Linux Tutorium

12. Shellprogrammierung

Version vom 02.07.2008 13:38:56

Shellprogrammierung

- im Grunde ist ein Shell-Skript nichts anderes als eine Textdatei, welche Befehlsfolgen enthält
- Shell-Skripte werden im Wesentlichen aus zwei Gründen geschrieben
  1. man kann so ständig wiederkehrende Kommandos zusammenfassen und diese durch einen einfachen Aufruf starten (z.B. Backup von Logfiles)
  2. weil man so einfache Programme schreiben kann, welche relativ intelligente/komplexe Aufgaben erledigen können (z.B. Kopie einer Audio-CD -> umwandeln in MP3 Format -> Archivierung der Stücke nach Spieldauer)

Shellprogrammierung

- für komplexe, zeitkritische oder langwierige Aufgaben sollte man besser zu mächtigeren Sprachen wie Perl, Python oder in Extremfällen zu C / C++ / Java greifen
- Ablauf
  - zu Beginn muss mit Hilfe eines Editors (z.B. „nano“ oder „gedit“) eine Textdatei angelegt werden, in die der Quelltext geschrieben wird
  - ist der gewünschte Quellcode enthalten sollte die Datei unter geeignetem Namen gespeichert werden z.B. backup.sh

Shellprogrammierung

- hierfür nicht den Namen „test“ verwenden, es existiert ein Unix-Systemkommando mit diesem Namen
- Shell-Skripte werden mit der Endung .sh versehen
- nach dem Abspeichern der Datei unter einem geeigneten Namen muss diese ausführbar gemacht werden (z.B. `chmod 755 backup.sh`)
- sind die benötigten Rechte gesetzt kann das Skript gestartet werden (am Beispiel von backup.sh)
  - im lokalen Verzeichnis durch `./backup.sh`
  - In einem beliebigen Ordner
    `# /ordner/ordner2/backup.sh`

Shellprogrammierung

- Aufbau
  - ein Shell-Skript beginnt mit der Angabe des Kommandointerpreters (Zeile 1 ist der Pfad zu dem Programm, das die folgenden Zeilen interpretieren kann)

    `#!/bin/sh` Verweis auf die Bourne-Shell

    - die restlichen Zeilen des Skripts werden von dieser Shell ausgeführt
    - dies kann alternativ auch eine andere Shell als die Bash sein
  - danach kommen meist Kommentare, die Meta-Informationen über das Shell-Skript enthalten z.B.

Shellprogrammierung

- Aufbau(2)

```
# Author:  $author
# Last Modification: $date
```

  - es folgen Kommandos und Kontrollstrukturen
  - Kommandos sind durch eine neue Zeile oder Semikolon zu trennen

Shellprogrammierung

- Kommandos und Kontrollstrukturen
  - Kommentare
    - in Shellskripts können über das Raute-Zeichen (#) Kommentare eingeleitet werden
    - Ausgenommen von dieser Kommentarregel sind jedoch die ersten zwei Zeichen der ersten Zeile, die den Interpreter angeben

```
#!/bin/sh
# ich bin ein Kommentar
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(II)
  - Variablen
    - in einem Shell-Skript hat man verschiedene Möglichkeiten, Variablen einzusetzen
    - anders als in den meisten modernen Programmier-sprachen gibt es keine Datentypen wie Integer, Float oder Strings
    - alle Variablen werden grundsätzlich als „String“ gespeichert
    - soll die Variable die Funktion einer Zahl übernehmen, dann muss das verarbeitende Programm die Variable entsprechend interpretieren (für arithmetische Operationen steht das Programm expr zur Verfügung)

Shellprogrammierung

- Kommandos und Kontrollstrukturen(III)
  - Variablen(2)
    - es ist auch möglich, in einer Variable einen Shell-Befehl abzulegen
    - die Belegung einer Variablen erfolgt, indem man dem Namen mit dem Gleichheitszeichen einen Wert zuweist
    - es darf zwischen dem Namen und dem Gleichheits-zeichen keine Leerstelle stehen, ansonsten erkennt die Shell den Variablennamen nicht als solchen und versucht, ein gleichnamiges Kommando auszuführen

```
tier=Hund
echo $tier
Hund
```

```
var1=`pwd`
echo $var1
/home/gates
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(IV)
  - Variablen(3)
    - um auf den Inhalt einer Variablen zuzugreifen, leitet man den Variablennamen durch ein $-Zeichen ein
    - das Quoten dient dazu, bestimmte Zeichen mit einer Sonderbedeutung vor der Shell zu 'verstecken' um zu verhindern, dass diese ersetzt werden (var=abc)
      - \ das Zeichen nach einem \ wird wörtlich genommen
      - ' ' alles zwischen diesen Zeichen wird wörtlich genommen, mit Ausnahme eines weiteren ' und \

        (echo '$var' liefert als Ausgabe $var)
      - " " alles zwischen diesen Zeichen ist buchstaben-getreu zu interpretieren (echo "$var" liefert abc)

