

Sie sind hier: [FH Wedel](#) > [Mitarbeiter](#) > [Malte Heins](#) > [Übung: UNIX](#) > [Aufgaben](#) > [Aufgabe 2](#)

## Aufgabe 2

*Shell-Skripte und Pipes, eigenständiges Testen, Umgang mit SVN - Teil 2*

### Aufgabenstellung

In der zweiten Aufgabe wollen wir uns zu einen mit der Verwendung von Filtern und Pipes beschäftigen und zum anderen den weiteren Umgang mit SVN auf der Konsole vertiefen. Zudem wollen wir das eigenständige systematische Testen weiter ausbauen.

Die Aufgabe gliedert sich daher in drei Teilaufgaben.

#### Teil 1: Auswertungstool

Ein fiktives Umfragetool hat als Ausgabe folgendes zeilenbasiertes Format (CSV-Datei).

Jede Zeile besteht aus 3 Feldern, die jeweils durch ein Komma getrennt sind:

- » Die erste Spalte enthält eine **Kennzahl**
- » die zweite eine **Bewertung**.
- » und die dritte eine **Bezeichnung**

Ungültige Zeilen enthalten den Ausdruck `***error***` in beliebiger Groß- und Kleinschreibung (ohne die Anführungszeichen).

Ihr sollt nun ein Programm (in diesem Fall ein kleines Shell-Skript) schreiben, welches die Ausgabe dieses fiktiven Umfragetools weiterverarbeiten kann.

Dafür bekommt euer Programm die Ausgabe des Umfragetools über die Standardeingabe (`stdin`).

Euer Programm soll den Namen (Bezeichnung) des Eintrags mit der schlechtesten (niedrigsten) Bewertung auf `stdout` schreiben. Dabei soll nur die Bezeichnung des Eintrags (kein Komma oder anderes Zeichen) in einer einzigen Zeile ausgegeben werden.

Als erster und einziger Parameter wird eurem Programm ein Dateiname übergeben. In diese Datei sollen alle Zeilen der Eingabe absteigend sortiert nach der Bewertung (2. Feld) und - wenn das nicht eindeutig ist - nach dem ersten Feld (der Kennzahl) ebenfalls absteigend sortiert, geschrieben werden. Der schlechteste Eintrag steht also ganz unten in der sortierten Datei. Ungültige Zeilen, also Zeilen, die den Ausdruck `***error***` (in beliebiger Groß-/Kleinschreibung) enthalten, werden hierbei nicht beachtet und entsprechend auch nicht mit in die Datei geschrieben.

Alle weiteren Parameter dürfen ignoriert werden.

Falls die Ausgabedatei bereits existiert, soll sie überschrieben werden.

Es sind zwei CSV-Dateien (Eingabe und erwartete Ausgabedatei) und ein kleiner Test für diese Aufgabe **vorgegeben**. Diese sollen das Format der Eingabedateien und die geforderten Ausgaben verdeutlichen.

#### Erlaubte Tools

In dieser Aufgabe sind fortgeschrittene Filter und Tools wie etwa `perl`, `sed`, `awk` und `ruby` weder notwendig noch erlaubt.

#### Fehlerbehandlung

Eine Fehlerbehandlung beim Aufruf ist nicht notwendig. Wir gehen einfach davon aus, dass der Benutzer unser Skript richtig verwendet, d.h. dass die Eingabedaten ein gültiges Format besitzen. Den Fall, dass sowohl die Bewertungen als auch die Kennzahlen bei mehreren Einträgen gleich sind, und den Fall, dass ihr keine Schreibrechte für die Datei habt, müsst ihr nicht behandeln.

Es kann passieren, dass ihr Warnungen von den eingesetzten Filtern bekommt, wenn ihr Dateien als Eingabe verwendet, die Sonderzeichen enthalten. Ein Beispiel für eine mögliche Warnung könnte so aussehen:

"Invalid or incomplete multibyte or wide character"

Diese Warnungen sind für die Lösung der Aufgabe nicht relevant und können diesmal ignoriert werden.

#### Umfang der Lösung

Zur Lösung dieser Aufgabe dürfen maximal **sechs Filteraufrufe** und **fünf Pipes** eingesetzt werden. Sollte eure Lösung umfangreicher sein, überprüft sie auf weitere Vereinfachungsmöglichkeiten. Es soll sich zudem prinzipiell um einen **Einzeiler** handeln, selbstverständlich abgesehen von der obligatorischen ersten Zeile in Shell-Skripten (siehe [Aufgabe 1](#)) und auch abgesehen von Kommentaren. Alle Filteraufrufe müssen durch Pipes verbunden sein. Achtet bei eurer Implementierung auf eine **möglichst effiziente Verarbeitung** (z.B. sinnvolle Reihenfolge der Filteraufrufe).

