Übung 4: Schreiben eines Shell-Skripts

Aufgabenteil 1

Ziel der Übung

Anhand eines einfachen Linux-Skripts sollen die Grundlagen der Shell-Programmierung vermittelt werden. Das für die Bash-Shell zu erstellende Shell-Skript verdeutlicht Basiskonzepte wie beispielsweise das Pipe-Konzept. Zum Erlernen der Syntax (z.B. Kontrollstrukturen) sei auf Internetquellen verwiesen.[1]

Einführung

Eine Shell ist in Linux ein Kommandointerpreter, der in einem Terminal ausgeführt wird. Die Shell ist auf den ersten Blick vergleichbar mit der Kommandozeile aus der Microsoft-Welt. Neben dem Ausführen einfacher, einzelner Befehle ermöglicht eine Shell jedoch unter Linux äußerst leistungsfähige Programme (sog. Shell-Skripte) zu programmieren, die in der Praxis zum Beispiel zur Administration von Servern eingesetzt werden. So werden beispielsweise automatische Backups von Dateien mit Shell-Skripten realisiert.

Wie wird ein Shell-Skript erstellt?

Um den Quelltext zu erstellen, ist ein einfacher Texteditor wie *vi*, *nano* oder *emacs* ausreichend. Kurze Bedienungsanleitungen für vi sind auf mehreren Webseiten zu finden.[2] Nachfolgend ist ein sehr einfaches Skript zu sehen.

```
#!/bin/bash
echo "Servus Linux!"
```

Ganz am Anfang eines Skripts wird die Shell spezifiziert, mit der die weiteren Anweisungen interpretiert werden soll. In der zweiten Zeile wird dann ein Text ausgegeben. Hat man das Skript erstellt und in einer Datei abgespeichert, so muss man diese Datei mit dem Befehl `chmod` als ausführbar (Modus: +x) markieren. Das Skript kann anschließend mit ./<Dateiname> ausgeführt werden.

Exkurs: Shell, Bash oder doch Bourne-Shell?

Shell: Unter einer Shell versteht man die traditionelle Benutzerschnittstelle unter Unix. Der Begriff Shell ist allerdings als Oberbegriff für verschiedene Kommandozeileninterpreter zu verstehen, sodass viele unterschiedliche „Shells" existieren. Diese unterscheiden sich meist in den verfügbaren Befehlen bzw. in der Syntax dieser. Nachfolgend ist eine kurze Auflistung über die gängigsten Shells gegeben:

Bourne-Shell(sh): Die Bourne-Shell ist Vorfahre vieler heutiger Shells. Noch heute finden sich in vielen Unix-Distributionen Nachfolger der Bourne-Shell.

*Bash (**B**ourne-**a**gain-**sh**ell):* Die Bash ist die Standard-Shell für viele Linux-Distributionen. Für weiterführende Informationen bitte die Man-Page aufrufen (`man bash`).

[1] Ein möglicher Suchbegriff bei Google wäre „Linux Shell Programmierung".
[2] Ein möglicher Suchbegriff bei Google wäre „vi editor übersicht".

Weitere Shells: Korn-Shell(ksh), C-Shell(csh), Thompson-Shell(osh)

Das Pipe-Konzept

Unter einer *Pipe* versteht man unter Linux (oder Unix) die Möglichkeit, die Standardausgaben eines Programms in die Standardeingabe eines anderen Programms zu schreiben. Die Ausgaben, die ein Programm liefert, können also als Eingabe an ein anderes übergeben werden, z.B.:

```
ls | wc -w
```

In vorhergehendem Beispiel wird die Ausgabe von `ls` (zur Erinnerung: ls listet den Inhalt eines Verzeichnisses auf) durch Nutzung des Pipe-Zeichens „|“ nach `wc` (Word-Count) umgeleitet. Statt einer Liste von Dateien und Unterverzeichnissen erhält man dadurch die Anzahl der Wörter, die der ls-Befehl ausgibt.

Argumente an ein Skript übergeben

Für gewöhnlich möchte man ein Skript durch Benutzereingaben parametrisieren. Argumente, die an ein Shell-Skript übergeben werden, werden durch Variablen repräsentiert. Variablen beginnen immer mit dem Zeichen „$“. Durch die Übergabe von Parametern sind einige Variablen schon vorbelegt.

Variablenname	**Bedeutung**
$#	Anzahl der übergebenen Argumente
$0	Name des ausgeführten Skripts
$1	Wert des 1. Parameters
$2	Wert des 2. Parameters
…	

Beim Aufruf eines Skripts werden die Parameter einfach durch Leerzeichen getrennt.

Befehle zu Kontrollstrukturen und Bedingungen[3]

- **if / then / else**

test	
Beschreibung	Verzweigungsmöglichkeit anhand einer Bedingung.
Optionen	
Syntax / Beispiele	**if** [<Bedingung>]**; then** <Code> **else** <Code> **fi** Beispiel siehe nachfolgende Übung.

