LinuxCampus.net
trainings from the experts

3100 Bash Shell Scripting Teil 1

Shell Script Grundlagen

PETER JAHN
SYSTEM ENGINEER
TRAINING@LINUXCAMPUS.NET

Kurs Agenda

Shell Script Grundlagen

- Starten von Scripts
- Dokumentation
- Debugging

Der Befehl echo

Übung

- Erweitern des Shell Scriptes backup.sh

```
#!/bin/bash
# backup2.sh
# Backup geeko's home directory to /backup using rsync

echo "Backing up /home/geeko to /backup/"
rsync  -a  --no-whole-file   /home/geeko  /backup
```

Aufgabe

- Welche Auswirkung hat dieser Befehl?

```
echo -e "Shell\tScript\nKurs"
```

- Was ist der Zweck von?

  -e \t \n

Der Befehl "echo"

OPTION	BEDEUTUNG
-n	kein Zeilenumbruch nach Ausgabe
-e	aktivieren der Backslash Sequenzen

-e

BACKSLASH SEQUENZEN	
\\	Backslash
\a	Soundsignal (Alert)
\b	letztes Zeichen löschen (Backspace)
\c	Zeilenumbruch verhindern
\f	in nächster Zeile selbe Pos. fortsetzen
\n	neue Zeile
\r	Zeile löschen
\t	Tabulatorsprung

Übung 1

```
#!/bin/bash
echo
echo "Das ist eine normale Zeile"
echo "Hier fehlt das -e! \n \t "
echo -e "Jetzt kommt ein Backslash:\\"
echo -e "Hier fehlt ein Buchstabe: FEHLER\bBERICHT"
echo -e "Jetzt kommt ein Zeilen\numbruch"
echo -e "Tabulatorsprünge:\tTAB\tTAB\t "
echo " "
```

Übung 2

- Einsatz von Fortsetzungszeilen
  - Ein einfacher \ bedeutet der Befehl wird in der nächsten Zeile fortgesetzt

```
#!/bin/bash
echo
echo "!!!Und jetzt das ganze mit einem echo Befehl!!!"
echo -e "\nJetzt kommt ein Backslash:\\ \n\
 Hier fehlt ein Buchstabe: FEHLER\bBERICHT \n\
 Jetzt kommt ein Zeilen\n umbruch  \n\
 Tabulatorsprünge:\tTAB\tTAB\t \n\n"
```

Übung 3

- Einfache Ausgabe
  - Zeilenumbruch verhindern

```
#!/bin/bash
echo -n "Datum und Uhrzeit:"; date
echo -n "Rechnername:"; hostname
```

DATE

- Ausgabeformate von date
  - date — Sat Mar 14 15:58:46 CET 2009
  - date -I — 2009-03-14
  - date "+%m-%d %H:%M" — 03-14 16:01
  - date "+%D, %r" — 03/14/09, 04:02:34 PM
  - date +%d.%m.%y — 14.03.09
  - date +%d.%m.%Y — 14.03.2009
  - date "+%e.%-m.%y, %l.%M %p" — 14.3.09, 4.05 PM
  - date "+%A, %e. %B %Y" — Saturday, 14. March 2009

+ …hier folgt das gewünschte Format

Übung 4

- Ausgabe des Datums

```
#!/bin/bash
#command_subs1.sh
#bash Schreibweise
echo "Heute ist der: $(date '+%m/%d/%Y')"
```

```
#ältere sh Schreibweise
echo "Heute ist der `date '+%m/%d/%Y'`"
```

Rechnen mit Date

- Vergangenheit
  - date --date="3 months ago" +%H%M-%d%m%Y
  - date --date="255 days ago" +%H%M-%d%m%Y
  - date --date="32 weeks ago" +%H%M-%d%m%Y
  - date --date="5 hours ago" +%H%M-%d%m%Y
  - date --date="5 minutes ago" +%H%M-%d%m%Y
- Zukunft
  - date --date="-5 hours ago" +%H%M-%d%m%Y
  - date --date="-5 minutes ago" +%H%M-%d%m%Y
  - Gleicher Syntax jedoch mit Minus vor der Zahl

Rechnen mit DATE

```
# Script löscht Jobs welche genau vor 6 Monaten gemacht wurden
tar czf home_$(date +%d%m%Y).tar.gz  /home
rm home_$(date --date="6 months ago" +%d%m%Y).tar.gz
```

```
# Protokolliert die Laufzeit von speziellen Script Aktionen
START=$(date +%s)
# start your script work here
      tar czf home.tar.gz  /home
# your logic ends here
END=$(date +%s)
DIFF=$(( $END - $START ))
echo "It took $DIFF seconds"
```

