

Dealing with Conflicting Updates in Git

CS 5010 Program Design Paradigms



Adapted from “Bootcamp” Lesson 0.6 by:

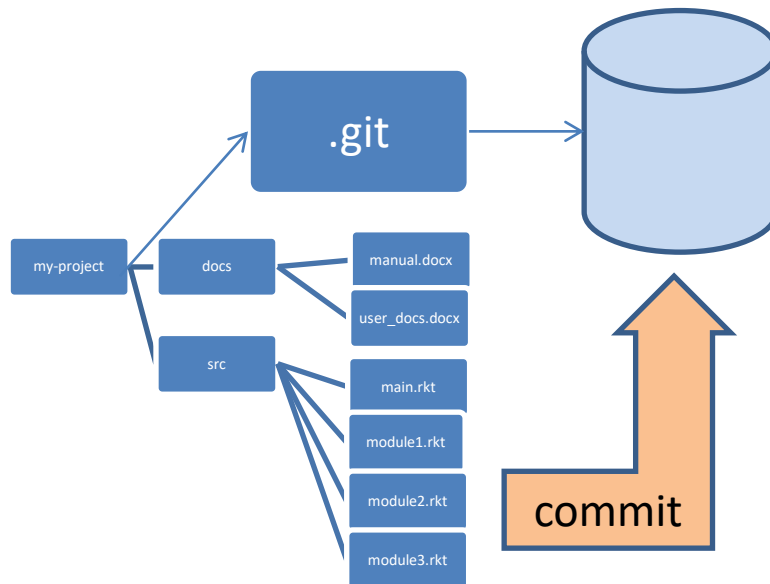
© Mitchell Wand, 2012-2014

This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).

Learning Objectives

- At the end of this lesson, you should be able to:
 - explain what happens when you pull changes from an upstream repository
 - understand what a conflict is
 - resolve a simple merge conflict in a text file (including a .rkt file)

