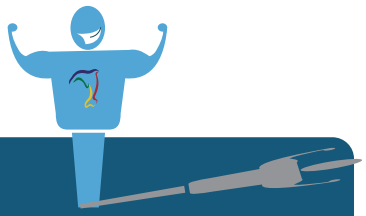
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

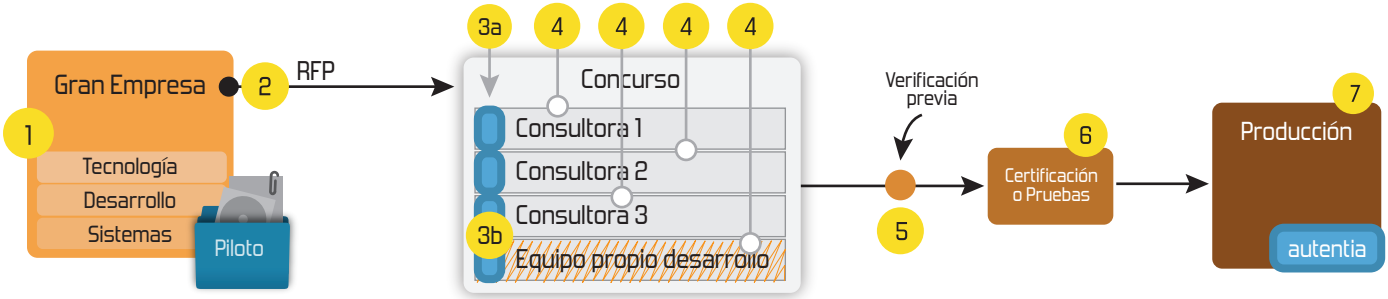
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2012-06-05 Tutorial visitado 1 veces Descargar en PDF

Comparando diferencias entre ficheros con java-diff-utils.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. El escenario.
- 4. Comparando tipos de diferencias.
- 5. Probando el ejemplo.
- 6. Referencias.
- 7. Conclusiones.

1. Introducción

Existen determinadas ocasiones en nuestro día a día donde necesitamos comparar diferencias entre ficheros. Si trabajamos con sistemas de control de versiones (CVS, Subversion, Git...) este hecho se repite aún con mayor frecuencia, sobre todo cuando tenemos un conflicto en un fichero. Normalmente este proceso se suele realizar a mano, con ayuda de herramientas que suelen traer integradas los clientes de dichos sistemas de control de versiones (véase el caso de Tortoise).

Sin embargo, pueden existir ocasiones en que necesitemos automatizar este proceso de comparación de ficheros. Pongamos el caso de una aplicación que necesita informar al usuario de que hubo cambios en una versión de un fichero con respecto a la anterior y listarle esas diferencias del mismo modo que hacen los clientes de sistemas de control de versiones.

En este tutorial veremos cómo identificar cambios entre ficheros de manera automática con ayuda de java-diff-utils, una librería Java Open Source que se distribuye bajo Licencia Apache 2.0.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Snow Leopard 10.6.7
- Entorno de desarrollo: Intellij Idea 11.1 Ultimate.
- java-diff-utils 1.2.1
- JDK 1.6
- JUnit 4.10
- Apache Maven 3.0.3

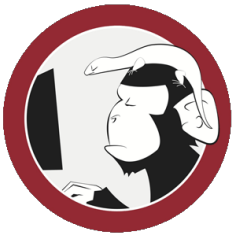
3. El escenario.

El ejemplo que vamos a ver es el siguiente. Tenemos dos ficheros a los que llamaremos originalFile.txt y revisedFile.txt. Queremos poder listar las diferencias entre ambos de forma que: podamos saber qué modificaciones ha habido con respecto al original, qué nuevas líneas se han metido y cuáles se han borrado. Además queremos saber las líneas en las que se produjo el cambio en el fichero revisedFile.txt con respecto a originalFile.txt, el número de líneas de las que están compuestos los cambios y el texto de los cambios.

Por tanto los requisitos serán:

- Poder obtener un listado de modificaciones en el fichero revisedFile.txt con respecto a originalFile.txt donde se indique en qué línea empieza el cambio, cúantas líneas tiene el cambio y el texto del que está compuesto.
- Poder obtener un listado de inserciones en el fichero revisedFile.txt con respecto a originalFile.txt donde se indique en qué línea se realizó la inserción, cúantas líneas se insertaron y el texto insertado.
- Poder obtener un listado de líneas eliminadas en el fichero revisedFile.txt con respecto a originalFile.txt y el número de cada línea que se eliminó.