#### Benennung

Nennt euer Skript `"worst.sh"`, damit die automatischen Tests laufen und achtet möglichst darauf, dass es als ausführbare Datei eingereicht ist (siehe Hinweise zu SVN aus [Aufgabe 1](#)).

#### Teil 2: Eigenständiges Testen

Der mitgelieferte **Beispieltest** testet nur einen kleinen Teil der Funktionalität eures Skripts und ist bei weitem nicht vollständig. Es ist also sinnvoll und bei dieser Aufgabe auch explizit gefordert (siehe unten), sich zunächst die Aufgabenstellung und den Beispieltest anzuschauen und anhand dieser eigene Testfälle zu entwickeln, mit denen ihr lokal und vor allem auch reproduzierbar testen könnt. Bedenkt, dass der Testserver euch genaugenommen immer nur Feedback

zu eurem eigenen Test und seiner Abdeckung liefert. Schreibt also als erstes eigene Testdateien für die Fälle, die wir bzw. ihr bisher nicht abgedeckt habt, wenn ihr einen Fehler in eurem Skript bzw. einen fehlgeschlagenen Servertest bemerkt.

Wir verwenden in der Unix-Übung das Testsystem **Arnold**. Ein Aufruf von Arnold mit dem zu testenden Skript `worst.sh` und der Testdatei `testbench.arnold` sieht beispielsweise wie folgt aus (vorher Arnold mit `chmod +x` ausführbar machen):

```
./arnold -c ./worst.sh testbench.arnold
```

Das Flag `-c` dient bei Arnold dem Einschalten des continuous-Mode, d.h. dass immer alle Tests durchlaufen und nicht beim ersten fehlgeschlagenen Test abgebrochen wird. Aus Komfortgründen empfiehlt es sich immer mit diesem Flag zu arbeiten.

Für die Ausführung von Arnold muss die expect-Shell (ein **tcl-Interpreter**) installiert sein. In den Rechenzentren und in der von uns bereitgestellten **VM** ist dies bereits der Fall, auf einem eigenen Debian-Linux (z.B. Ubuntu) kann man dazu folgenden Befehl nutzen:

```
sudo apt-get install tcl tk expect
```

Ob die expect-Shell korrekt installiert ist, könnt ihr überprüfen indem ihr sie einfach mal mit expect startet. Wenn dies funktioniert hat, kann Arnold wie in der Aufgabe beschrieben gestartet werden können.

In der Abnahme erwarten wir, dass ihr **eigene Testdateien** im SVN-Repository nachweisen und erklären könnt. Dies bedeutet zum einen Eingabedateien und die passende erwartete Ausgabe um das Programm zu testen (analog zu `beispieltest.in` und `beispieltest.exp`) und zum anderen eine Testbench oder entsprechende manuelle Beschreibung der Tests, die diese Dateien verwenden.

Das Format (arnold, nightmare oder eine manuelle Beschreibung der Eingaben und erwarteten Ausgaben) ist dabei euch überlassen.

### Hinweis zur Arnold Testbench

Was einige Dinge in der Arnold-Testbench angeht, müssen wir der Vorlesung etwas vorgreifen, die hier vorgestellten Konzepte lassen sich allerdings auch ohne weiteres Hintergrundwissen einsetzen.

Im Feld expect der Testbench, wird die `stdout`-Ausgabe des bei command definierten Aufrufs getestet. Um sicherzustellen, dass sich dort lediglich die gewünschte Ausgabe befindet, wird der ihr Ausgabe ein `^` (steht für den Zeilenanfang) vorangestellt und ein `$` (steht für das Zeilenende) hinten angestellt.

Für den Fall, dass man Dateien auf Gleichheit testen will, kann das Tool `diff` verwendet werden, welches Unterschiede in Dateien aufzeigt. Wenn 2 Dateien identisch sind, so gibt `diff` einfach nichts aus. Dies wird dann dementsprechend mit dem Ausdruck `^$` überprüft, der besagt, dass zwischen Zeilenanfang und Zeilenende nichts stehen darf.

### Teil 3: SVN-Benutzung Teil 2

Dieses Mal solltet ihr euch insbesondere mit den weiteren Befehlen und Möglichkeiten von SVN in der Konsole (also ohne grafische Oberfläche) vertraut machen.

