Übung: Unix
Einführung: Aufgabe 2

Malte Heins
Helga Karafiat

Themen

- **SVN-Benutzung Teil 2**
  - Hilfreiche SVN-Befehle auf der Konsole, insbesondere Statusinformationen
  - Erzeugen und Lösen eines SVN-Konflikts auf der Konsole
- **Auswertungstool**
  - Anwendung von Pipes und Filtern
- **Eigenständiges Testen**
  - Weiterentwicklung der Fähigkeiten zum Software testen

Auswertungstool

- Ort: **ueb02**
- Name: **worst.sh**

- Auswerten von Daten aus einem fiktiven Umfragetool
- Lesen von **stdin**
- Schreiben auf **stdout** und in eine **angegebene Datei**
- Verwendung des ersten Kommandozeilenparameters (Dateiname)

Auswertungstool

- Eingabe
  - 3 Felder pro Zeile: Kennzahl, Bewertung, Bezeichnung
  - Fehlerhafte Zeilen enthalten „***error***“ (beliebige Groß-/ Kleinschreibung)
- Verarbeitung der Eingabedaten
  - Ignorieren von Fehler-Zeilen
  - Absteigende Sortierung nach dem zweiten Feld (Bewertung),
    wenn nicht eindeutig, dann zusätzlich nach dem ersten Feld (Kennzahl)
- Ausgabe
  - Datei: Sortierte Eingabedaten ohne Fehlereinträge
  - Standardausgabe: Bezeichnung des schlechtesten Eintrags (ohne weitere Infos)
  - Bereits vorhandene Dateien werden überschrieben

Auswertungstool - Beispiel

- Eingabedaten

  122,1,Foo
  112,2,Bar
  111,2,Batz

- Inhalt Ausgabedatei

  112,2,Bar
  111,2,Batz
  122,1,Foo

- Ausgabe auf **stdout**

  Foo

Anforderungen Auswertungstool

- Anforderungen aus Aufgabe 1 (Dokumentation, Hinweise zu Skripten etc.) gelten in jeder Aufgabe
- Erlaubter Umfang / Ausgestaltung des Skriptes
  - Einzeiler, abgesehen von Kommentaren und Shebang
  - Maximal sechs Filteraufrufe und fünf Pipes
  - Alle Filteraufrufe müssen durch Pipes verbunden sein
  - Auf effiziente Implementierung achten (Reihenfolge Filteraufrufe)

- Servertest
  - Verfügbar nach Ankündigung in der Newsgroup
  - Achtet darauf euer Skript rechtzeitig vor dem Servertest einzuchecken
  - Servertestergebnis geht mit 2/3 in die Bewertung der Aufgabe mit ein
- Eigenes Testen
  - Weiterhin kein eigener automatisierter Test gefordert (aber möglich)
  - Aber Vorhandensein von ausreichenden Testdaten, welche die interessanten Testfälle für dieses Skript beinhalteten und dazu passende Beschreibung oder Testbench (angelehnt an unsere vorgegebene arnold-Testbench)
- Nachweise über SVN-Benutzung auf der Konsole
  - Screenshots und beschreibende Textdatei
  - Geht mit 1/3 in die Bewertung der Aufgabe mit ein

Hilfe zum eigenständigen Testen

- Vorgabe Beispieltest
  - Eingabedatei (beispieltest.in)
  - Erwartete Ausgabe (beispieltest.exp)
  - Arnold-Testbench (beispieltest.arnold)
- Aufruf:

  ./arnold -c ./worst.sh beispieltest.arnold

- Bei dieser Aufgabe zwei sinnvolle Testarten
  - Test der Standardausgabe (direkt in Arnold)
  - Vergleich von Ausgabedatei mit erwarteter Datei (mit Hilfe von **diff**)
- Hinweis: Beliebig viele Tests pro Testbench möglich

Hilfe zum eigenständigen Testen (Testbench)

```
test {
   name    = Beispieltest Teil 1 (stdout)
   comment = Testet die Standardausgabe
   type    = param
   cmd     {cat beispieltest.in | DUT beispieltest.out}
   expect  {^SoRichtigSchlecht$}
}
test {
   name    = Beispieltest Teil 2 (Datei)
   comment = Testet die geschriebene Datei aus Basistest Teil 1
   type    = param
   cmd     {diff beispieltest.out beispieltest.exp}
   expect  {^$}
}
```

- **cmd**: Befehlskette, die ausgeführt wird, DUT = device under test (zu testendes Programm)
- **expect**: erwartete Ausgabe auf **stdout**
  - ^ = Zeilenanfang, $ = Zeilenende
    => in der Zeile darf nur das zwischen ^ und $ angegebene stehen, sonst nichts
  - **diff**: Tool, welches Dateien auf Unterschiede untersucht
    wenn Dateien identisch sind, keine Ausgabe auf **stdout** (^$ = Ausgabe leer)