Shellprogrammierung

- Kommandos und Kontrollstrukturen(V)
  - Variablen(4)
    - Beispiele Quoting

```
$ echo 'Ticks "schützen" Anführungszeichen'
Ticks "schützen" Anführungszeichen
$ echo "Ist dies ein \"Sonderfall\"?"
Ist dies ein "Sonderfall"?
$ echo "Es gibt `ls | wc -l` Dateien in `pwd`"
Es gibt 43 Dateien in /home/rschaten
$ echo "Der Wert von \$x ist $x"
Der Wert von $x ist 100
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(VI)
  - Vordefinierte Variablen
    - es gibt eine Reihe vordefinierter Variablen, deren Benutzung ein wesentlicher Bestandteil des Shell-Programmierens ist
    - alle aktuell definierten Variablen können durch das Kommando „set“ aufgelistet werden
    - wichtig sind u.a.

HOME	Home-Directory (absoluter Pfad)
PATH	Suchpfad für Kommandos und Skripts
PS1	System-Prompt ($ oder #)
PS2	Prompt für Anforderung weiterer Eingaben (>)
SHELL	Name der Shell

Shellprogrammierung

- Kommandos und Kontrollstrukturen(VII)
  - Variablen-Substitution
    - unter Variablen-Substitution versteht man verschiedene Methoden, um die Inhalte von Variablen zu benutzen
    - das umfasst sowohl die einfache Zuweisung eines Wertes an eine Variable als auch einfache Möglichkeiten zur Fallunterscheidung
    - z.B.

`${var:-value}`	benutze *var* wenn gesetzt, sonst *value*
`${var:?value}`	*benutze var wenn gesetzt, sonst gib value aus*
`${#var}`	*Anzahl der Zeichen in var*

Shellprogrammierung

- Kommandos und Kontrollstrukturen(VIII)
  - Bedingte Anweisungen

    if

    - hat einen einfachen Aufbau, wobei man stets darauf achten sollte, die nötigen Leerzeichen um die eckigen Klammern zu setzen
    - hinter dem if-Befehl selbst folgt die in der Regel in eckigen Klammern eingeschlossene *Bedingung*
    - darauf folgt das Schlüsselwort then, das die Anweisungen einleitet, die abgearbeitet werden sollen, sofern diese Bedingung erfüllt ist

    - elif

      - ist die erste Bedingung nicht erfüllt, wird (wenn vorhanden) die nächste Bedingung über elif geprüft

Shellprogrammierung

- Kommandos und Kontrollstrukturen(IX)
  - Bedingte Anweisungen(2)
    - ist auch diese Bedingung nicht erfüllt, werden die restlichen elif-Bedingungen so lange überprüft, bis entweder eine Bedingung erfüllt oder keine weitere elif-Bedingung mehr vorhanden ist
    - else
      - sind alle Bedingungen nicht erfüllt, werden (wenn vorhanden) die Anweisungen hinter dem else-Schlüsselwort ausgeführt
    - fi
      - die if-Anweisung wird durch das Schlüsselwort fi beendet
    - eine if-Anweisung muss nicht zwingend eine elif- oder eine else-Anweisung beinhalten!

Shellprogrammierung

- Beispiel if-Anweisung

```
if [ Bedingung ]
then
   Anweisung1
   Anweisung2
elif [ BedingungN ]
then
   Anweisung1
   Anweisung2
else
   Anweisung1
fi
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(IX)
  - Bedingte Anweisungen(3)

case

- bei case werden die möglichen „Fälle“, die eintreten können, explizit angegeben
- das von der if-Anweisung bekannte else-Schlüsselwort finden wir in ähnlicher Form (durch den Fall „ * “) auch für die case-Anweisung vor
- ein Anweisungsblock für einen Fall ist durch zwei Semikolons abzuschließen

- *)
  - der Fall *) enthält die Anweisungen, die ausgeführt werden sollen, wenn keiner der zuvor aufgeführten Fälle eintritt
- esac
  - die case-Anweisung wird durch das Schlüsselwort esac beendet

Shellprogrammierung

- Beispiel case-Anweisung

```
ZAHL=5
case $ZAHL in
   3)
     echo "Die Zahl ist 3!"
     ;;
   4)
     echo "Die Zahl ist 4!"
     ;;
   *)
     echo "Zahl ist nicht 3 oder 4!"
     ;;
esac
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(X)
  - Schleifen
    - auch in der Shell stehen einem bei Bedarf verschiedene Schleifen zur Verfügung
    - while, for, until, seq
    - while und for sind in ihrer Funktion von anderen Programmiersprachen bekannt
    - neu hinzu kommt die until-Schleife, die eine Art Negation der while-Schleife darstellt

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XI)
  - Schleifen

    while

    - die while-Schleife setzt sich aus einer bedingten Anweisung, wie sie bereits aus der if-Anweisung bekannt ist, und einem Anweisungsblock zusammen
    - der Anweisungsblock wird durch das Schlüsselwort *do* eingeleitet und durch das Schlüsselwort *done* beendet