Startarten von Shell Scripts

Starten von Shell Scripts

- Möglichkeiten zum Starten eines Shell Scripts
  - Direkter Start
  - Shell-Aufruf
  - Punkt-Befehl

Direkter Start

- Direkter Aufruf
  - durch die Eingabe des Shell Script Namens

```
# Direkter Start eines Shell Scripts
peter@tux~/bin>  ./beispiel.sh
Hallo Welt
```

- Eigenschaften
  - das Script wird als normales Kommando gestartet
  - es wird eine Subshell gestartet, welche alle globalen Variablen und das aktuelle Verzeichnis erbt

Shell-Aufruf

- Aufruf
  - das Script wird als Argument einer Shell angegeben
  - sh beispiel.sh
  - bash beispiel.sh

```
# Starten eines Shell Scripts als Shell-Argument
peter@tux~/bin>  bash  beispiel.sh
Hallo Welt
```

- Eigenschaften
  - es wird eine Subshell gestartet, welche alle globalen Variablen und das aktuelle Verzeichnis erbt

Punkt Befehl

- Aufruf
  - Durch Punkt-Abstand-Scriptname . beispiel.sh
  - Durch source Scriptname source beispiel.sh

```
# Starten durch Include
peter@tux~/bin>  . beispiel.sh
Hallo Welt
```

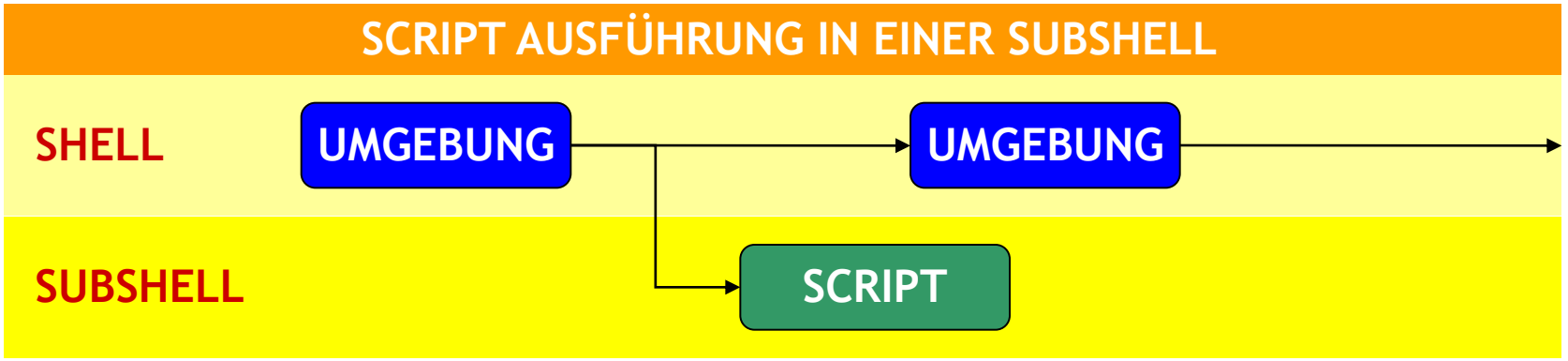
```
# Starten durch Include
peter@tux~/bin>  source beispiel.sh
Hallo Welt
```

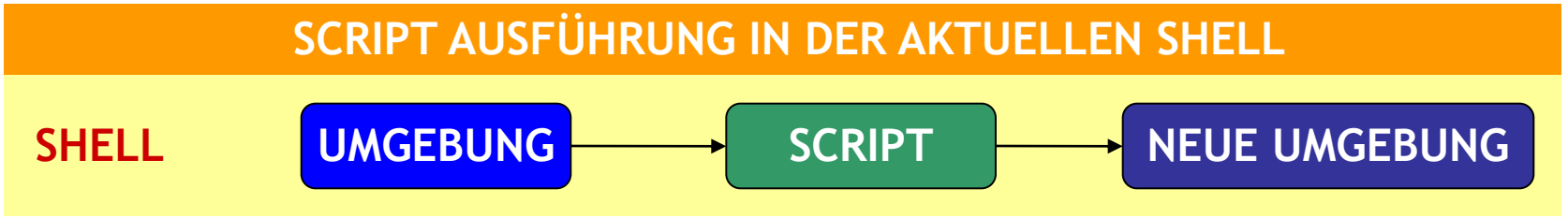
- Eigenschaften
  - es wird KEINE Subshell gestartet, das Script ist Teil der aktuellen Shell
  - kann die aktuelle Shell verändern