[3] Weitere ablaufsteuernden Befehle sowie weitere Optionen zu den vorgestellten Befehlen können z.B. der folgenden Website entnommen werden: http://de.wikibooks.org/wiki/Linux-Kompendium:_Shellprogrammierung (abgerufen am 15.03.2014 15:15 Uhr)

- **test**

test	
Beschreibung	Überprüft Dateien/Verzeichnisse und vergleicht Werte.
Optionen	Expression **-a** Expression: und -Verknüpfung String1 = String2: Vergleich von Strings **-e** FILE: Überprüft ob Datei existiert
Syntax / Beispiele	test <Expression> test –e test.sh && echo „Datei vorhanden“ || echo „Datei nicht vorhanden“ [␣–e test.sh␣] && echo „Datei vorhanden“ || echo „Datei nicht vorhanden“

- **for - Schleife**

test	
Beschreibung	Kopfgesteuerte Schleife
Optionen	
Syntax / Beispiele	Syntax **for** <Bedingung> **do** <Code> **done** Beispiel 1: **for** ((i = 0 ; i < 10 ; i++)) **do** echo $i **done** Beispiel 2: **for** i **in** {0..10} **do** echo $i **done**

- **while - Schleife**

test	
Beschreibung	Kopfgesteuerte Schleife
Optionen	
Syntax / Beispiele	Syntax **while** <Bedingung> **do** <Code> **done** Beispiel **while** (i < 10) **do** echo $i **done**

Hinweis: Wie im letzten Beispiel ersichtlich wird, kann man statt des `test`-Befehls auch eckige Klammern („[“ und „]“) verwendet werden. Bei dieser Verwendung **muss** jedoch auf die Leerzeichen vor bzw. nach der eckigen Klammer geachtet werden.

Übung: Überprüfen Sie nun, ob in Ihrem Home-Verzeichnis ein Ordner mit dem Namen „Test“ angelegt ist! (Hinweis: suchen Sie in der Man-Page des test-Befehls nach einem geeigneten Parameter hierfür)

Der gerade gelernte `test`-Befehl wird häufig in Verzweigungen in sog. if-Anweisungen benötigt. Der Aufbau der if-Anweisungen ist der Syntax aus Java sehr ähnlich.

Folgendes Skript überprüft, ob der erste mit dem zweiten Parameter identisch ist. Es soll die Funktionsweise von if-Anweisungen demonstrieren.

```
#!/bin/bash

if [ "$1" = "$2" ]; then
        echo "Die Parameter sind gleich"
else
        echo "Die Parameter unterscheiden sich"
fi
```

Übung: Legen Sie eine Datei mit obigem Quellcode als Inhalt an und bringen Sie es zum Ablauf. (*Hinweis*: Evtl. ist hier der `touch`-Befehl bzw. der bereits bekannte Befehl `vi` hilfreich)

Nachdem Sie die Datei angelegt und abgespeichert haben, müssen Sie diese noch ausführbar machen (betrachten Sie hierzu ggf. das Linux-Berechtigungskonzept aus Übung 01 erneut). Anschließend wechseln Sie in das Verzeichnis, in der Sie die Datei abgespeichert haben und führen diese mit dem Befehl `./<Dateiname> <Parameter1> <Parameter2>` (z.B. `./compare Linux Linux`) aus.

Vorsicht: Achten Sie bei Verwendung von Copy&Paste aus dem Script in die Linux-Kommandozeile darauf, dass damit auch nicht sichtbare Sonderzeichen übertragen werden können, die der Bash-Interpreter nicht interpretieren kann. Also lieber abtippen!

Aufgabenstellung:

Arbeiten Sie sich in die Grundlagen der Shell-Programmierung ein. Als Einstiegspunkt für die Einarbeitung könnte beispielsweise die Website mit der folgenden URL helfen: http://de.wikibooks.org/wiki/LinuxKompendium[4] (Suche nach „Shellprogrammierung“ und „Programmablaufkontrolle“). Versuchen Sie nun an folgendes Skript zu erstellen und zum Ablauf zu bringen.