```
while [ Bedingung ]
do
   Anweisung1
   Anweisung2
   ...
done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XII)
  - Schleifen(2)

    for

    - for arbeitet einen Anweisungsblock mit jedem angegebenen Wert einmal durch
    - dieser Wert wird während des Durchlaufens des Anweisungsblocks an die Variable VAR zugewiesen, mit der die Anweisungen arbeiten können
    - der Anweisungsblock beginnt mit *do* und wir mit *done* beendet

```
for VAR in WERT1 WERT2 WERT3
do
   Anweisung1
   Anweisung2
done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XIII)
  - Schleifen(3)

    until

    - der Anweisungsblock der until-Schleife wird so lange ausgeführt, wie die Bedingung nicht erfüllt ist

```
while [ ! 1 ]; do echo Test; done
until [ 1 ]; do echo Test; done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XIV)
  - Schleifen(4)

seq

- seq wurde entwickelt um eine Schleife mit einer Ziffernfolge durchlaufen zu können
- primitivste Form einer Zählschleife
- soll seq beispielsweise alle Zahlen von 1 bis 3 auflisten, dann ist dies ganz einfach (man übergibt den Start- und Endwert)
- möchte man seq nun in eine Schleife packen, wird am besten die Kommandosubstitution verwendet

```
for i in `seq 1 3`
 do
    echo "$i"
done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XV)
  - Schleifen(5)

    break

    - um eine Schleife mitten im Durchlauf an einer beliebigen Position zu verlassen, muss man nur das break-Schlüsselwort an die jeweilige Stelle setzen
    - dies funktioniert sowohl mit while- als auch mit for- und until-Schleifen

```
while true
do
anweisung1
anweisung2 && break
done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XVI)
  - select
    - ein Menü lässt sich schnell und gleichzeitig einfach in ein Skript einbauen, indem man die select-Anweisung verwendet
    - der Aufbau von select gestaltet sich ähnlich wie der Aufbau der for-Schleife

```
select VAR in WERT1 WERT2
do
   Anweisung1
   Anweisung2
   ...
done
```

Shellprogrammierung

- select-Beispiel

```
#!/bin/bash
echo "Was haben Sie für ein Haustier?"
select HAUSTIER in Hund Katze Maus
do
echo "Sie haben also ein/eine(n) $HAUSTIER"
echo "Kann Ihr Haustier programmieren?"
done
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XVI)
  - Funktionen
    - eine Funktion in der Shell besteht aus einem Funktionskopf und einem Anweisungsblock
    - der Funktionskopf gibt den Namen der Funktion an

`function NAME` oder `NAME()`

    - der Anweisungsblock wird durch geschweifte Klammern begrenzt

```
Funktionsname()
{
   Anweisung1
   Anweisung2
}
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XVII)
  - Funktionen(2)
    - Shell-Skripts können mit Parametern aufgerufen werden, auf die über ihre Positionsnummer zugegriffen werden kann

$#	Anzahl der Argumente
$0	Name des Kommandos
$1-$9	1.-9. Argument
$@	alle Argumente
$*	alle Argumente konkateniert

    - anders als bei Programmiersprachen wie C werden die Funktionsparameter nicht zuvor festgelegt, benannt oder bekommen gar Datentypen

Shellprogrammierung

- Beispiel Funktionsparameter
  - nachfolgend wird eine Funktion beschreiben, die das Quadrat einer Zahl bildet

```
quadr()
{
    expr $1 \* $1
}
```

  - wird die Funktion nun aufrufen, übergeben wir einfach eine Zahl als Parameter($1 = 3)

```
$ quadr() { expr $1 \* $1; }
$ quadr 3   ->    9
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XVIII)
  - Funktionen(3)
    - Shell-Funktionen haben noch eine weitere Eigenschaft: Sie können einen Wert zurückgeben
    - man nennt diesen Wert daher den Rückgabewert
    - mithilfe dieses Rückgabewertes kann in der Shell geprüft werden, ob die Ausführung einer Funktion oder eines Programms erfolgreich war
    - dabei wird der Rückgabewert in der Variable $? gespeichert
    - um diese Funktion im eigenen Shellskript zu nutzen, baut man die Anweisung *return* in den Funktionscode ein

Shellprogrammierung

- Beispiel Rückgabewert

```
dols() {
cd $1
if [ "$?" = "1" ]; then return 1; fi
ls
if [ "$?" = "1" ]; then return 1; fi
return 0
}
```

```
$ dols /root; echo $?   (Aufruf der Funktion)
cd: /root: Permission denied
1
```

Shellprogrammierung

- Kommandos und Kontrollstrukturen(XIX)
  - die Anzahl an weiteren Kommandos, Operationen und Möglichkeiten ist gigantisch
  - u.a. Arrays, Datenströme, Mustererkennung, Arithmetische Ausdrücke, ...