Bash-Skripting

Linux-Kurs der Unix-AG

Sebastian Weber

07.01.2013

Was ist ein Bash-Skript?

- Skript muss mit `chmod +x` ausführbar gemacht sein
- Aneinanderreihung von Befehlen
- „normale“ Befehle nutzbar

Was ist ein Bash-Skript?

- Skript muss mit `chmod +x` ausführbar gemacht sein
- Aneinanderreihung von Befehlen
- „normale“ Befehle nutzbar

Einfache Beispiele

```
1 #!/bin/bash
2 echo "Hello, World!"
```

- `echo Argument` gibt das Argument an die Standardausgabe aus
- `echo -n Argument` gibt das Argument ohne Zeilenumbruch aus

Was ist ein Bash-Skript?

- Skript muss mit `chmod +x` ausführbar gemacht sein
- Aneinanderreihung von Befehlen
- „normale“ Befehle nutzbar

Einfache Beispiele

```
1 #!/bin/bash
2 echo "Hello, World!"
```

- `echo Argument` gibt das Argument an die Standardausgabe aus
- `echo -n Argument` gibt das Argument ohne Zeilenumbruch aus

```
1 #!/bin/bash
2 cd ..
3 ls
```

- `#!` gibt an, mit welchem Programm das Skript ausgeführt werden soll

```
#!/bin/bash
```

- Standard-Nutzer-Shell in Debian

```
#!/bin/sh
```

- /bin/sh immer vorhanden.
- verweist manchmal auf /bin/bash, häufig jedoch nicht.
- Kann weniger als /bin/bash

Variablen

Shellvariable

- wird mit `Variable=Wert` definiert
- mit `$Variable` kann auf die Variable zugegriffen werden
- löschen mit `unset`

Variablen

Shellvariable

- wird mit `Variable=Wert` definiert
- mit `$Variable` kann auf die Variable zugegriffen werden
- löschen mit `unset`

Umgebungsvariable

- Weitergabe an Kindprozesse
- Definition durch: `export VARIABLE`
- Mit `export -n` wird aus einer Umgebungsvariable wieder eine Shellvariable

Programmparameter

- Programme werden mit `./programm` aufgerufen
- es können Parameter angegeben werden
- auf diese kann mit `$1`, `$2` zugegriffen werden
- `$*` enthält alle Parameter
- `$#` steht für die Anzahl der Parameter

Exit-Status

- Programme können mit oder ohne Fehler beendet werden
- Kein Fehler bedeutet Rückgabewert 0
- „Fehler“ erzeugen einen Rückgabewert größer 0
- Kann mit `$?` ausgelesen werden
- Kann durch `exit Wert` gesetzt werden (Bsp. `exit 33`)

test

- test stellt fest, ob eine Bedingung wahr ist
- `-e file` (exists) wahr, wenn die Datei existiert
- `-d file` (directory) wahr, wenn die Datei ein Ordner ist

test

- `s1 = s2` wahr, wenn der String s1 identisch mit s2 ist
- `s1 != s2` wahr, wenn sie nicht identisch sind

test

- `n1 -eq n2` (equal) wahr, wenn n1 und n2 gleich sind
- `n1 -ne n2` (not-equal) wahr, wenn n1 und n2 nicht gleich sind
- `n1 -gt n2` (greater then) wahr, wenn n1 größer als n2 ist
- `n1 -lt n2` (lower then) wahr, wenn n1 kleiner als n2 ist

test

- `[]` macht das selbe wie `test`
- `[ $? -eq 0 ]` wahr, wenn Rückgabewert gleich 0
- Leerzeichen zwischen den Klammern und Befehlen sind wichtig!

if-Bedingung

```
1 if Bedingung
2 then
3 Befehl
4 elif Bedingung #elif ist optional
5 then #wird von elif benoetigt
6 Befehl
7 else     #else ist optional
8 Befehl
9 fi
```

- Die Befehle hinter if werden ausgeführt, wenn die Bedingung wahr ist bzw. Rückgabewert 1 liefert
- Ansonsten wird der Befehl hinter else ausgeführt
- elif bedeutet else if
- else und elif sind optional

for-Schleifen

```
1  for i in XXX
2  do
3  Befehl
4  done
```

- XXX kann eine Liste oder ein Befehl sein
- Befehle müssen `‘befehl‘` oder `$(befehl)` geschrieben werden
- `for i in $*` geht alle Parameter durch

Beispiele

```
1 #!/bin/bash
2 liste='1 2 3'
3 for i in $liste
4 do
5 echo $i
6 done
```

```
1 #!/bin/bash
2 for i in `seq 1 3`
3 do
4 echo $i
5 done
```

while-Schleifen

```
1 while Bedingung
2 do
3 befehl
4 done
```

- Die Schleife wird wiederholt, solange die Bedingung wahr ist

größere Programme

```
1  #!/bin/bash
2  echo Was ist das richtige Ergebnis?
3  echo -n "2+2="
4  while true
5  do
6  read i
7  if [ $i -eq 4 ]
8  then
9  echo "richtig!"
10 break
11 else
12 echo "falsch!"
13 sleep 10
14 echo -n "2+2="
15 fi
16 done
```

nützliche Dinge

- `read` wartet auf eine Eingabe ins Terminal. Wird mit Enter bestätigt
- `read -s` liest die Eingabe ohne sie anzuzeigen
- `sleep 10` wartet 10 Sekunden ab
- `$((Ausdruck))` wertet den Ausdruck arithmetisch aus
- `x=$(($x+1))` erhöht x um eins