A Commit

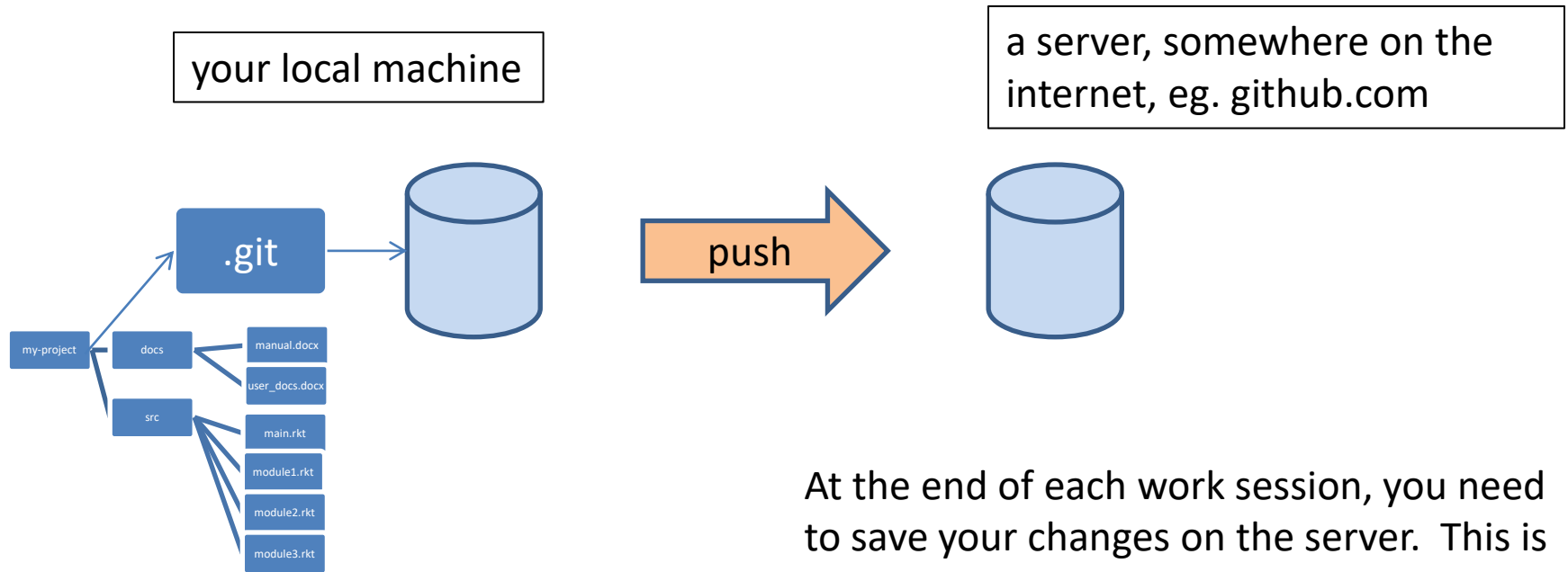


Remember the basic story from the preceding lesson

When you do a “commit”, you record all your local changes into the mini-fs.

The mini-fs is “append-only”. Nothing is ever over-written there, so everything you ever commit can be recovered.

Synchronizing with the server (1)

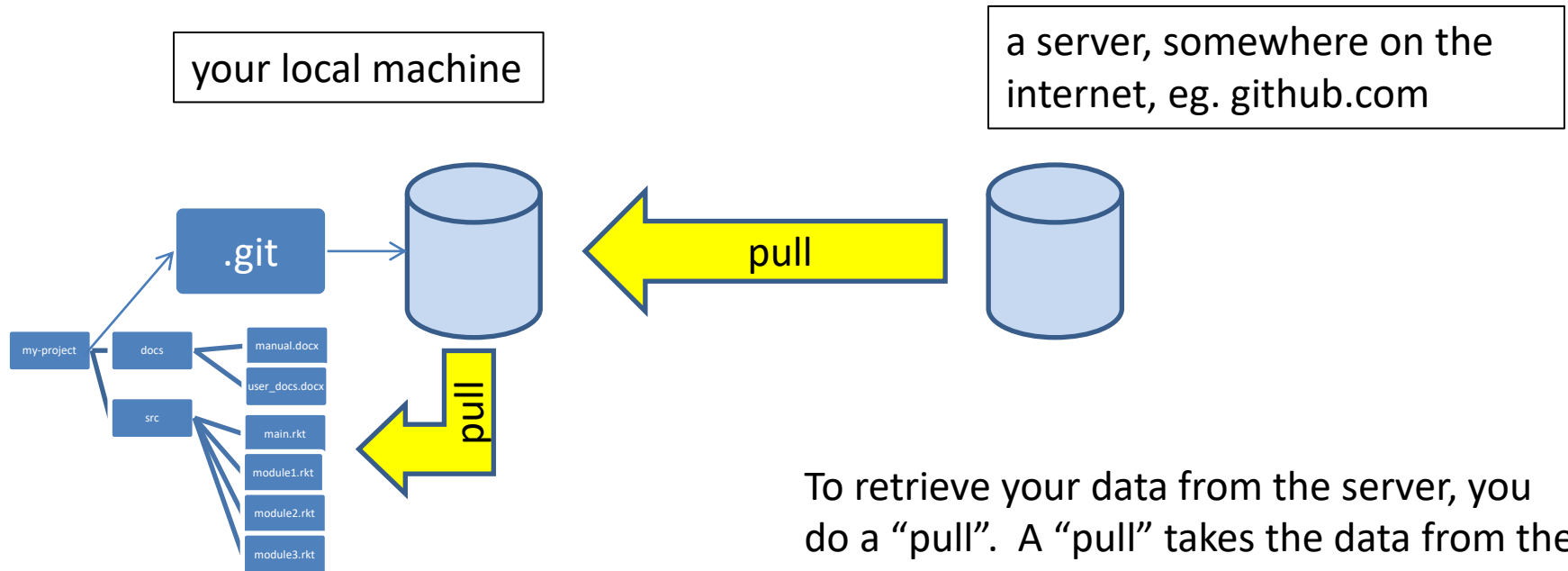


At the end of each work session, you need to save your changes on the server. This is called a “push”.

Now all your data is backed up.