Das zu erstellende Skript mit dem Dateinamen „mvprot“ soll im aktuell gewählten Verzeichnis alle Dateien mit der angegebenen Endung in ein neu erstelltes Unterverzeichnis verschieben und die Anzahl der verschobenen Dateien sowie deren Namen – falls gewünscht – in eine Protokolldatei im Quellverzeichnis schreiben. Das Skript besitzt also die drei Parameter *Dateiendung, Name des Unterverzeichnisses* und ein *Flag* für die Auswahl, ob ein Protokoll über den Vorgang erstellt wird. Zum Entwickeln und Testen Ihrer Lösung legen Sie sich bitte ein neues Verzeichnis mit einigen Testdateien an. Viel Vergnügen! ☺

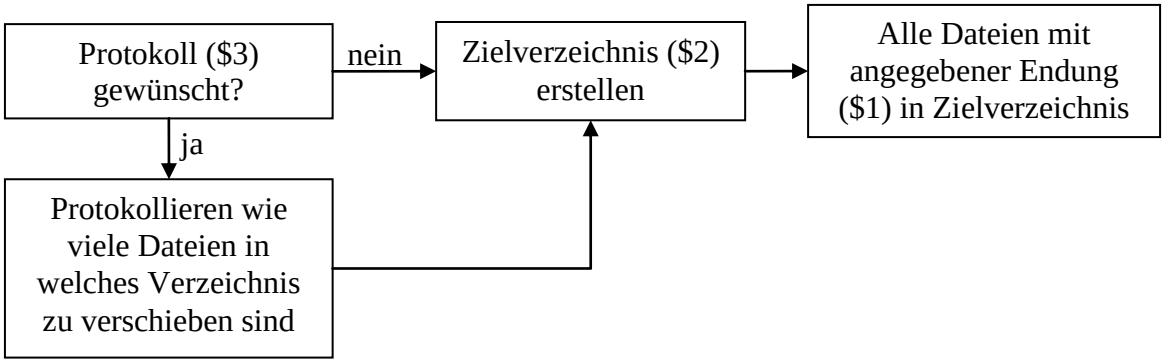

Abbildung 1 - Schematischer Ablauf der Aufgabenstellung

Noch ein Hinweis:

Mit dem Kommando `AnzahlDateien=\`ls *.txt | wc -w\`` weisen Sie der Variable *AnzahlDateien* die Anzahl der Dateien mit der Endung „.txt“, die sich im aktuellen Verzeichnis befinden, zu. Durch die Anwendung der sog. Backquotes (Vorsicht: nicht die einfachen Hochkommata) wird eine Kommadosubstituton durchgeführt. Das bedeutet, dass die in Backquotes eingeschlossenen Kommandos zusammenhängend ausgeführt werden und die Ausgabe der gesamten Befehlsfolge als Ergebnis in die Variable geschrieben wird. Alternativ dazu kann man das Kommando in der bash wie folgt notieren: `AnzahlDateien=$(ls *.txt | wc -w)`

Mit diesem Mechanismus könnten Sie z.B. die Anzahl der verschobenen Dateien zählen. Aber es gibt auch andere Möglichkeiten.

[4] Letzter Zugriff am 15.03.2014, 16:00 Uhr)

Aufgabenteil 2

Erweiterte Aufgabenstellung:

Versuchen Sie nun, Ihr Programm so zu erweitern, dass eine Prüfung der übergebenen Parameter stattfindet. Zudem soll überprüft werden, ob das zu erstellende Verzeichnis bereits existiert. In diesem Fall soll dem Anwender eine entsprechende Fehlermeldung ausgegeben werden (*Hinweis*: Vielleicht hilft Ihnen der Befehl `test` weiter). Ferner ist gefordert, eine Hilfefunktion zu implementieren, die eine kurze Beschreibung des Programms und der möglichen Parameter liefert.

Erweiterte Informationen zur Shell-Programmierung

Manchmal möchte man wissen, ob ein Skript erfolgreich abgeschlossen, oder ob es vorher unerwartet beendet wurde. Hierzu liefert jedes Skript automatisch einen sog. *Return-Code*. Ebenso ist es manchmal in einem zu erstellenden Skript hilfreich ist, die Anzahl der übergebenen Parameter zu ermitteln. Auch dies lässt sich abfragen.

Informieren Sie sich zuerst über die Funktionsweise folgender „Parameter“ der Shell-Programmierung und prüfen Sie anschließend, ob Sie diese evtl. in Ihrem erweiterten Skript verwenden können.

- $#
- $?
- $0

Ein weiterer hilfreicher Befehl lautet `bash -x <Programmname>`. Informieren Sie sich, was dieser macht und testen Sie ihn anhand Ihres zuvor erstellten Skripts.

Weitere hilfreiche Linux-Befehle

Neben den bisher vorgestellten Linux-Befehlen existieren noch eine ganze Reihe weiterer Befehle, die einem Administrator die Arbeit erleichtern. Nachfolgend sind einige, häufig benötigte Kommandos aufgelistet. Informieren Sie sich kurz über das jeweilige Kommando und beschreiben Sie in einem Satz, was dieses macht. Testen Sie es anschließend an einem Beispiel Ihrer Wahl.

- grep
- find
- sort
- awk
- sed
- read
  (Bauen Sie diesen Befehl in Ihr erweitertes Skript sinnvoll ein)
- cut