Sviluppo del progetto

Descrizione

- scrivere un comando che prenda un albero di directory e ritorni un file testuale che lo descriva (e il viceversa) tale per cui gestendo il file risultante con RCS si ottenga una gestione delle configurazioni soddisfacente. Opzionalmente gestire anche i link simbolici.

Spiegazione aggiuntiva:

- il file in output deve comprendere anche il contenuto dei file sia testuali che binari ma solo i binari devono essere convertiti in formato tale da poter visionare XML di output anche con un banalissimo cat/editor di solo testo.

Bibliografia

RCS (revision control system):

- http://www.cs.purdue.edu/homes/trinkle/RCS/ [http://www.cs.purdue.edu/homes/trinkle/RCS/]
- http://www.ibiblio.org/pub/Linux/docs/HOWTO/translations/it/html_single/RCS.html.gz [http://www.ibiblio.org/pub/Linux/docs/HOWTO/translations/it/html_single/RCS.html.gz]

Possibile formato di uscita del file: XML

- http://it.wikipedia.org/wiki/XML [http://it.wikipedia.org/wiki/XML]
- http://www.w3.org/XML [http://www.w3.org/XML]
- http://www.w3schools.com/xml/default.asp [http://www.w3schools.com/xml/default.asp]
- http://www.xml.org [http://www.xml.org]

Sintassi del file di uscita

Esempio delle keywords necessarie:

```
<directory>
  <name>nome_directory</name>
  <ownerid>501</ownerid>
  <groupid>100</groupid>
  <permission>rwxrwx---</permission>
  <mtime>1165483079</ctime>
  <ctime>1165333390</ctime>
</directory>

<file>
  <name>nome_file</name>
  <ownerid>501</ownerid>
  <groupid>100</groupid>
  <permission>rwxrwx---</permission>
  <mtime>1165483080</ctime>
  <ctime>1165333391</ctime>
</file>

<link>
  <name>nome_link</name>
  <ownerid>501</ownerid>
  <groupid>100</groupid>
  <destination>../../../altro/file</destination>
  <mtime>1165483083</ctime>
```

```
  <ctime>1165333392</ctime>
</link>
```

Esempio di albero:

```
<directory>
  <directory>
   ...
  </directory>
  <directory>
    <file>
     ...
    </file>
    <link>
     ...
    </link>
  </directory>
    <file>
     ...
    </file>
</directory>
```

Evoluzione della sintassi

Nel progettare e nel realizzare il software sono venuti alla luce alcuni dettagli inzialmente non presi in considerazione che richiedono un'analisi aggiuntiva e una modifica/evoluzione della sintassi:

1. La sintassi XML sopra ha il difetto di non essere autoesplicativa riguardo all'uso di caratteri particolari quali spazi e minore/maggiore all'interno di nomi e path quindi occorre inserire il valore fra apici:

```
<file>
  <name>"la bella va in campagna.doc"</name>
  ...
</file>
```

oppure utilizzando una sintassi alternativa che sarebbe anche piu' consona a distinguere campi che richiedono di essere messi su una sola linea per una questione di leggibilita' anche dal punto di vista di un diff:

```
<file>
  <name="la bella va in campagna.doc" />
  ...
</file>
```

2. Il campo dei permessi deve essere espresso in termini di 6 cifre ottali per comprendere tutto il possibile spettro di permessi inoltre il tag potrbbe anche cambiare nome perche' in quel campo non vengono salvati solo i permessi di tipo rwx:

```
<file>
  <mode="100664" />
  ...
</file>
```

bisogna ancora che faccia una valutazione per capire se davvero tutte e sei le cifre ottali sono realmente significative nel rispristino dei dati oppure ne bastino solamente le prime 4, quelle che generalmentesi danno in pasto al comando chmod.

