull the code and edit one source file.
Step 3.1 On Kim, edit the same source code so that the output of the program also changes (e.g. add System.out.println();).
Step 3.2 On Kim, stage **the source code only**. Commit and push.

Step 4. On Trump, pull and look at your screen. You might see:

```
From https://github.com/khwang0/test-lab8
 * branch          master     -> FETCH_HEAD
Updating 690338c..4226aa5
error: Your local changes to the following files would be overwritten by merge:
        A.java
Please, commit your changes or stash them before you can merge.
Aborting
```

Step 5. Check the git log on both clients. Try to understand what is going on and why there is a conflict.

Task 2: Resolve the conflict with Manual Merge

Step 1.1 On Trump, commit your latest changes.
Step 1.2 Pull again and you shall see

```
From https://github.com/khwang0/test-lab8
 * branch          master     -> FETCH_HEAD
Auto-merging A.java
CONFLICT (content): Merge conflict in A.java
Automatic merge failed; fix conflicts and then commit the result.
```

Step 1.3 Open the source file (A.java in my case) and resolve the conflict manually. You might see some symbol >>>>>>>>> and <<<<<<<<< in your java file. After this steps you should not be keeping any of those.

Step 1.4 Commit and push it again.
Step 1.5 Check your git log.

Step 2. On Kim, pull it and check the git log again.

Reflection question: Can you spot the merge from the git log? Does the git log of two clients synchronize now?

Task 3: Resolve the conflict with Stash.

Step 1. Reset both clients to the conflict status (right after Step 4 of Task 1). You might want to use git reset to help you. After reset Kim to the correct commit, you need to discard your change on the source file from Task 2 Step 2.

If you can't figure out how to do it with git reset, you can just redo Task 1.

Step 2. On Kim, stage the class file as well and make another commit. **Force push it**. (git push –f origin master)

You might try to repeat Task 2 to resolve the conflict. But it would not work as the class file is binary. The >>>>>>> and <<<<< symbol is not written in the class file. After trying this, use git reset - - hard to rollback.

Step 3.1. On Trump, pull the code and you shall see this

```
From https://github.com/khwang0/test-lab8
 * branch            master     -> FETCH_HEAD
Updating 690338c..5c04f7f
error: Your local changes to the following files would be overwritten by merge:
        A.class
        A.java
Please, commit your changes or stash them before you can merge.
Aborting
```

Step 3.2 On Trump, use git stash to stash your code.
Step 3.3 Pull again and you shall be able to resolve the conflict.

Step 4. Check the git log again to check your latest structure.

Reflection question: What does git stash do? What is the disadvantage in using git stash?

Task 4: Using .gitignore

It is rather a bad idea to stage your build file (e.g. .class, .exe, etc…) We want to use a file .gitignore to stop git to stage our class file.

Reflection Question: Why it is a bad idea? Can you comment it with the experience in Task 3?

Step 0. On Kim, git rm -f all class file, e.g. git rm –f A.class.

Step 1.1. On Kim, create a file call .gitignore under your folder. The file contains only one line.

```
*.class
```

Step 1.2 On Kim, stage the file .gitignore.

Step 1.3 On Kim, commit the changes.

Step 1.4 On Kim, edit the Java source and build again. The class should have changed already. Check the git status again. You should not see the class file is tracked by the git anymore.

> Reflection Question: Look at the .gitignore file in the project Github. It is not correctly set right now. Can you fix it?

Task 5: Writing Java doc

Java has defined some syntax in writing documentation. By conforming those syntax, we can use the javadoc program to generate a nice looking HTML pages.

Step 1. Run the Gradle task Javadoc in your project.

(note: You might experience the following error. In this case, replace all <br/> with <br>)

```
E:\JavaWorkspace\WebScrapper\src\main\java\comp3111\webscraper\WebScraper.java
:16: error: self-closing element not allowed: <br/>
 * <br/>
   ^
```

Step 2. Open the \build\docs\javadoc\index.html. View your index.html file.

To understand the topic a little bit more, you can follow the exercise
https://github.com/khwang0/Line-chatbot-for-COMP3111/tree/master/lab7-JavaDoc
It was prepared for the students of the course in last year.