Unix-basierte Implementierung kleiner Datenbanken

Prof. Dr. Franz Schweiggert mit Norbert Heidenbluth
Wintersemester 2007/2008

Übungsblatt 05
(Stichtag: 28.11.2007)

Aufgabe 1: Das Unix-DB-SyncTool (5 Punkte)

Für die erste Aufgabe in dieser Woche benötigen wir nur zwei „Zutaten“: das test-Kommando der bash sowie eine vage Erinnerung an AI2 und das dort behandelte Thema „Rekursive Funktionen“. Aha! Worum geht's?

Nehmen wir an, Ihr habt ein Verzeichnis in Eurem Heimatverzeichnis, das eine sehr große Anzahl von Dateien sowie weiteren Unterverzeichnissen enthält, wobei in letzteren wieder viele weitere Dateien sowie Unterverzeichnisse stecken, wobei in letzteren wieder viele weitere Dateien sowie Unter.....
(Okay - soviel zur Rekursion :-))

Jetzt nehmen wir weiter an, Ihr archiviert in dieser kompletten Unterstruktur wichtige Dateien (z.B. Fotos oder MP3-Files). Die Inhalte der vorhandenen Dateien verändern sich nicht mehr, aber es kommen regelmäßig neue (Dateien / Verzeichnisse) hinzu. Weil Ihr das Thema Datensicherung ernst nehmt, möchtet Ihr die gesamte Unterverzeichnisstruktur regelmäßig an einen anderen Speicherort (externe Festplatte, anderer Rechner, etc.) kopieren (auch „spiegeln“ genannt). Der einfachste Weg wäre da natürlich, jedesmal einfach alles zu kopieren. Aber das hat zwei Nachteile: erstens kopiert man unnötig viele Daten (das meiste ist ja ohnehin schon da, es kommt nur auf das Neue an), und zweitens kann man bei diesem Vorgehen sehr schnell Inkonsistenzen bekommen. (Wer kontrolliert, ob zwischenzeitlich nicht versehentlich auf dem Backup-Medium Daten gespeichert wurden, die auf dem „Original“ nicht vorhanden sind und durch den Kopiervorgang nun evtl. verloren gehen?)

Wer auf die Frage im letzten Satz mit „Wir, die Unix-DB'lerInnen“ antwortet, hat bereits die Aufgabenstellung herausgefunden! Und für alle anderen:

>> *Schreibt ein kleines Shell-Skript, das zwei (absolute) Pfadangaben als Parameter übergeben bekommt und diese beiden Pfade vollständig (d.h. rekursiv) durchläuft. Dabei soll eine Ausgabe („Protokoll“) entstehen, welche diejenigen Datei- bzw. Verzeichnisnamen auflistet, die in „dem einen“, nicht aber „dem anderen“ Verzeichnis enthalten sind. (D.h., die Überprüfung soll in beiden Richtungen vorgenommen werden.)* <<

Anm.: Mit Hilfe des Protokolls könnte man in einem zweiten Schritt dann nur die dort genannten Dateien kopieren lassen (und somit Problem 1 beheben), und man wird auf die Abweichungen zwischen Quelle und Ziel aufmerksam gemacht und kann sich dann um Problem 2 kümmern. ***Aber:*** Das ist nicht Gegenstand dieser Aufgabe - hier geht es nur um das Shell-Skript zur Erzeugung des Protokolls.

Beispiel:

Usage-Meldung ausgeben:
```
turing$ ./syncTool.sh
Usage: ./syncTool.sh dir1 dir2
```

Inhalt von Verzeichnis testDir1:
```
turing$ ls -l
drwxrwsr-x [...] myDir1
drwxrwsr-x [...] myDir2
-rw-rw-r-- [...] myFile1
-rw-rw-r-- [...] myFile2
```

Inhalt von Verzeichnis testDir2:
```
drwxrwsr-x [...] myDir1
-rw-rw-r-- [...] myFile1
-rw-rw-r-- [...] topSecretRootPassword
```

Inhalt von Verzeichnis testDir1/myDir1:
(diese Datei fehlt in testDir2/myDir1)
```
-rw-rw-r-- [...] justAText.txt
```


Beispiel (Fortsetzung):

Tool starten:
```
turing$ ./syncTool.sh ~/demoBlatt05/testDir1 ~/demoBlatt05/testDir2
Checking if content of /home/thales/heidi/demoBlatt05/testDir1 exists in /home/
thales/heidi/demoBlatt05/testDir2...
justAText.txt not exisitent in /home/thales/heidi/demoBlatt05/testDir2/myDir1
Whole subdirectory myDir2 not exisitent in /home/thales/heidi/demoBlatt05/testDir2
myFile2 not exisitent in /home/thales/heidi/demoBlatt05/testDir2
...DONE
--------------------
Checking if content of /home/thales/heidi/demoBlatt05/testDir2 exists in /home/
thales/heidi/demoBlatt05/testDir1...
topSecretRootPassword not exisitent in /home/thales/heidi/demoBlatt05/testDir1

All done :-)
```