- You can retrieve it, on your machine or some other machine.
- We can retrieve it (that’s how we collect homework)

Synchronizing with the server (2)

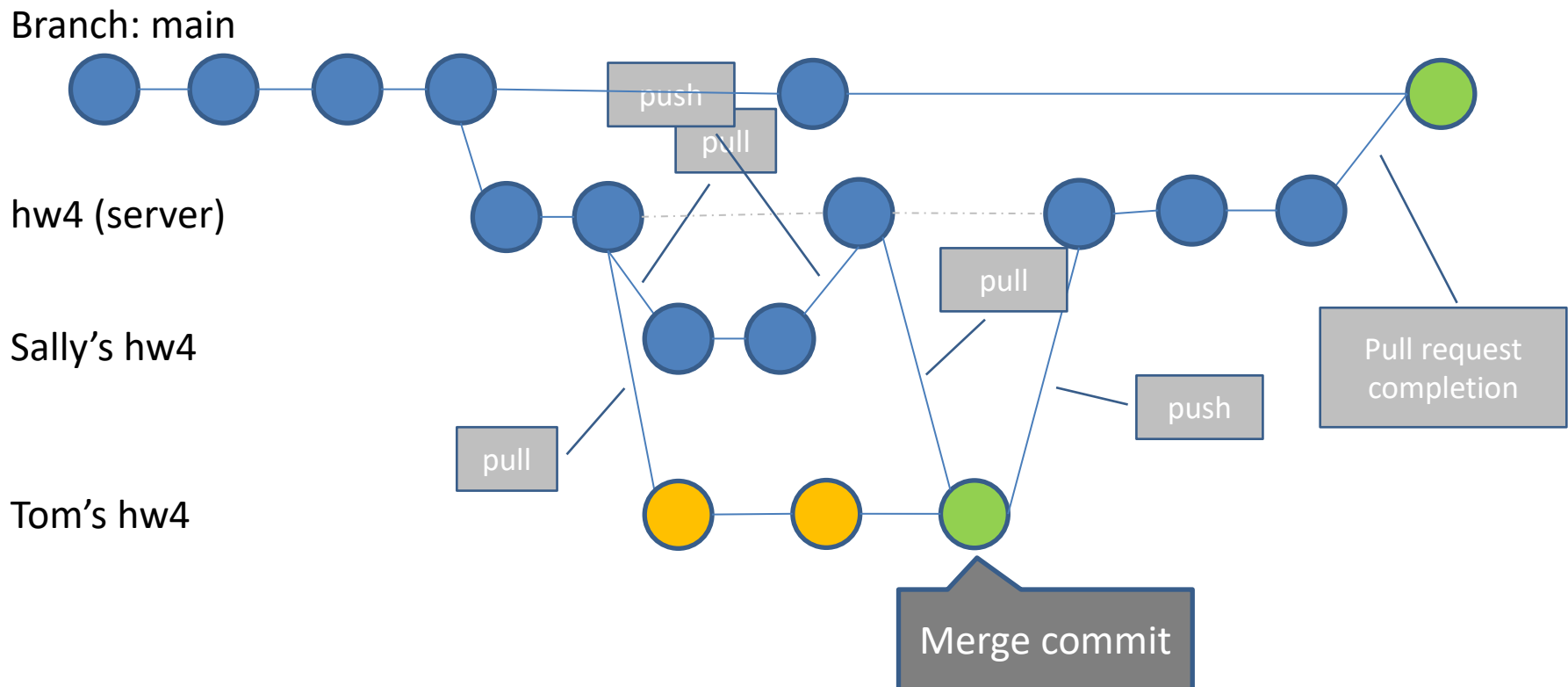


To retrieve your data from the server, you do a “pull”. A “pull” takes the data from the server and puts it both in your local mini-fs and in your ordinary files.

If your local file has changed, git will merge the changes if possible. If it can't figure out how to the merge, you will get an error message. ~~Dealing with this is beyond the scope of this tutorial~~ 😞

Q: When might you need to merge?

A: When your partner committed some changes to the server, which you don't have.