Catálogo de servicios Autentia



Síguenos a través de:

Últimas Noticias

» Autentia colabora con la ONG Proyecto Ciclista Solidario

» Curso de Kanban Core en Madrid con Masa K. Maeda

» Failure demand

» Comentando el Libro: Lean StartUp (el método)

» VIII Autentia Cycling Day

Histórico de noticias

Últimos Tutoriales

» Mejorando la calidad de nuestro código PHP

» JQuery Waypoints: realizando acciones al llegar a un punto de la página con el scroll.

» Invocar a procedimientos almacenados de Oracle usando Spring JDBC

» Responsive CSS. Redimensionado de imagenes con Skeleton

» Primeros pasos con Cloud Foundry

Últimos Tutoriales del Autor

Nuestro fichero originalFile.txt será el siguiente:

```
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

Y nuestro fichero revisedFile.txt será el este:

```
Line 2
Line 3 with changes
Line 4
Line 5 with changes and
a new line
Line 6
new line 6.1
Line 7
Line 8
Line 9
Line 10 with changes
```

Como vemos hay cambios considerables entre los dos ficheros. En concreto:

- En revisedFile.txt se ha eliminado la línea 1 (Line 1) del fichero original.
- En revisedFile.txt se ha modificado la línea 2 (línea 3 en el original) añadiendo la cadena de caracteres " with changes".
- En revisedFile.txt se ha modificado la línea 4 (línea 5 en el original) añadiendo la cadena de caracteres " with changes and" y añadiendo una línea nueva "a new line".
- En revisedFile.txt se ha añadido la línea 7 ("new line 6.1") del fichero original.
- En revisedFile.txt se ha modificado la línea 11 (línea 10 en el original) añadiendo la cadena de caracteres " with changes".

Por tanto tenemos 3 grupos de cambios en el fichero revisedFile.txt con respecto a originalFile.txt:

- Modificaciones: líneas 2, 4 (línea 4 modificada y 5 añadida) y 11 del fichero revisedFile.txt
- Inserciones: línea 7 del fichero revisedFile.txt
- Eliminaciones: línea 1 del fichero originalFile.txt no está en revisedFile.txt

Y alguno puede estar pensando ahora mismo: "Ya podías haber puesto un ejemplo con menos diferencias, que ya nos hacemos una idea". Pues sí, yo también lo estoy pensando, pero así vemos mejor toda la potencia de esta librería... :)

4. Comparando tipos de diferencias.

LLegados a este punto ya tenemos claro lo que tenemos que hacer y qué resultado esperamos en el ejemplo que hemos propuesto. Ahora vamos a ver cómo se hace todo esto con java-diff-utils.

Lo que vamos a hacer es crear una clase FileComparator que contenga tres métodos públicos: getChangesFromOriginal, getInsertsFromOriginal y getDeletesFromOriginal, que nos devolverán la lista de modificaciones, inserciones y eliminaciones en el fichero revised con respecto al original.

Estas listas de cambios nos serán devueltas en forma de Chunk (clase propia de la librería) que no es más que una clase con la información relativa al cambio, como puede ser la posición desde la que empieza el cambio o las líneas que lo componen.

Nuestra clase quedaría de la siguiente forma:

```
package com.autentia.tutoriales.javadiffutils;

import difflib.Chunk;
import difflib.Delta;
import difflib.DiffUtils;
import difflib.Patch;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class FileComparator {

    private final File original;

    private final File revised;

    public FileComparator(File original, File revised) {
        this.original = original;
        this.revised = revised;
    }

    public List<Chunk> getChangesFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.CHANGE);
    }

    public List<Chunk> getInsertsFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.INSERT);
    }

    public List<Chunk> getDeletesFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.DELETE);
    }

    private List<Chunk> getChunksByType(Delta.TYPE type) throws IOException {
        final List<Chunk> listOfChanges = new ArrayList<Chunk>();
        final List<Delta> deltas = getDeltas();
        for (Delta delta : deltas) {
            if (delta.getType() == type) {
                listOfChanges.add(delta.getRevised());
            }
        }
        return listOfChanges;
    }

```

» JQuery Waypoints: realizando acciones al llegar a un punto de la página con el scroll.

» Creando un videojuego con HTML5 y Javascript

» Trabajando con Intellij IDEA 11

» JsTestDriver: testea tu código Javascript

» Cómo forzar al navegador a que descargue recursos estáticos cuando los tiene cacheados.

Últimas ofertas de empleo

2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.

