Shell Programmierung Unix
Shell Scripts Unix

Mag. Thomas Griesmayer

Allgemeines

- Ein „shell script“ ist eine Textdatei, welche eine Liste von Befehlen (Standard Unix Befehle) beinhaltet.
- Diese Datei kann durch Aufrufen ausgeführt werden.
- „Shell scripts“ werden interpretiert und nicht kompiliert.
- Bei der Ausführung wird eine „child shell“ gestartet und das Programm wird Zeile für Zeile durchgelesen und ausgeführt.
- Nach Ausführung des letzten Kommandos terminiert die „child shell“.

Vorgangsweise

- Erstellen eines neuen Textfiles - Editor „vi“.
- Entscheiden, welche shell man verwenden will - bash.
- Kommandos hinzufügen und die Datei sichern.
- Berechtigungen zum Lesen und Ausführen setzen.
- „shell script“ aufrufen - Prompt.

```
$ vi anz_files.sh
#!/bin/bash

echo "Number of Files"
ls | wc -l
$ chmod 700 anz_files.sh
$ anz_files.sh
Number of Files
      1
```

Kommentar

- Kommentare beginnen mit dem Zeichen „#“ und enden am Zeilenende.

```
$ vi comment.sh
#!/bin/bash
# FILENAME: anz_files.sh
# AUTHOR:   Thomas Griesmayer
# PURPOSE:  Current Directory, number of files
# HISTORY:  9.3. Created, 10.3. displays current path
echo Number of files
pwd
ls | wc -l
$ chmod 700 comment.sh
$ comment.sh
Number of files
/home/thomas/uebD
      2
```

Parameter (1)

- An Unix „shell scripts“ kann eine Liste von Parametern übergeben werden, die man im Programm verwenden kann.

Parameter	Beschreibung
`$0`	Name des Shell Programms
`$1 bis $9`	Die ersten neun Parameter
`$#`	Die Anzahl an Parametern
`$@`	Alle Parameter

Parameter (2)

```
$ vi test_param.sh
#!/bin/bash

echo "Shell script:" $0
echo "Number of parameters:" $#
echo "1 parameter:" $1
echo "2 parameter:" $2
echo "All parameters:" $@
$ chmod 700 test_param.sh
$ test_param.sh *
Shell script: ./test_param.sh
Number of parameters: 3
1 parameter: anz_files.sh
2 parameter: comment.sh
All parameters: anz_files.sh comment.sh test_param.sh
```

shift (1)

- Der Befehl „shift“ dient zum Verschieben der Parameter nach links.
- Die Variable $1 enthält den Inhalt der Variable $2, $2 den der Variable $3, ... und $9 den Wert des zehnten Parameters.
- Die Syntax lautet:

```
shift
```

shift (2)

```
$ vi test_shift.sh
#!/bin/bash

echo "Rest:" $# "Parameters:" $*
shift
echo "Rest:" $# "Parameters:" $*
shift
echo "Rest:" $# "Parameters:" $*
$ chmod 700 test_shift.sh
$ test_shift.sh hallo 1 2 3
Rest: 4 Parameters: hallo 1 2 3
Rest: 3 Parameters: 1 2 3
Rest: 2 Parameters: 2 3
```

Process ID

- Die aktuelle Process ID kann man über die Variable „$$“ erfahren.

```
$ vi get_pid.sh
#!/bin/bash

echo "Actual PID:" $$
$ chmod 700 get_pid.sh
$ get_pid.sh
Actual PID: 2373
$ get_pid.sh hallo 1 2 3
Actual PID: 2374
```

exit status

- Unter Unix liefert jeder Prozess (Befehl) einen „exit status“.
- Dieser kann über die Variable $? abgefragt werden.

```
$ vi check_user.sh
#!/bin/bash

grep $1 /etc/passwd
echo $?
$ chmod 700 check_user.sh
$ check_user.sh thomas
thomas:x:500:100:Thomas:/home/thomas:/bin/bash
0
$ check_user.sh mozart
1
```

&& bzw. ||

- Die Operatoren „&&“ und „||“ sind spezielle Konstrukte.
- Die Syntax lautet:

```
command1 && command2
command1 || command2
```

- Der Operator „&&“ bedeutet, dass nur das zweite Kommando ausgeführt wird, wenn das erste einen „exit status“ von 0 zurückgibt.
- Der Operator „||“ bedeutet, dass nur das zweite Kommando ausgeführt wird, wenn das erste einen “exit status” von ungleich 0 zurückgibt.

if (1)

- Das „if“ Kommando dient als Abfrage.
- Die Syntax lautet:

```
if command
then
   command_list
fi
```

```
if command1
then
   command_list1
else
   command_list2
fi
```

if (2)

```
$ vi check_user2.sh
#!/bin/bash

if grep $1 /etc/passwd >/dev/null
then
   echo "I found" $1
else
   echo "There is no" $1
fi
$ chmod 700 check_user2.sh
$ check_user2.sh thomas
I found thomas
$ check_user2.sh mozart
There is no mozart
```

test (1)