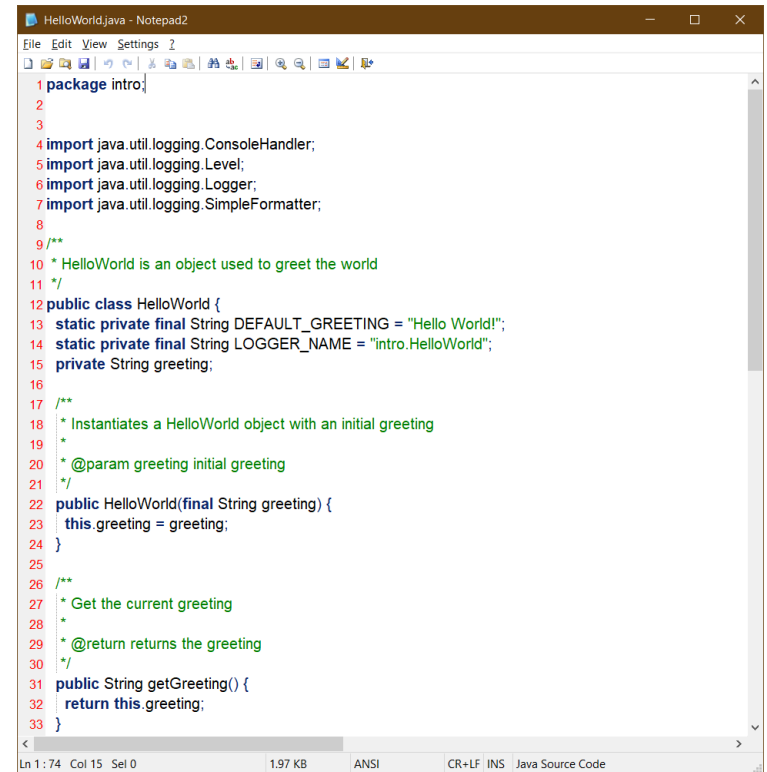


Most of the time, this works well

- So long as you and your partner are working on separate parts of the file, this works fine.
- Both sets of changes get made, and the history on the server stays linear
- But what happens if you and your partner commit incompatible changes?