Hilfe zum eigenständigen Testen (Ausgabe von Arnold - Erfolgsfall)

```
Arnold is now runnning in continuous-mode! Thank you !!!
####################
# Test 'Beispieltest Teil 1 (stdout)' in progress...
# Testet die Ausgabe des Tools auf der Standardausgabe
####################
CMD  : 'cat beispieltest.in | ./worst.sh beispieltest.out'
EXP  : '^SoRichtigSchlecht$'
OUT  : 'SoRichtigSchlecht'
 ==> --- Beispieltest Teil 1 (stdout) OK ---

####################
# Test 'Beispieltest Teil 2 (Datei)' in progress...
# Testet die in Beispieltest Teil 1 geschriebene Datei
####################
CMD  : 'diff beispieltest.out beispieltest.exp'
EXP  : '^$'
OUT  : ''
 ==> --- Beispieltest Teil 2 (Datei) OK ---


Test-Statistik:
===============
  Testname: Beispieltest Teil 1 (stdout) OK
  Testname: Beispieltest Teil 2 (Datei) OK

Alles klar, gut und so. Von dem Test IHM sein Ende!
```

- Alles gut, Programm tut was es laut Test soll
- CMD: Aufruf, der zu der Ausgabe geführt hat (kann man direkt so kopieren und ausführen)
- EXP: Was erwartete der Test?
- OUT: Was hat er bekommen?

Hilfe zum eigenständigen Testen (Ausgabe von Arnold - Fehlerfall)

```
Arnold is now runnning in continuous-mode! Thank you !!!
####################
# Test 'Beispieltest Teil 1 (stdout)' in progress...
# Testet die Ausgabe des Tools auf der Standardausgabe
####################
CMD  : 'cat beispieltest.in | ./worst.sh beispieltest.out'
EXP  : '^SoRichtigSchlecht$'
OUT  : 'wasganzanderes'
 ==> --- Beispieltest Teil 1 (stdout) failed ---

####################
# Test 'Beispieltest Teil 2 (Datei)' in progress...
# Testet die in Beispieltest Teil 1 geschriebene Datei
####################
CMD  : 'diff beispieltest.out beispieltest.exp'
EXP  : '^$'
OUT  : '1c1,3
<
---
> 100,3,NochWenigerSchlecht
> 19,1,NichtGanzSoSchlecht
> 7,0,SoRichtigSchlecht'
 ==> --- Beispieltest Teil 2 (Datei) failed ---


Test-Statistik:
===============
  Testname: Beispieltest Teil 1 (stdout) failed
  Testname: Beispieltest Teil 2 (Datei) failed

Leider durchgefallen :-O
```

- Fiktiver Fehlerfall:
  - Ausgabe auf stdout ist was ganz anderes
  - Die erzeugte Datei ist leer
- Hinweis zur Ausgabe von diff (2ter Test):

  < nicht passender Inhalt der linken Datei
  (hier laut Test beispieltest.out)
  in diesem Fall: leer

  > nicht passender Inhalt der rechten Datei
  (hier laut Angabe im Test beispieltest.exp)
  in diesem Fall die komplette Datei

  --- Trennzeichen zwischen den beiden Dateien
- Hinweis:
  die Zeile, die nach CMD steht kann kopiert und direkt ausgeführt werden um auf der Konsole zu sehen was passiert
  (beachtet dabei, dass der zweite auf dem ersten Test aufbaut und nur die Datei testet, die dort entstanden ist)

Allgemeine Anforderungen

- Vorbereitung auf die Erklärung der Inhalte und der Funktionsweise des Skripts
- Beschäftigung mit eigenständigem Testen und Erstellen von Eingabedaten und den passenden zu erwartenden Ausgaben
- Weitere Beschäftigung mit SVN auf der Konsole und Anlegen der entsprechenden Screenshots und der Beschreibungsdatei
- Dateien, die im Repository zum Servertestzeitpunkt vorhanden sein müssen:
  - worst.sh
  - Testdaten und dazu passende Testbench oder Beschreibungsdatei
  - Unterordner svn mit den Beschreibungen und Screenshots zur SVN-Benutzung Teil 2 auf der Kommandozeile

Nachbesserungsfähige nicht-funktionale Mängel

Nachbesserungsgrund	Abzug Prozentpunkte bei Nichterfüllung
Repository / Ordnerstruktur schlecht gepflegt	5
Interpreterangabe fehlt	5
Uneinheitliche Formatierung oder Sprachwahl für Bezeichner oder Kommentierung	5
Unangemessene Bezeichner (z.B. hinsichtlich ihrer Länge oder Aussagekraft)	5
Mangelhafte Formatierung / schlecht lesbarer Code (z.B. Einrückung, Tabs und Leerzeichen gemischt)	5
Mangelhafte Dokumentation (z.B. Funktionsköpfe / Inline-Kommentierung)	5
Mangelnde Codehygiene (z.B. toter Code)	5
Unnötige Codeverdopplung	10
Verwendung von unstrukturierter Programmierung (z.B. unangemessen viele returns oder exits)	10
Mangelnde Effizienz / umständliche Lösung (z.B. unnötige temporäre Dateien / Variablen)	20
Nicht vorhandene bzw. ungenügende Testfälle	30
Nicht aufgabenkonforme Implementierung	30