Script-Start Unterschiede

- Direkter Start ./beispiel.sh
  - Das Script muss ausführbar sein (chmod +x)
  - Shebang muss vorhanden sein (#!/bin/sh)
  - erzeugt Sub-Shell
- Shell-Aufruf sh beispiel.sh
  - Das Script muss NICHT ausführbar sein
  - Shellinterpreter wird beim Aufruf definiert
  - erzeugt Subshell
- Punkt-Befehl source beispiel.sh
  - Änderungen durch das Script bleiben auch nach dem Scriptende wirksam
  - erzeugt KEINE Subshell

Unterschied Shell & Subshell



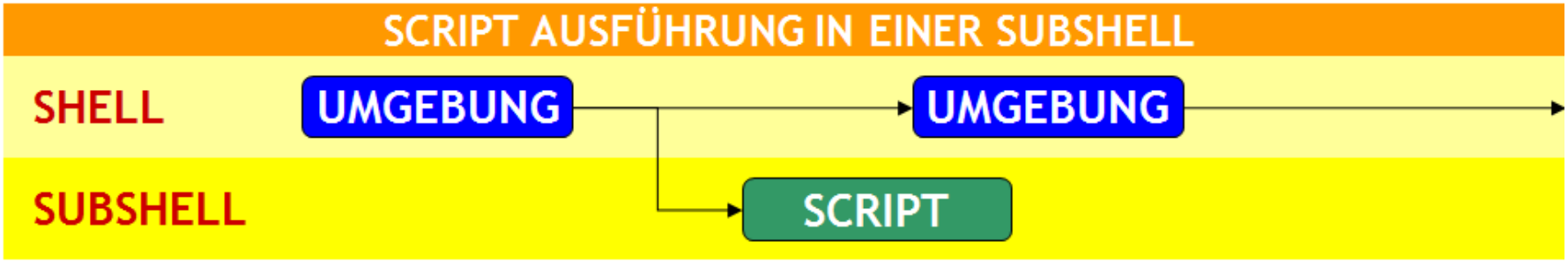


Aufgabe

- Aufgabe des Scripts
  - Anzeigen der aktuellen Prozessstruktur
  - Wechseln des aktuellen Verzeichnisses
  - Definieren der Variable "FIRMA"

```
#!/bin/sh
# Prozesse als Baumstruktur anzeigen
ps Tf
# aktuelles Verzeichnis wechseln
cd /tmp
# Variable definieren
FIRMA="www.TripleS.at"
```

Subshell




```
# Starten des Scripts als Subshell
peter@tux~/bin>  sh subtest.sh
  PID TTY      STAT   TIME COMMAND
 9972 pts/1    Ss     0:00 /bin/bash
10006 pts/1    S+     0:00  \_ sh subtest.sh
10007 pts/1    R+     0:00      \_ ps Tf
peter@asterix:~/bin>
```

```
# Änderungen überprüfen
peter@asterix:~/bin> pwd
/home/peter/bin
peter@asterix:~/bin> echo $FIRMA
```

- ps

wird in einer Subshell gestartet

- Verzeichnis

wurde nicht verändert

- Variable

ist nicht mehr gültig

ohne Subshell


```
# Starten des Scripts als Teil der Shell
peter@tux~/bin>  . subtest.sh
  PID TTY      STAT   TIME COMMAND
 9972 pts/1    Ss     0:00 /bin/bash
10007 pts/1    R+     0:00   \_ ps Tf
peter@asterix:/tmp>
```

```
# Änderungen überprüfen
peter@asterix:~/bin> pwd
/tmp
peter@asterix:~/bin> echo $FIRMA
www.TripleS.at
```

- ps

läuft in der aktiven Shell

- Verzeichnis

wurde verändert

- Variable

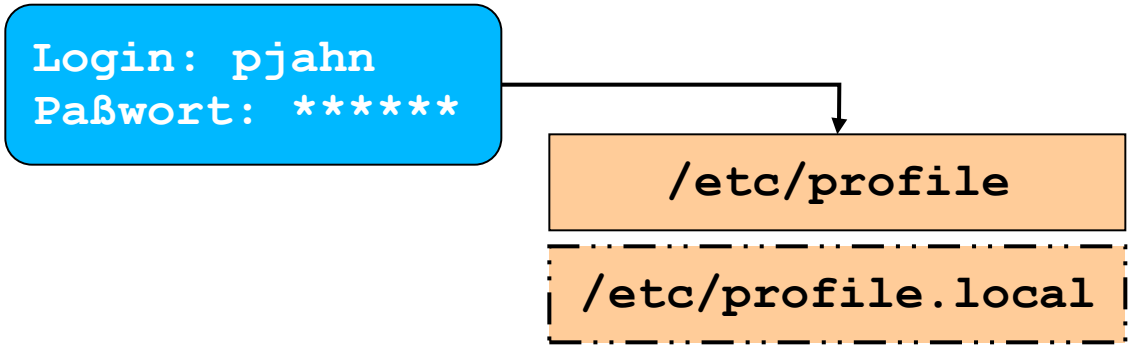
ist noch gültig

Beispiel von source

- Existiert die Datei profile.local?

```
test -s /etc/profile.local && . /etc/profile.local
```