Aufgabe 2: Fred Feuersteins Steinzeit-SLC (12 Punkte)

Wer am WiMa-Tag beim Web-2.0-Vortrag war, hat einen kleinen Einblick bekommen, wie unser neues SLC-System („SLC 2010“) aussehen wird. Heute drehen wir das Rad der Zeit aber mal zurück und stellen uns vor, wie SLC in einer ganz frühen Version (SLC „Stone Age Edition“) ausgesehen haben könnte.

Damals wurden die Logins der Studierenden in einen Stein namens „`studis.txt`“ gemeißelt. Die Struktur des Inhalts war dabei zeilenweise wie folgt:

```
login:realname
```

(realname heißt hier Vor- und Nachname(n)).

Auf verschiedenen Schiefertafel (je genau eine pro Vorlesung) wurden die Ergebnisse der Übungsaufgaben sowie der Klausur eingeritzt - und zwar mit der folgenden Struktur:

```
login:Blatt 1:Blatt 2:[...]:Klausur
```

Das heißt, wir gehen hier davon aus, dass es pro Veranstaltung zwar beliebig viele Übungsblätter, aber nur genau eine Klausur gibt.

Desweiteren enthalten diese Dateien stets eine Zeile für den (fiktiven) Login-Namen „maxPoints“, in welcher die Maximal-Punktzahl der einzelnen Übungsblätter bzw. der Klausur eingetragen ist.

Diese Schiefertafeln sowie der „`studis.txt`“-Stein können per Brieftaube von unserer Vorlesungshomepage geholt werden.

Baut nun einen Bash-Abakus (a.k.a. „Shell-Skript“), der als erstes Argument den Namen der Datei, äh - Schiefertafel - einer bestimmten Veranstaltung bekommt und als zweites Argument die Angagbe, ob sich die Auswertung auf die Übungsblätter oder die Klausur beziehen soll. Das Ergebnis der Auswertung(en) soll dann den auf der nächsten Seite gezeigten Beispielen ähnlich sehen.

Die Verwendung des Hilfsmittels „elektrischer Strom“ ist für dieses Übungsblatt ausdrücklich gestattet! Feuersteine und Keulen können ebenfalls verwendet werden. Wilma darf jedoch nicht mit ins Tutorium kommen!

Beispiele:

Usage-Meldung:
```
turing$ ./slcStoneAge.sh
Usage: ./slcStoneAge.sh filename mode
```

Ungültiger Auswertungsmodus:
```
turing$ ./slcStoneAge.sh ana.txt abc0815
abc0815 ist ein ungueltiger Modus!
```

Angegebene Datei existiert nicht:
```
turing$ ./slcStoneAge.sh udb.txt uebungen
Die angegebene Datei udb.txt existiert nicht!
```

Auswertung nach Übungen:
```
turing$ ./slcStoneAge.sh ana_1.txt uebungen
Maximale Punkte der Uebungsblaetter: 40
Michaela Schmidt (Login: michi) hat insgesamt 12 Uebungspunkte (= 30 %).
Kurt Maier (Login: km007) hat insgesamt 35 Uebungspunkte (= 87 %).
Josef Huber (Login: joe) hat insgesamt 38 Uebungspunkte (= 95 %).
Heinz Mueller (Login: heinzi) hat insgesamt 11 Uebungspunkte (= 27 %).
```

Auswertung nach Klausuren:
```
turing$ ./slcStoneAge.sh la.txt klausur
Maximale Punkte in der Klausur: 100
Heinz Mueller (Login: heinzi): 90 von 100 (= 90 %) -- bestanden
Kurt Maier (Login: km007): 0 von 100 (= 0 %) -- nicht bestanden
Michaela Schmidt (Login: michi): 0 von 100 (= 0 %) -- nicht bestanden
Josef Huber (Login: joe): 56 von 100 (= 56 %) -- bestanden
```

Aufgabe 3: Fred Feuersteins Steinzeit-SLC - Version 2.0 (3 Punkte)

Nur eine kleine Aufgabe zum Nachdenken - es ist keine Implementierung erforderlich:

Die Einschränkung, dass es nur genau eine Klausur pro Veranstaltung gibt, ist ja nicht realitätsnah. Wie könnte man das Programm dahingehend abändern / ergänzen, dass auch beliebig viele Klausuren verwaltet werden können?

Wie gesagt: Ihr braucht dies nicht zu programmieren!