3. L'inserimento del contenuto dei file puo' essere supportato attraverso una sintassi che specifichi quanti bytes il parser debba saltare per evitare conusione fra la sintassi XML corrente e quella che eventualmente fosse contenuta nel file da includere:

```
                             _ il contatore della lunghezza parte
<file>                      |  dopo il primo \n dopo il > del tag
  <content <length="1234">\n|
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX|\n
  </content>                    |
  ...                           |_ il contatore della lunghezza termina
</file>                            e immediatamente dopo segue un \n
```

riguardo ai file binari questi sono individuati in quanto contengono caratteri diversi da 0×09 (tabulation), 0x0A (line feed), 0x0D (form feed) oppure esterni all'intervallo 32 (spazio) - 126 (tilde) della codifica ASCII [http://www.asciitable.com/] e per poterli inserire all'interno del campo =content= venogno tradotti con il medesimo algoritmo di uuencode/uudecode cioe' in base64.

La sopracitata sintassi viola lo standard XML che prevede l'escaping di alcuni caratteri oppure l'uso della sezione CDATA con l'escaping di]]> ma dal mio punto di vista entrambe le soluzioni previste dall'XML non riproducono con totale fedelta' il file testuale contenuto nell'XML rispetto a quello originale. Allo stato attuale i due valori fedelta' e standard XML sembrano mutuamente esclusivi.

Se invece si vuole rispettare sintatticamente lo standard la seguente propone un compromesso anche abbastanza ragionevole:

```
<file>
  <content length="1234">
<![CDATA[
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
]]><endtag num="2"><![CDATA[
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
]]>
  </content>
  ...
</file>
```

dove il tag <content> premette di individuare quale porzione il parser specializzato puo' tranquillamente saltare mentre per un controllore sintattico il formato XML sarebbe rispettato perche' ad esempio la stringa]]>]]> spezza la sezione CDATA, inserisce un tag <endtag> con valore duplice e poi riprende la sezione CDATA. In questo modo pero' un file XML con CDATA che volessimo includere finirebbe per essere alterato in modo non banale quindi non piu' facilmente leggibile.

Successivamente **Claudio scrive** per mail:

```
  A seconda dei vincoli messi nella DTD/Schema ci sono due possibilita':

   1. i blanks all'interno dell'elemento sono significativi
   2. i blanks all'inizio e fine del contenuto dell'elemento debbono essere ignorati
```

Algoritmo di escaping: se nel file di testo o nel buffer base64 encoded si trova la sequenza]]> allora lo lascio e la CDATA e' automaticamente chiusa, quindi inserisco <endtag per="N"> dove N e' il numero di ripetizioni della sequenza]]> e infine apro un'altra CDATA e proseguo con il rimanente del buffer.

Algoritmo di embedding testo: utilizzo <![CDATA[%s]]> dove %s e' il testo comprensivo dell'escaping sopra citato

Algoritmo di embedding binario: utilizzo base64 di uuencode e mi riporto al caso di embedding di testo.

```
<file>
```

```
  <content>
<![CDATA[XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX]]>
<endtag per="2">
<![CDATA[XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXX]]>
  </content>
  ...
</file>
```

ANCORA DA CAPIRE: nella sintassi `<![CDATA[%s]]>` se la stringa `%s` e' testo che comincia con degli spazi allora il punto 2 del `DTD/Schema` prevede che debbano essere ignorati ma questo probabilmente non riguarda la sezione `CDATA`.

4. Valutare quale algoritmo di sorting sia piu' indicato per minimizzare il confronto differenziale fra due revisioni di questo file XML. Probabilmente quello basato su creation time in ordine crescente perche' l'aggiunta di un file con contenuto simile ad uno precedente non confonde diff che se lo trova invece accodato in fondo (aggiunto). **OPZIONALE**