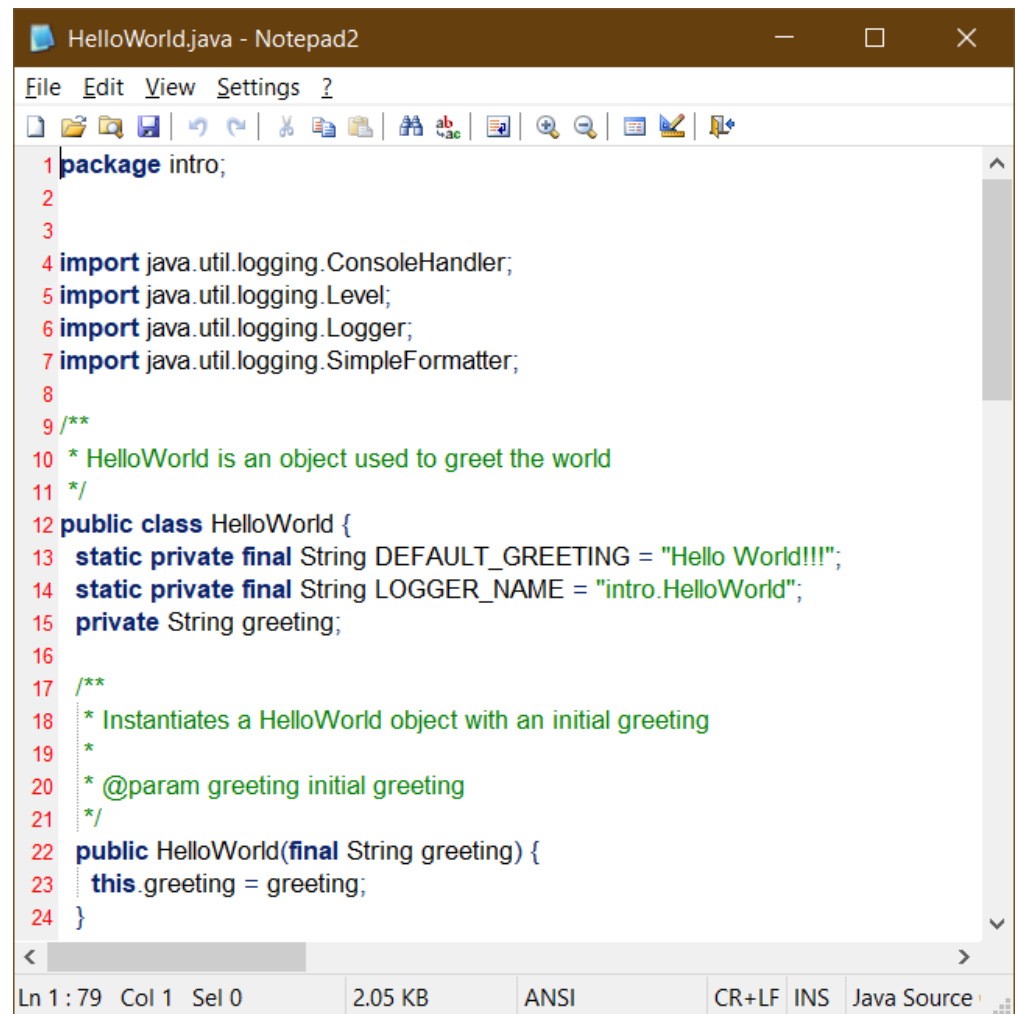
A merge conflict is born

- HelloWorld.java
- We want to change the greeting.
- Both Tom and Sally make changes.



```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13     static private final String DEFAULT_GREETING = "Hello World!";
14     static private final String LOGGER_NAME = "intro.HelloWorld";
15     private String greeting;
16
17     /**
18      * Instantiates a HelloWorld object with an initial greeting
19      *
20      * @param greeting initial greeting
21      */
22     public HelloWorld(final String greeting) {
23         this.greeting = greeting;
24     }
25
26     /**
27      * Get the current greeting
28      *
29      * @return returns the greeting
30      */
31     public String getGreeting() {
32         return this.greeting;
33     }
```