- Der „test“ Befehl erlaubt die folgenden Prüfungen:
  - Länge eines Strings
  - Vergleich zweier Strings
  - Vergleich zweier Zahlen
  - Prüfung des „filetypes“
  - Prüfung der Berechtigungen eines Files
  - Logische Operatoren (and or not)
- Die Syntax lautet:

```
[ expr ]
```

test (2)

expr	Beschreibung
`-z string`	Die Länge des „string“ ist 0
`-n string`	Die Länge des „string“ ist ungleich 0
`string1 = string2`	Die beiden „strings“ sind gleich
`string1 != string2`	Die beiden „strings“ sind unterschiedlich
`string`	Der string ist nicht NULL

```
$ vi look_kurt.sh
#!/bin/bash

if [ $1 = "kurt" ]
then
   echo "It is kurt!"
fi
$ chmod 700 look_kurt.sh
$ look_kurt.sh thomas
$ look_kurt.sh kurt
It is kurt!
```

test (3)

expr	Beschreibung
`int1 -eq int2`	Die erste Zahl ist gleich groß wie die zweite Zahl. (equal)
`int1 -ne int2`	Die erste Zahl ist nicht gleich groß wie die zweite Zahl. (not equal)
`int1 -gt int2`	Die erste Zahl ist größer als die zweite Zahl. (greater than)
`int1 -lt int2`	Die erste Zahl ist kleiner als die zweite Zahl. (less than)
`int1 -ge int2`	Die erste Zahl ist größer gleich der zweiten Zahl. (greater equal)
`int1 -le int2`	Die erste Zahl ist kleiner gleich der zweiten Zahl. (less equal)

```
$ vi check_five.sh
#!/bin/bash

if [ $1 -lt 5 ]
then
   echo $1 "less than 5!"
fi
$ chmod 700 check_five.sh
$ check_five.sh 3
3 less than 5!
```

test (4)

expr	Beschreibung
`-r file`	Die Datei existiert und ist lesbar. (readable)
`-w file`	Die Datei existiert und ist schreibbar. (writeable)
`-x file`	Die Datei existiert und ist ausführbar. (executeable)
`-f file`	Die Datei existiert und ist eine normale Datei. (regular file)
`-d file`	Die Datei existiert und ist ein Verzeichnis. (directory)
`-s file`	Die Datei existiert und ist größer als 0. (size)

expr	Beschreibung
`!`	NOT Operator
`-a`	AND Operator
`-o`	OR Operator

test (5)

```
$ cat test_file.sh
#!/bin/bash

if [ -r $1 -a -s $1 ]
then
   echo $1 "a readable file and the size greater 0"
fi
$ chmod 700 test_file.sh
$ test_file.sh file1.txt
file1.txt a readable file and the size greater 0
$ test_file.sh not_existing.txt
```

case (1)

- Das „case“ Statement erlaubt dem Benutzer, einen Wert mit verschiedenen anderen Werten zu vergleichen.
- Die Wildcharacters „*“ und „?“ können verwendet werden.
- Die Syntax lautet:

```
case value in
   part1)   command-list1;;
   part2)   command-list2;;
esac
```

case (2)

```
$ vi calc_bas.sh
#!/bin/bash

case $2 in
   '+' ) echo "Ergebnis:" $[$1+$3];;
   '-' ) echo "Ergebnis:" $[$1-$3];;
   '*' ) echo "Ergebnis:" $[$1*$3];;
   *   ) echo "Unknown Operator";;
esac
$ chmod 700 calc_bas.sh
$ calc_bas.sh 5 + 3
Ergebnis: 8
$ calc_bas.sh 5 - 3
Ergebnis: 2
$ calc_bas.sh 7 \* 2
Ergebnis: 14
$ calc_bas.sh 7 \/ 2
Unknown Operator
```

for (1)

- Das „for“ Kommando wiederholt eine Sequenz von Befehlen für jeden Wert der Liste.
- Die Syntax lautet:

```
for variable in word...
do
   command-list
done
```

for (2)

```
$ vi week_days.sh
#!/bin/bash

for today in mon tue wed thu fri
do
   echo $today
done
$ chmod 700 week_days.sh
$ week_days.sh
mon
tue
wed
thu
fri
```

for (3)

```
$ vi print_param.sh
#!/bin/bash

count=0
for param in $*
do
   count=$[$count+1]
   echo $count parameter is $param
done
$ chmod 700 print_param.sh
$ print_param.sh *
1 parameter is anz_files.sh
2 parameter is calc_bas.sh
3 parameter is check_five.sh
...
13 parameter is test_shift.sh
14 parameter is week_days.sh
```

while (1)

- Der „while“ Befehl führt eine Folge von Kommandos aus bis die Bedingung den Wert „true“ erhält.
- Die Syntax lautet:

```
while expr
do
    command-list
done
```

while (2)

```
$ vi count_down.sh
#!/bin/bash

count=0
while [ $count -le 10 ]
do
   echo $count
   count=$[$count+1]
done
$ chmod 700 count_down.sh
$ count_down.sh
0
1
2
...
9
10
```

break (1)

- Das Kommando „break“ wird verwendet, um aus einer Schleife zu springen.
- Die Syntax lautet:

```
break [number]
```

break (2)

```
$ vi check_quit.sh
#!/bin/bash

for param in $*
do
   if [ $param = "quit" ]
   then
      break
   fi
   echo "Parameter:" $param
done
$ chmod 700 check_quit.sh
$ check_quit.sh 7 5 4 hello quit my path is `pwd`
Parameter: 7
Parameter: 5
Parameter: 4
Parameter: hello
```

continue (1)

- Dieses Kommando überspringt die restlichen Zeilen der Schleife und beginnt am Anfang der Schleife.
- Die Syntax lautet:

```
continue
```

continue (2)

```
$ vi check_skip.sh
#!/bin/bash

for param in $*
do
   if [ $param = "skip" ]
   then
      continue
   fi
   echo "Parameter:" $param
done
$ chmod 700 check_skip.sh
$ check_skip.sh 7 4 skip path is skip `pwd`
Parameter: 7
Parameter: 4
Parameter: path
Parameter: is
Parameter: /home/thomas/uebD
```