5. Integrando xmlParser (scelto perche' e' il piu' piccolo e quello che mi pareva piu' semplice) la sintassi del file e' stata modificata come segue:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<XMLTAR version="0.0.8" compiled="Dec 15 2006 14:05:13">
  <Header
    workdir="/home/roberto/Projects/fsexplore/src"
    datetime="Sun Apr  7 09:04:44 1974" />
  <file
    name="Makefile.in"
    mode="100664"
    ownerid="500"
    groupid="500"
    mtime="1166093776"
    ctime="1166093776">
    <content binary="FALSE" length="17009">
<![CDATA[.....................................................................
.....................................................................
.... ............]]>
<endtag per="3">
<![CDATA[.....................................................................
.......]]>;
    </content>
  </file>
  <directory
    name="test"
    mode="040775"
    ownerid="500"
    groupid="500"
    mtime="1166094263"
    ctime="1166094263">
    <link
      name="biciao"
      refer="pippo/ciao"
      mode="120777"
      ownerid="500"
      groupid="500"
      mtime="1165849451"
      ctime="1166018260">
    </link>
  <directory>
</XMLTAR>
```

anche perche' la precedente sintassi era probabilmente errata in quanto ammetteva tag senza nome e contenenti un attributo come ad es.: `<name="ciao">`.

6. Integrando xmlParser nella fase 2 mi sono accorto che `<endtag>` è fondamentalmente inutile:

```
  <file
    ...
    ... >
    <content binary="FALSE" length="17009">
<![CDATA[.....................................................................
.....................................................................
.... ............]]>
<![CDATA[]]>
<![CDATA[]]>
<![CDATA[]]>
<![CDATA[.....................................................................
.......]]>;
    </content>
  </file>
```

Sviluppo software

FASE 1

Progettazione e sviluppo del software in grado di attraversare un albero su un filesystem e trasformarlo in un archivio XML

- fsexplore-001.tar.gz: Primo rilascio dell'esploratore di filesystem
- fsexplore-002.tar.gz: Secondo rilascio che gestisce la sintassi XML
- fsexplore-003.tar.gz: Terzo rilascio che supporta XML ricorsivo
- fsexplore-004.tar.gz: Quarto rilascio: supporta XML ricorsivo (questo funziona)
- fsexplore-005.tar.gz: Quinto rilascio: supporta XML ricorsivo con inclusione di file (solo testuali) - aggiunte librerie di I/O file e incluso il tutto in un progetto anjuta
- fsexplore-006.tar.gz: Sesto rilascio: supporta XML con sezioni CDATA e con uuencode in base64 per includere il contenuto dei files (corregge la ricorsione nelle directory)
- fsexplore-007.tar.gz: Settimo rilascio: migliorata la parte relativa alla libreria I/O file
- fsexplore-008.tar.gz: Ottavo rilascio: intestazione XML. cambio directory, primitiva gestione delle opzioni da riga di comando, inizio test di interpretazione XML.
- fsexplore-009.tar.gz: Nono rilascio: cambio della sintassi in maniera conforme allo standard XML e inizio sviluppo della parte xml→fs-tree

TODO: cose da fare (o che si potrebbero fare) in questa fase

- trasformare =fsexplore= in un programma capace di elaborare opzioni da riga di comando al fine di trasformarlo in un archiviatore — **done in 016**
- includere la gestione di stringhe (farne una libreria) capaci di allocarsi dinamicamente per superare il limite del MAX_SIZE_PATH — **useless in 018**
- migliorare la libreria di I/O file per gestire correttamente una lettura di zero bytes che pero' ritorni un =EGAIN= — **done in 007**
- nel caso un file o una directory non siano esplorabili modificare temporaneamente i permessi per includerli — **done in 017**

XML PARSER

- The Expat XML Parser [http://expat.sourceforge.net/] - Expat is an XML parser library written in C.
  - SCEW [http://www.nongnu.org/scew/] - Simple C Expat Wrapper
- tinyxml [http://www.grinninglizard.com/tinyxml/] - !TinyXml is a simple, small, C++ XML parser that can be easily integrating into other programs.
  - !TinyXML is released under the zlib license
- xmlParser [http://iridia.ulb.ac.be/~fvandenb/tools/xmlParser.html] - Small, simple, cross-platform, free and fast C++ XML Parser.
  - xmlParser is released under BSD licence — **ho scelto di usare questo perchè piccolo e semplice**
- Parsifal [http://www.saunalahti.fi/~samiuus/toni/xmlproc/] - Parsifal is a validating XML 1.0 parser written in ANSI C. Parsifal API is based on SAX2.
- w3c XML parser [http://dev.w3.org/cvsweb/XML/parser.c] - questo link porta ad un CVS log di un parser ancora trovare.
- irrXML [http://xml.irrlicht3d.org/] - irrXML is a simple and fast open source xml parser for C++.

FASE 2

Progettazione e sviluppo del software in grado di ricreare un albero su un filesystem a partire da un archivio XML

- xmltar-010.tar.bz2: Decimo rilascio: rinomina e riorganizzazione del progetto per il passaggio alla fase 2 di sviluppo
- xmltar-011.tar.bz2: Undicesimo rilascio: creazione dell'albero ma senza contenuti a partire dal file XML, mancano anche proprietario e gruppo
- xmltar-012.tar.bz2: Dodicesimo rilascio: modificata la sintassi XML riguardo allo <endtag> che è superfluo, prima versione capace di scrivere il contenuto
- xmltar-013.tar.bz2: Tredicesimo rilascio: scrittura e settaggio dell'albero delle directory in due passaggi + inizio riconversione uuencode/uudecode
- xmltar-014.tar.bz2: Quattordicesimo rilascio: modifica marginale sintassi XML per file vuoti, corretto uuencode/uudecode, produzione di albero con file anche binari
- xmltar-015.tar.bz2: Quindicesimo rilascio: ristrutturato il codice per prevedere tutti i possibili errori, aggiunta la gestione di stampa e codice errore, corretto libiofile.
- xmltar-016.tar.bz2: Sedicesimo rilascio: aggiunta e gestione di una sintassi simile a tar e iniziato ad aggiungere anche un supporto sperimentale per la compressione.
- xmltar-017.tar.bz2: Diciasettesimo rilascio: semplificato il codice con l'aggiunta di funzioni, aggiunta la gestione dei tempi di accesso e di modifica e il proprietario e il gruppo anche per i link mentre il tempo di creazione pare non modificabile a posteriori.
- xmltar-018.tar.bz2: Diciottesimo rilascio: integrata la smplstr ma risulta inutile il suo uso, eliminata

una stampa di debug inutile, controllo generale del codice

- xmltar-019.tar.bz2: Dicianovesimo rilascio: controllo generale, aggiustamenti minori, corretta l'estensione della libreria di I/O file, inclusi anche i file che iniziano con punto – **superato la prova /etc**

TODO: cose da fare (o che si potrebbero fare) in questa fase

- includere e utilizzare la libreria di stringhe per superare il limite di `chdir(dirname)` e `chdir("..")` — **useless in 018**
- sbobinare l'albero in due passaggi: il primo (ANDATA) crea l'labero mentre il secondo (RITORNO) pone i permessi e le proprietà dei file/directory; questo per evitare che i permessi memorizzati siano tali che non è possibile scrivere la struttura (mentre era possibile leggerla) — **done in 013**
- utilizzare il xmlParser anche per la porzione di codice che si occupa della scrittura del file XML — **opzionale**
- verificare corner case nei nomi dei file e delle directory.

LIMITI

Alcuni limiti strutturali dell'architettura POSIX/UNIX

- non è possibile cambiare il creation time dei file/link/directory
- non è possibile cambiare il modtime e l'access time dei link
- solo root puo' impostare tutte le combinazioni gruppo/utente e permessi in modo corrispondente all'originale, gli utenti hanno alcune limitazioni
- se usato da root puo' causare problemi di sicurezza nel caso che la tavola utenti/gruppi siano cambiati oppure siano diversi da una macchina all'altra. Si tratta di un problema mitigabile introducendo un controllo che utente/uid e gruppo/gid corrispondano ma nel caso di una macchina differente in cui pur siano soddisfatte le corrispondenze non è possibile verificare che a utenti/gruppi corrispondano davvero le medesime persone.