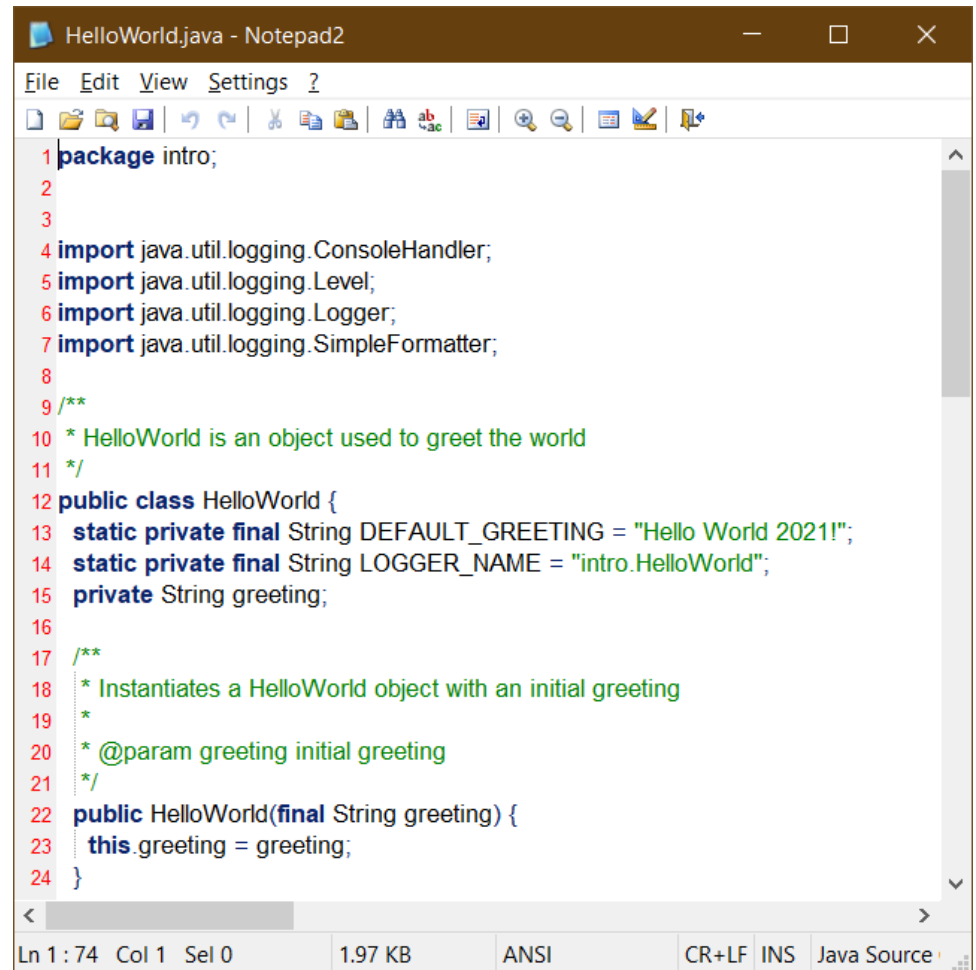

Tom adds some extra
exclamation marks to
DEFAULT_GREETING



```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13     static private final String DEFAULT_GREETING = "Hello World!!!";
14     static private final String LOGGER_NAME = "intro.HelloWorld";
15     private String greeting;
16
17     /**
18      * Instantiates a HelloWorld object with an initial greeting
19      *
20      * @param greeting initial greeting
21      */
22     public HelloWorld(final String greeting) {
23         this.greeting = greeting;
24     }
```

Ln 1 : 79 Col 1 Sel 0 2.05 KB ANSI CR+LF INS Java Source

Sally adds the year
DEFAULT_GREETING



```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13     static private final String DEFAULT_GREETING = "Hello World 2021!";
14     static private final String LOGGER_NAME = "intro.HelloWorld";
15     private String greeting;
16
17     /**
18      * Instantiates a HelloWorld object with an initial greeting
19      *
20      * @param greeting initial greeting
21      */
22     public HelloWorld(final String greeting) {
23         this.greeting = greeting;
24     }
25 }
```

Ln 1 : 74 Col 1 Sel 0 1.97 KB ANSI CR+LF INS Java Source

The Merge

Sally merges Tom's changes into her branch.

