Systemverwaltung mittels Skripts

Inhaltsverzeichnis:

1. Skriptsprachen

Betriebssystem-Abhängige Skriptsprachen:
Unix / Linux Shell Scripts, Windows Scripts

Betriebssystem-unabhängige Skriptsprachen:
Perl

Scriptsprache wird mit einem Texteditor erstellt und in einer Datei abgespeichert. Ein erstelltes Script muss nicht kompiliert werden. Es enthält viele Shellbefehle, die so zusammengefasst werden.

2. Zur Erstellung von Skripts

Am Anfang des Skripts soll eine Kommentarzeile stehen, die Aussagen über die Shell angibt, für die das Skript geschrieben wurde.

Für die Bourne Shell: `#!/bin/sh`
für die bash-Shell: `#!/bin/bash`
Für die Korn-Shell: `#!/bin/ksh`

Wenn eine Skript-Datei fertig geschrieben wurde muss diese mit dem Befehl `chmod` ausführbar gemacht werden.

Falls von jedem Verzeichnis aus das Script aufgerufen werden soll wird das Verzeichnis, das das Script enthält, am besten in der PATH-Variable eingetragen.

3. PATH

Falls ein Script von der Kommandozeile aufgerufen werden soll (d.h. entsprechende Datei ausführen) muss diese in einem im PATH definierten Verzeichnis liegen. Ansonsten kann man die PATH-Variable anpassen und das neue Verzeichnis einfügen. Die einzelnen Verzeichnisse werden durch einen : getrennt.

```
PATH=/bin:/usr/bin:/home/name/bin
```

4. Befehlsbeispiel

```
cp –i –r ./xy
```

`cp` Befehlsname (hier: Unix-Befehl für copy / Datei kopieren)
`-i –r` Kommandooptionen
`./xy` Kommandoparameter

5. Variablen

Eine Variable speichert zugewiesener Text. PATH z.B. eine Variable. Falls aus der Shell ausgeloggt wird, existiert die Variable nicht mehr.

Variable x mit Inhalt Xaver erstellen:	`x=Xaver`
Wert der Variable x abrufen:	`$x`
Wert der Variable auf Screen ausgeben:	`echo $x`
Variable x löschen:	`unset x`

Vordefinierte Variablen:

PATH:	Suchpfad, wo von der Shell ausführbare Scripts sind (diese Scripts sind von jedem Verzeichnis aus aufrufbar)
HOME:	Pfad des benutzereigenen Verzeichnisses
PWD:	Pfad des gerade aktuellen Verzeichnisses
PS1:	setzt den Prompt am Anfang der Kommandozeile fest
PS2:	Text, den Shell ausgibt, wenn zusätzliche _Eingabedaten von dem Benutzter erwartet werden
LOGNAME:	eingeloggter Username
OSTYPE:	Welches Betriebssystem vorhanden ist

6. Parameter

```
checkfile datei xy blabla
```

Die Parameter sind in durchnummerierten Variablen gespeichert

$0 `./checkfile`
$1 `datei`
$2 `xy`
$3 `blabla`
usw...

7. Spezialargumente Variablen

`$#`	Anzahl der Skriptargumente
`$*`	Enthält als String alle Parameter
`$$`	Aktueller PID (Process ID) des ausgeführten Skripts
`$!`	PID des letzten in den Hintergrund versetzten Prozesses
`$@`	wie `$#`, liefert aber die Sktiptargumente je für sich in Anführungszeichen
`$-`	Zeigt wie **set** aktuelle Shellvariableninhalt
`$?`	Zeigt Ausführungsstatus (exit status) des letzten Befehls (`0` für keine Fehler, andere Werte für Fehler)

8. Quoting

In der Kommandozeile wird jedes durch einen Leerschlag getrennte Wort für sich bearbeitet. Falls mehrere Worte zusammen gehören, kann man dies durch Quotes bezeichnen, dass diese als „ein Wort“ behandelt werden.

Doppelte Anführungsstriche: ""
für Zeichenketten, aber nicht wirksam für die Zeichen `$`, `´`, `\`
Bsp: `„Hallo echo $cd“`

Einfache Anführungsstriche: ´´
gilt auch für die Zeichen `$`, `´`, `\`

back quotes: ``
für die Zuweisung von Rückgabewerten von Befehlen an Variablen
Bsp: `xy=`date``

**Backslash: **
hindert Shell am Interpretieren folgender Zeichen: `&`, `*`, `=`, `^`, `$`, `´`, `?`
z.B. `echo es kostet 2 \$ für das reinigen`
(`$` wird ohne `\` ausgeschrieben)

9. Metazeichen

Metazeichen	Verwendung	Beispiel
*	Kein, ein oder mehrere Zeichen	`ls *.c`
?	ein beliebiges Zeichen	`print main.?`
[xy]	Ein x oder ein y	`ls main.[hc]`
[a-z]	ein kleiner Buchstabe (zwischen a und z)	`ls [a-z]module.c`
[!0-9]	Zeichen, das keine Zahl ist	`print mod[!0-9]`
#	leitet Kommentarzeile ein	`# Kommentar`
> <	Ein- / Ausgabeumleitung	`ls * > list.txt`
|	leitet Ausgabe des linksstehenden Befehls an rechtsstehenden Befehl	`ls | sort`

10. Ein- / Ausgabe

Mittels der Zeichen < und > kann man die Eingabe bzw. Ausgabe umleiten. Das klassische Beispiel hierfür ist sicher `ls * > directory.txt`.
Dieser Befehl schreibt die Ausgabe des `ls`-Befehls in eine Datei namens `directory.txt`.

Eingabe:
wie gewohnt mit dem < - Zeichen (alternativ kann auch `0<` geschrieben werden)

Ausgabe:

`>`	Ausgabekanal für Text und Fehlermeldung
`1>`	Ausgabekanal ohne Fehlermeldung
`2>`	Ausgabekanal für die Fehlermeldung (und nur für die Fehlermeldung)

Bsp: `sort 0< studi 1> studi.txt 2> studi.err`

11. Gruppierung von Befehlen

`k1;k2`	Zuerst wird `k1`, dann `k2` ausgeführt
`k&`	Befehl wird im Hintergrund ausgeführt
`k1&&k2`	`k2` wird ausgeführt, falls `k1` erfolgreich war (d.h. Rückgabewert von `k1`=0)
`k1||k2`	`k2` wird nur dann ausgeführt, falls `k1` fehlerhaft war
`(k)`	`k` wird in Subshell ausgeführt

12. Test

Mit dem Befehl `test` können die einzelnen Dateistaten abgefragt werden.

`test –d verz`	ist verz ein Verzeichnis?
`test –r`	ist Datei lesbar?
`-w`	ist Datei schreibbar?
`-x`	ist Datei ausführbar?
`-s`	ist Datei nicht leer?
`-f`	ist es eine reguläre, existierende Datei?

Antworten:
`0` = true
`1` = false

Alternativ kann der test-Befehl auch durch `[` und `]` ausgeführt werden. So sind diese Befehle identisch:

`test –d hans` und `[ -d hans ]`

Dabei ist zu beachten, dass nach dem `[` und vor dem `]` zwingend einen Leerschlag stehen muss.