-s	...Datei existiert und beinhaltet Daten
&&	...nur ausführen wenn Befehl1 erfolgreich war
.	...(Punkt=source) liest Inhalt in DIESE Shell ein



Shell Interpreter

- /bin/sh
  - ist ein Link zu /bin/bash
  - wenn /bin/sh verwendet wird startet die /bin/bash in einem Compatibilitätsmodus und nicht alle bash Funktionen stehen zur Verfügung

```
# /bin/sh im Detail
peter@tux~>  ll  /bin/sh
lrwxrwxrwx 1 root root 4 2006-06-13 08:12 /bin/sh -> bash
```

Beenden

Beenden eines Scripts

- Beenden eines Scripts
  - automatisch wenn das Script abgearbeitet ist
  - vorzeitig durch den Befehl "exit"
- Rückgabewert eines Scripts
  - informiert ob es mit Fehler abgeschlossen wurde
  - 0 ...ohne Fehler 1-255 ...mit Fehler
- Abfragen des Rückgabewertes
  - echo $?

Debugging

Debugging Optionen I

- Auf Syntaxfehler prüfen
  - Führt Befehle nicht aus, prüft nur auf Fehler
  - set -o noexec …in der Shell
  - bash -n script.sh …beim Scriptaufruf
  - #!/bin/bash -n …im Script

Debugging Optionen II

- Vor der Ausführung eines Befehles
  - erfolgt die Ausgabe mit echo

  - set -o verbose …in der Shell
  - bash -v script.sh …beim Scriptaufruf
  - #!/bin/bash -v …im Script

Debugging Optionen III

- Nach der Ausführung eines Befehles
  - erfolgt Ausgabe mit echo
  - ähnlich wie -v jedoch werden vor der Ausgabe Variablen und Sonderzeichen interpretiert

  - set -o xtrace …in der Shell
  - bash -x script.sh …beim Scriptaufruf
  - #!/bin/bash -x …im Script

Debugging Tipps

- Erleichtern der Fehlersuche
  - Einbauen von echo Meldungen

```
#!/bin/sh
...
echo "Debugging message 1."
...
echo "Debugging message 2."
...
```

Debugging Tipps

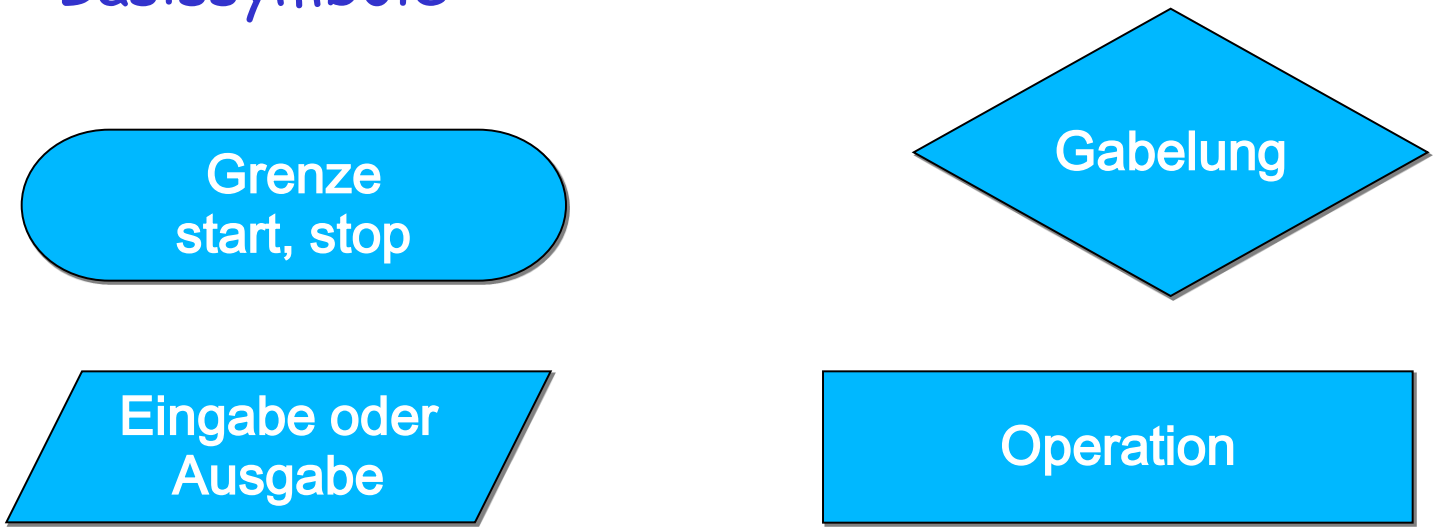
- Teile des Scripts debuggen
  - Innerhalb des Scripts Debugging Optionen setzen

```
#!/bin/sh
# Nur Scriptteile debuggen
...
set -x                  # activate debugging from here
w
set +x                  # stop debugging from here
...
```