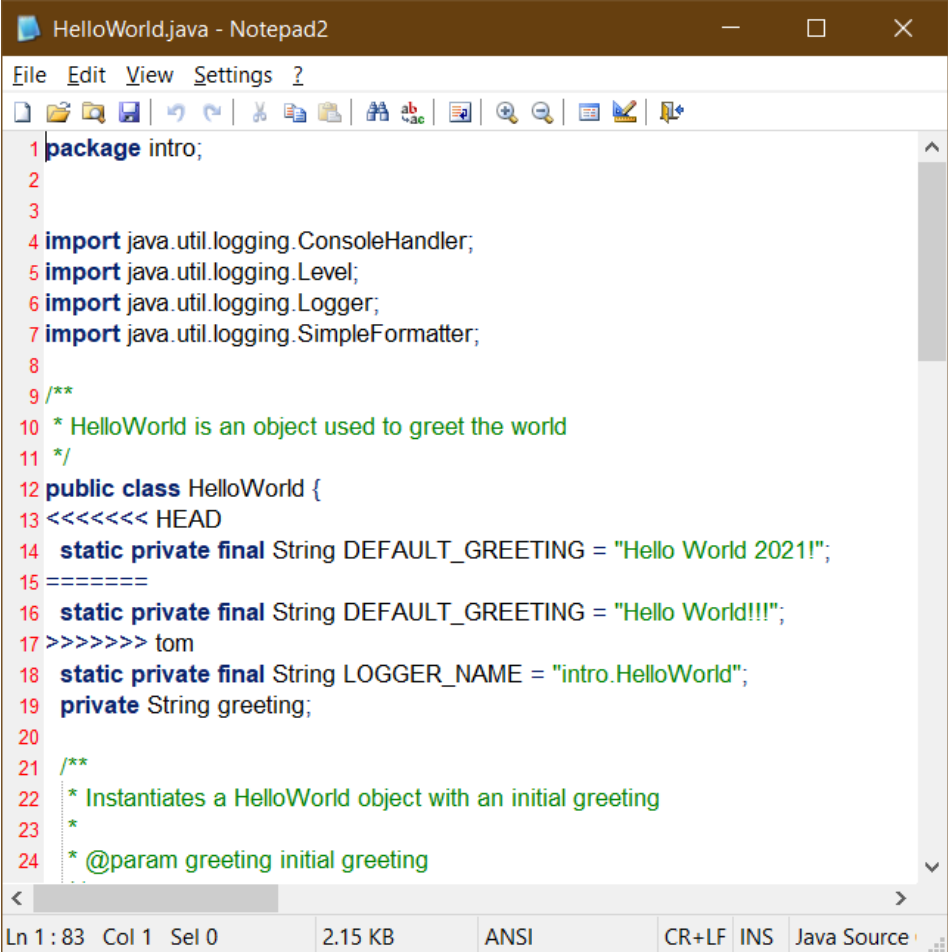
```
Auto-merging HelloJava16/src/main/java/Intro/HelloWorld.java
```

```
CONFLICT (content): Merge conflict in  
HelloJava16/src/main/java/Intro/HelloWorld.java
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

Examining the conflict

- Open conflicting files in your text editor
- Find merge markers
 - Search for
<<<<<<<
- HEAD is Sally's branch
- Tom's incoming change labelled.
- Fix the code as appropriate

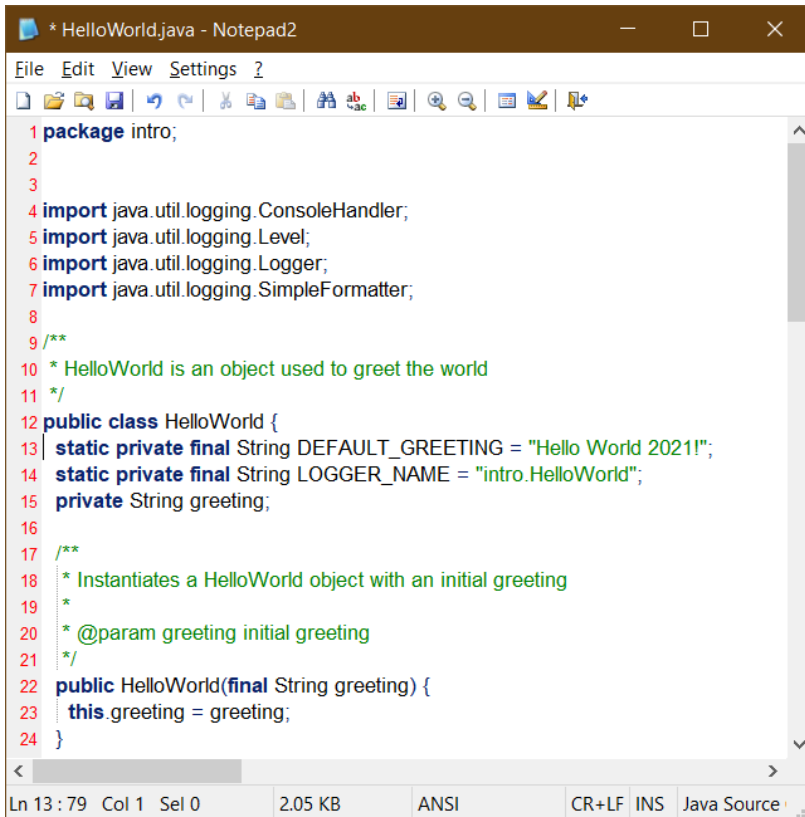