```
49      private List<Delta> getDeltas() throws IOException {
50
51          final List<String> originalFileLines = fileToLines(original);
52          final List<String> revisedFileLines = fileToLines(revised);
53
54          final Patch patch = DiffUtils.diff(originalFileLines, revisedFileLines);
55
56          return patch.getDeltas();
57      }
58
59      private List<String> fileToLines(File file) throws IOException {
60          final List<String> lines = new ArrayList<String>();
61          String line;
62          final BufferedReader in = new BufferedReader(new FileReader(file));
63          while ((line = in.readLine()) != null) {
64              lines.add(line);
65          }
66
67          return lines;
68      }
69
70  }
```

Prestemos atención a la línea 54. El método diff de la clase DiffUtils recibe los ficheros original y revised como colecciones de líneas y devuelve un objeto "Patch".

El objeto "Patch" contiene una lista de Deltas (clase Delta) que no es más que la relación entre el fichero original y el revised con respecto a un cambio. Por tanto, la lista de Deltas será la lista de cambios entre el fichero original y el fichero revised. Cada Delta contendrá dos Chunk el relativo al fichero original y al revised.

Vamos a listar todos estos conceptos tan raros que acaban de surgir en un momento para que quede más claro:

- Chunk: "trozo" de fichero que ha cambiado (caso del fichero revised) o sobre el que se han hecho cambios (caso del fichero original). Contiene información relativa a este cambio como dónde empieza o de cuántas líneas está compuesto.
- Delta: mantiene la relación entre el Chunk del fichero original y el Chunk del fichero revised e indica que tipo de cambio es: modificación, inserción o eliminación.
- Patch: contiene el listado de todos los Deltas de dos ficheros comparados.

Como puede verse, el método getChunksByType, lo único que hace es, partiendo de la lista de Deltas (cambios) devolver únicamente los cambios con respecto al fichero original (chunks del fichero revised) filtrando por alguno de los tres tipos que maneja: modificaciones, inserciones o eliminaciones.

Los métodos públicos: getChangesFromOriginal, getInsertsFromOriginal y getDeletesFromOriginal, llaman al método getChunksByType filtrando por cada tipo de cambio que corresponda.

Pues esto es todo lo que necesitamos para tener nuestro comparador de cambios entre ficheros. Fácil ¿no?.

5. Probando el ejemplo.

Pues lo último que queda es probar si este invento funciona o no. Para ello vamos a testear (JUnit) nuestra clase probando con los dos ficheros que presentamos en el punto 3.

Realizamos 3 test, uno por cada uno de los métodos públicos de la clase FileComparator. La clase de test es la siguiente:

```
1   package com.autentia.tutoriales.javadiffutils;
2
3
4   import difflib.Chunk;
5   import org.junit.Test;
6
7   import java.io.File;
8   import java.io.IOException;
9   import java.util.List;
10
11  import static org.junit.Assert.assertEquals;
12  import static org.junit.Assert.fail;
13
14  public class FileComparatorTest {
15
16      private final File original = new File("./files/originalFile.txt");
17
18      private final File revised = new File("./files/revisedFile.txt");
19
20      @Test
21      public void shouldGetChangesBetweenFiles() {
22
23          final FileComparator comparator = new FileComparator(original, revised);
24
25          try {
26              final List<Chunk> changesFromOriginal = comparator.getChangesFromOriginal();
27              assertEquals(3, changesFromOriginal.size());
28
29              final Chunk firstChange = changesFromOriginal.get(0);
30              final int firstLineOfFirstChange = firstChange.getPosition() + 1;
31              final int firstChangeSize = firstChange.size();
32              assertEquals(2, firstLineOfFirstChange);
33              assertEquals(1, firstChangeSize);
34              final String firstChangeText = firstChange.getLines().get(0).toString();
35              assertEquals("Line 3 with changes", firstChangeText);
36
37              final Chunk secondChange = changesFromOriginal.get(1);
38              final int firstLineOfSecondChange = secondChange.getPosition() + 1;
39              final int secondChangeSize = secondChange.size();
40              assertEquals(4, firstLineOfSecondChange);
41              assertEquals(2, secondChangeSize);
42              final String secondChangeFirstLineText = secondChange.getLines().get(0).toStr
43              final String secondChangeSecondLineText = secondChange.getLines().get(1).toSt
44              assertEquals("Line 5 with changes and", secondChangeFirstLineText);
45              assertEquals("a new line", secondChangeSecondLineText);
46
47              final Chunk thirdChange = changesFromOriginal.get(2);
48              final int firstLineOfThirdChange = thirdChange.getPosition() + 1;
49              final int thirdChangeSize = thirdChange.size();
50              assertEquals(11, firstLineOfThirdChange);
51              assertEquals(1, thirdChangeSize);
52              final String thirdChangeText = thirdChange.getLines().get(0).toString();
53              assertEquals("Line 10 with changes", thirdChangeText);
54
```

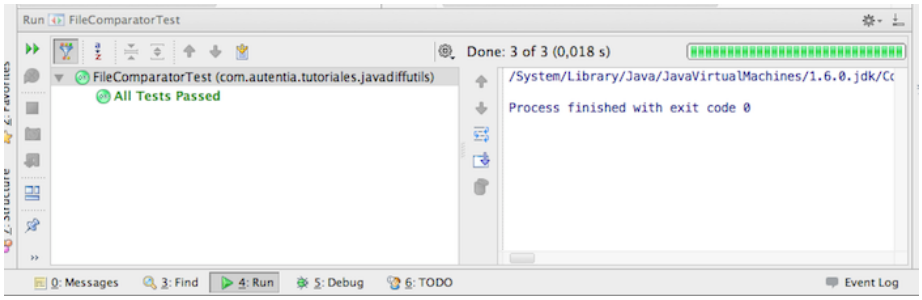
```
55          } catch (IOException ioe) {
56              fail("Error running test shouldGetChangesBetweenFiles " + ioe.toString());
57          }
58      }
59
60      @Test
61      public void shouldGetInsertsBetweenFiles() {
62
63          final FileComparator comparator = new FileComparator(original, revised);
64
65          try {
66              final List<Chunk> insertsFromOriginal = comparator.getInsertsFromOriginal();
67              assertEquals(1, insertsFromOriginal.size());
68
69              final Chunk firstInsert = insertsFromOriginal.get(0);
70              final int firstLineOfFirstInsert = firstInsert.getPosition() + 1;
71              final int firstInsertSize = firstInsert.size();
72              assertEquals(7, firstLineOfFirstInsert);
73              assertEquals(1, firstInsertSize);
74              final String firstInsertText = firstInsert.getLines().get(0).toString();
75              assertEquals("new line 6.1", firstInsertText);
76
77          } catch (IOException ioe) {
78              fail("Error running test shouldGetInsertsBetweenFiles " + ioe.toString());
79          }
80      }
81
82      @Test
83      public void shouldGetDeletesBetweenFiles() {
84
85          final FileComparator comparator = new FileComparator(original, revised);
86
87          try {
88              final List<Chunk> deletesFromOriginal = comparator.getDeletesFromOriginal();
89              assertEquals(1, deletesFromOriginal.size());
90
91              final Chunk firstDelete = deletesFromOriginal.get(0);
92              final int firstLineOfFirstDelete = firstDelete.getPosition() + 1;
93              assertEquals(1, firstLineOfFirstDelete);
94
95          } catch (IOException ioe) {
96              fail("Error running test shouldGetDeletesBetweenFiles " + ioe.toString());
97          }
98      }
99
100 }
```

Como vemos, en cada test lo que se está haciendo es llamar al método correspondiente que se quiere testear y comprobar si el número de cambios del tipo que corresponda es el esperado.

Luego se comprueba que cada Chunk comience en la línea que se espera. La clase Chunk contiene un método getPosition que devuelve la posición en el fichero donde empiezan los cambios de la misma forma en que lo hace un array o una colección (0 la primera posición, 1 la segunda, etc...) por lo que para obtener la línea donde empieza debemos sumar 1 al valor que retorna.

Además, por cada Chunk comprobamos que tenga el número de líneas y texto esperados.

Lanzamos los test y todo funciona como queremos... :D



6. Referencias.

- java-diff-utils

7. Conclusiones.

En este tutorial hemos visto lo sencillo que es obtener de forma automática diferencias entre ficheros gracias a la librería java-diff-utils. Por ponerle un "pero" diría que le falta un poco de documentación. En su defensa destacar que es bastante intuitiva de utilizar.

Como siempre, recordaros que podéis hacer lo que queráis con el código fuente expuesto en este tutorial.

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

IMPULSA Impulsores Comunidad ¿Ayuda?

---
sin clicks

0 personas han traído clicks a esta página

+ + + + + + + +

powered by karmacracy

Copyright 2003-2012 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

W3C XHTML 1.0 W3C CSS XML RSS XML ATOM