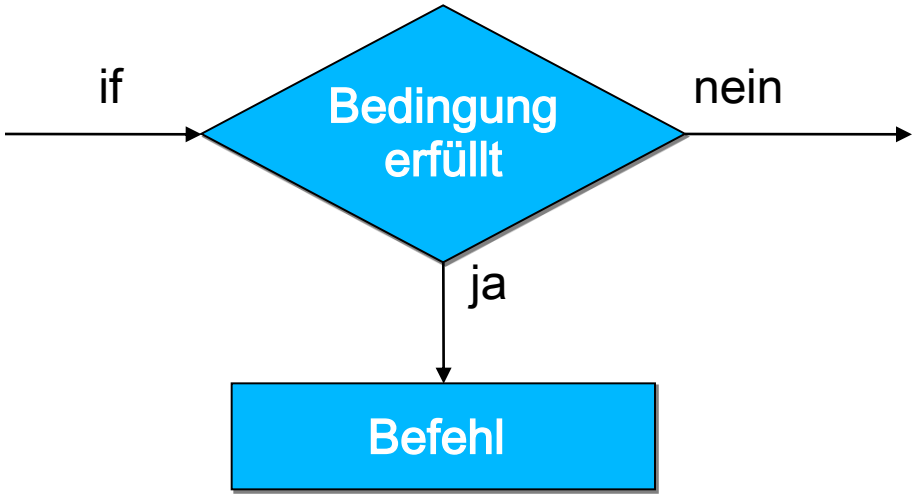
Flussdiagramme

Flussdiagramme

- Aufgabe von Flussdiagrammen
  - grafische Ansicht eines Programmes
  - zeigt genauen Ablauf des Programmes
  - hilft bei der Fehlersuche
- Basissymbole



Beispiel Verzweigung

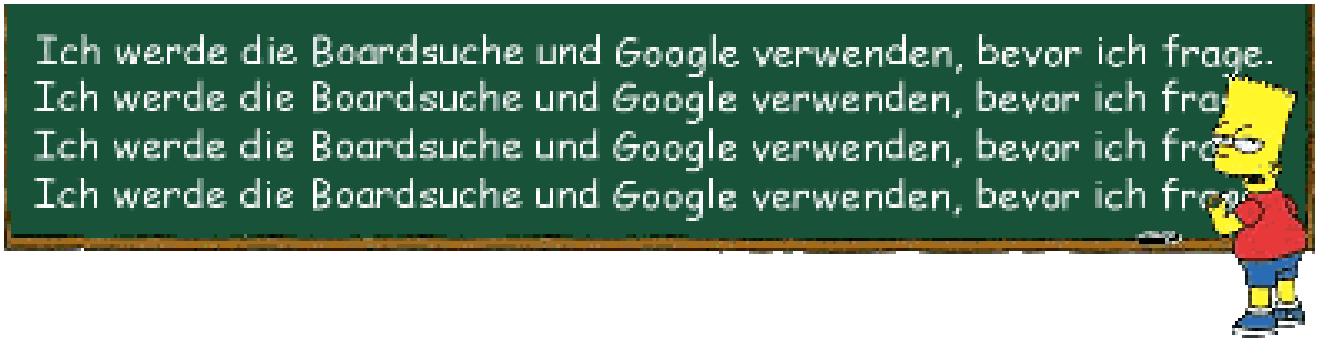


- Bedingung: bin ich als Root angemeldet?
- JA: Software installieren
- NEIN: Fehlermeldung: Bitte als Root anmelden

Tipp

- Gute Shellscripts Beispiele

  http://de.wikibooks.org/wiki/Linux-Kompendium:_Shellprogrammierung




LinuxCampus.net
trainings from the experts