```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13 <<<<<<< HEAD
14     static private final String DEFAULT_GREETING = "Hello World 2021!";
15 =====
16     static private final String DEFAULT_GREETING = "Hello World!!!";
17 >>>>>>> tom
18     static private final String LOGGER_NAME = "intro.HelloWorld";
19     private String greeting;
20
21     /**
22     * Instantiates a HelloWorld object with an initial greeting
23     *
24     * @param greeting initial greeting
```

Ln 1 : 83 Col 1 Sel 0 2.15 KB ANSI CR+LF INS Java Source

Resolving the conflict

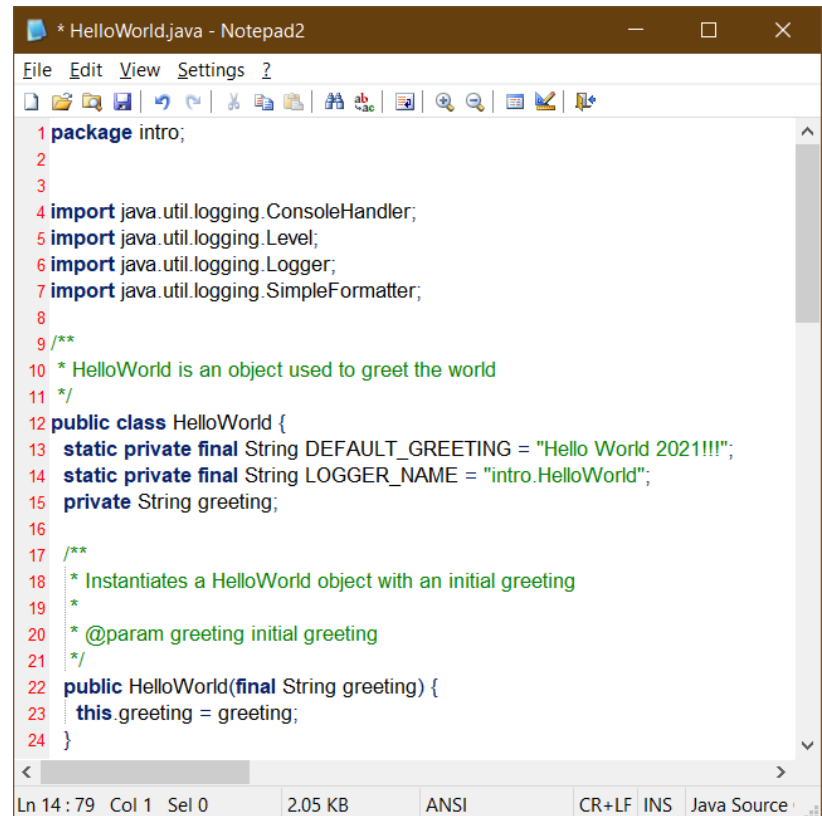
Take one or the other



```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13     static private final String DEFAULT_GREETING = "Hello World 2021!";
14     static private final String LOGGER_NAME = "intro.HelloWorld";
15     private String greeting;
16
17     /**
18      * Instantiates a HelloWorld object with an initial greeting
19      *
20      * @param greeting initial greeting
21      */
22     public HelloWorld(final String greeting) {
23         this.greeting = greeting;
24     }
25 }
```

Ln 13 : 79 Col 1 Sel 0 2.05 KB ANSI CR+LF INS Java Source

Manual merge text together



```
1 package intro;
2
3
4 import java.util.logging.ConsoleHandler;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import java.util.logging.SimpleFormatter;
8
9 /**
10  * HelloWorld is an object used to greet the world
11  */
12 public class HelloWorld {
13     static private final String DEFAULT_GREETING = "Hello World 2021!!!";
14     static private final String LOGGER_NAME = "intro.HelloWorld";
15     private String greeting;
16
17     /**
18      * Instantiates a HelloWorld object with an initial greeting
19      *
20      * @param greeting initial greeting
21      */
22     public HelloWorld(final String greeting) {
23         this.greeting = greeting;
24     }
25 }
```

Ln 14 : 79 Col 1 Sel 0 2.05 KB ANSI CR+LF INS Java Source

Completing the merge

1. Make sure there are no more merge markers
2. Save the file
3. Repeat for any other conflicts
4. `git add` the conflicting files
5. `git merge --continue`

Is this a pain?

- Yes, but it shouldn't happen too often.
- Your interaction with the shell might look somewhat different.
- But the workflow is the same:
 - identify the files that are conflicted
 - identify and resolve the conflicts in each file
 - the conflicted region will be marked with >>>'s.
 - Use your favorite text editor for this.
 - When you get the file the way you want it, add it to your commit.
 - Commit all the fixed-up files.

Summary

- In this lesson you have learned
 - what happens when you pull changes from an upstream repository
 - what a conflict is
 - how to resolve a simple merge conflict in a text file