Dazu zählen die folgenden Aktionen:

- » Anzeigen diverser Statusinformationen über das SVN-Repository im Allgemeinen und für einzelne verwaltete Dateien.
- » Lösen eines Konfliktes. Dafür müsst ihr euch natürlich vorher Gedanken gemacht haben, was ein Konflikt überhaupt ist und wie er entstehen kann.
- » Zurücksetzen auf eine frühere Arbeitsversion und diese im Anschluss als aktuelle Arbeitsversion definieren. Dies ist sinnvoll, wenn nach längerer Entwicklungsarbeit festgestellt wurde, dass der falsche Weg eingeschlagen wurde und mit einer früheren Version weitergearbeitet werden soll.

Wie in der letzten Aufgabe sollte ihr diese Schritte entsprechend wieder mit Screenshots und einer textuellen Beschreibung belegen. Legt bei der textuellen Beschreibung euer Augenmerk insbesondere auf Konflikte und deren Behebung. Fragen, die dabei beantwortet werden sollen, sind:

- » Was ist ein SVN-Konflikt und wie kann er entstehen?
- » Kann ein Konflikt nur bei mehreren Benutzern des gleichen Repositories entstehen?
- » Wie geht man schrittweise vor um einen Konflikt zu beheben?

Baumkonflikte könnt ihr bei eurer Betrachtung außen vor lassen, da diese deutlich seltener entstehen als Konflikte, die einzelne Dateien betreffen.

Legt die Screenshots und die ergänzende Textdatei wieder in einen Unterordner mit dem Namen `svn` diesmal unter `ueb02` ab und achtet darauf den Ordner und alle Dateien auch mit ins SVN einzuchecken.

Die Screenshots und die textuelle Beschreibung müssen auch dieses Mal wieder zum Servertestzeitpunkt im Repository vorhanden sein.

Dieser Aufgabenteil geht mit einem Drittel in die Bewertung der Aufgabe mit ein.

### Hinweis zum Servertest

Bitte beachtet, dass auch diesmal der Servertest nur eure Implementierung (`worst.sh`) testet, aber weder das Vorhandensein der Screenshots und der passenden textuellen Beschreibung noch das Vorhandensein eurer eigenen Tests.

Das Gesamtergebnis der Aufgabe wird wie bei Aufgabe 1 im Nachgang der Abnahmen ermittelt und entsprechend auf dem Server eingetragen.

### Nicht-funktionale Mängel

Nachbesserungsfähige nicht-funktionale Mängel:

Nachbesserungsgrund	Abzug Prozentpunkte bei Nichterfüllung
Repository / Ordnerstruktur schlecht gepflegt	5
Interpreterangabe fehlt	5

Nachbesserungsgrund	Abzug Prozentpunkte bei Nichterfüllung
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

## Bewertung


Der **automatische Servertest** wird bei dieser Aufgabe mit **zwei Dritteln** in die Gesamtwertung eingehen, das letzte Drittel der Punkte ergibt sich aus den Screenshots und der textuellen Beschreibung zum Umgang mit SVN.

Das eigenständige systematische Testen wird ab dieser Aufgabe als nachbesserungsfähiger Mangel betrachtet (siehe nicht-funktionale Mängel).

## Allgemeine Anforderungen

Bitte beachtet, dass die allgemeinen Anforderungen (Dokumentation, Hinweise zu Skripten, etc.) aus **Aufgabe 1** auch in allen weiteren Aufgaben gelten.

## Folien

Die **Folien**  zur Aufgabenvorstellung (die ja in diesem Semester leider nicht stattfinden kann), geben einen groben Überblick über die Aufgabenstellung und sollen dem besseren Einfinden in die Aufgabenstellung dienen. Sie sind nicht als weitere Spezifikation gedacht und auch nicht so zu verstehen - im Zweifelsfall gelten immer die Angaben, so wie sie hier in der Aufgabenstellung stehen.

Bitte beachtet, dass die Folien diesmal auch einige nützliche Zusatzinformationen liefern, die in dieser Form nicht in der Aufgabenstellung enthalten sind. Insbesondere die Erklärungen zu `arnold` und den Testbenches, die wir sonst immer im Hörsaal gemacht haben.

## Archiv

Das **Archiv** enthält den mitgelieferten Beispielttest (inkl. Beispieldateien für Ein- und Ausgabe) und das Tool `arnold` selbst. Ihr könnt das Archiv auf der Konsole mit dem Befehl

```
tar -xf aufgabe2.tgz
```

entpacken.

*letzte Änderung: 19.05.2020 14:41*