ProSeminar

Shell-Programmierung für Fortgeschrittene

Stefan Harinko

Technische Universität München

13. März 2007

Shellskript

Ein Shellskript ist eine Textdatei, die von einer Shell ausgeführt werden kann. In der Datei `shellskript.sh` könnte stehen:

```
#!/bin/bash

echo 'Hallo_ProSeminar Unix-Tools !'

exit 0
```

Möglichkeiten die Datei `shellskript.sh` auszuführen:

- Wenn die Datei mit `chmod +x` ausführbar gemacht wurde, kann das Skript mit `./shellskript.sh` gestartet werden.
- Ansonsten benutzt man "`bash shellskript.sh`"

Beispielaufruf

```
$ ./shellskript.sh
Hallo ProSeminar Unix-Tools !
$ bash shellskript.sh
Hallo ProSeminar Unix-Tools !
$
```

einfache Ersetzungen durch Variablensubstitutionen:

- **`${Variable#pattern}`** entfernt das erste Pattern vom Anfang des Variableninhalts
- **`${Variable##pattern}`** entfernt das längste Pattern vom Anfang des Variableninhalts

```
$ Variable=/home/user/datei
$ echo ${Variable#/*/}
user/datei
echo ${Variable##/*/}
datei
```

- **`${Variable%pattern}`** das Pattern wird vom Ende her entfernt
- **`${Variable%%pattern}`** das längste Pattern wird entfernt
- **`${Variable/pattern/string}`** das erste Pattern wird durch string ersetzt
- **`${Variable//pattern/string}`** alle Muster werden durch string ersetzt
- weitere Möglichkeiten findet man in der Manpage zu bash unter "`Parameter Expansion`"

Ausgabeumlenkung von Standardausgabe und Standardfehler

```
2>&1
kopiert Standardfehler auf Standardausgabe
```

live Beispiel ...

- Exit Status und Befehlssequenzen
- Ausdrücke
  - arithmetische Ausdrücke
  - Bedingungsausdrücke
  - Verknüpfung von Ausdrücken
- Kontrollstrukturen
  - if
  - for
  - while und until
  - case
- Funktionen
- Ausblick

- Der Exit-Status eines Befehls ist 0, wenn er erfolgreich war und ungleich 0 bei einem Fehler.
- Befehlssequenzen oder Befehlslisten werden mit speziellen Operatoren getrennt und terminiert
  - `;` `&` `&&` oder `||` trennen Listen
  - `;` `&` oder `<newlines>` terminieren Listen

Beispiel

```
$ befehl1 || befehl2
# befehl2 wird nur gestartet,
# wenn befehl1 fehlschlug
```

- Wenn Befehl2 ausgeführt wird, war der Exit-Status von Befehl1 ungleich 0.

- **`(( expression ))`** wird als arithmetischer Ausdruck ausgewertet, expression hat eine C-ähnliche Syntax
- einige mögliche Operatoren sind:
  `+ - * / % ++ -- == != > < >= <= && ||`

Beispiel

```
(( 5>10 || 7>3 ))
```

- **`[[ expression ]]`** ist ein Ausdruck, der eine Bedingung überprüft
- kann Strings, Variablen, Dateiattribute und auch arithmetische Ausdrücke testen
- wichtige Operatoren:
  - **`string1 == string2`**
  - **`string1 != string2`**
  - > und < testen immer lexikographisch größer oder kleiner
  - **`-n string`** ist wahr wenn die Stringlänge ungleich Null ist.
  - **`-a Datei`** ist wahr wenn Datei existiert
  - **`-d Verzeichnis`** ist wahr wenn Verzeichnis existiert

- **`( expression )`** gruppiert einen Ausdruck um die Auswertungsreihenfolge zu verändern
- **`! expression`** ist genau dann wahr wenn expression falsch ist
- `&&` und `||` funktionieren auch zwischen zwei Ausdrücken

Beispiel

```
  ! (( 3>10 )) || [[ 3 -gt 1 ]]   # ist true
! ( (( 3>10 )) || [[ 3 -gt 1 ]] ) # ist false
```

Hinweis: **`liste`** ist im Folgenden immer eine Befehlssequenz und oder eine Reihe von Ausdrücken

if Syntax

```
if liste
 then
  liste
 [ elif liste; then liste; ] ...
 [ else liste; ]
fi
```

Beispiel

```
if (( "$#" == 0 ))
then
 echo keine Parameter angegeben
 exit 1
fi
```

for Syntax1

```
for name [ in word ] ; do liste ; done
```

- word wird erweitert
- name wird nacheinander auf ein Element aus word gesetzt

Beispiel

```
for datei in skripts/*.sh
 do
  ./$datei
done
```

for Syntax2

```
for (( expr1; expr2; expr3 )); do liste; done
```

- expr1 wird ausgewertet
- expr2 solange bis sie 0 ergibt
- wenn expr2 != 0 wird liste ausgeführt
- dann expr3

Beispiel

```
for ((i=0; i<5; i++));do echo $i; done
```

gibt die Zahlen 0 bis 4 aus

Syntax

```
while liste; do do-Befehlsliste; done
until liste; do do-Befehlsliste; done
```

- while führt do-Befehlsliste solange aus bis liste nicht erfolgreich beendet
- until führt do-Befehlsliste solange aus bis liste erfolgreich läuft

Beispiel

```
while true; do echo true true; sleep 1;done
```

case syntax

```
case word in [ [(] pattern [ | pattern ] ... ) l
```

Beispiel

```
case "$1" in
start)
echo STARTE DAEMON
    /sbin/daemon
;;
    stop)
echo STOPPE DAEMON
killall daemon
;;
esac
```

Syntax

```
[ function ] name () { liste; }
```

Beispiel

```
#!/bin/bash
function name () {
echo "$# Argumente: $*"
}
name "mit zwei" Parametern
echo Ende
exit
```

Ausgabe

2 Argumente: mit zwei Parametern Ende

- lokale Variablen
- Arrays
- Signale, die mit **`trap`** abgefangen werden können
- Debugging mit **`bash -n`** und **`bash -x`**
- named pipes
- Bash Version 3 kann reguläre Ausdrücke in **`[[ ]]`**

und jetzt viel Spaß mit der **Shell**
einem

hi

Quellen

- man bash
- http://www.tldp.org/LDP/abs/html/