Bash-Programmierung

Sven Velt

Was ist Bash-Programmierung?

- Automatisieren von gleichen/ähnlichen Abläufen
- Ausnutzung der Vorteile gegenüber anderen Systemen
- Unterstützung der Faulheit ;-)

Vorgaben für diesen Vortrag

Normale Schrift
- Anmerkungen
- Erklärungen

Courier-Font
- Ausgaben der Bash bzw. der aufgerufenen Befehle

- Eingabe an der Bash-Kommandozeile beginnen mit "$>"

Einfache Bash-Befehle (1)

- "echo" - Ausgabe

$> echo Hier kann ganz viel Text stehen...

Hier kann ganz viel Text stehen...

- "file" - Bestimmen eines Dateityps

$> file (BELIEBIGE_DATEI)

  - /bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), stripped

  - ./bashprogrammierung.mgp: ISO-8859 text

  - ./flyertuxine.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), "Created with The GIMPÿÛ", 72 x 72

Einfache Bash-Befehle (2)

- "#" - Ein Kommentar

```
$> # Hier darf alles stehen
$>
```

- "chmod" - Dateirechte ändern

```
$> ls -l backup.sh
-rw-r--r--  svelt  users  [...] backup.sh

$> chmod +x backup.sh

$> ls -l
-rwxr-xr-x  svelt  users  [...] backup.sh
```

Aller Anfang ist schwer...

Die erste Zeile eines Bash-Skriptes:

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

```
$> file bashprogrammierung.mgp
bashprogrammierung.mgp: ISO-8859 text

$> file bash-skript.sh
bash-skript.sh: Bourne-Again shell \
                script text executable
```

Wie fängt man am besten an?

Wir suchen uns etwas, was man nicht gerne tut

Voraussetzung: gleicher / ähnlicher Ablauf

BackUp-Skript, das eine Config-Datei liest

stdin, stdout, stderr

- Standard-Eingabe
  - normalerweise Tastatur
  - Umleitung durch "<"

$> html2text <eingabe.html

- Standard-Ausgabe
  - normalerweise Bildschirm, Xterm, Konsole
  - Umleitung durch ">"

$> make bzImage >kernel_compile_ausgabe

- Standard-Fehlerausgabe
  - normalerweise Bildschirm, Xterm, Konsole
  - Umleitung durch "2>"

$> make bzImage >k_c_out 2>k_c_err oder

$> make bzImage >k_c_out 2>&1

Variablen in der Bash

- Variablen sind Platzhalter
- können leer oder gefüllt sein
- Inhalt kann geändert werden

Belegung durch

```
VARIABLENNAME=inhalt
VARIABLE2="Ein Text mit Leerzeichen"
```

Verwenung durch

```
$VARIABLENNAME
```

backup.sh - Teil 1

```
#!/bin/bash

# Hier sollen unsere Backup-Dateien
# gespeichert werden
ZIELORT=/var/backup

# Hier liegt die Konfigurationsdatei
CFGDATEI=/etc/backup.cfg
```

Einlesen von Daten in Variablen (1)

"read" liest Daten von der Standardeingabe in Variablen

Aufruf
 read VAR1 VAR VAR3 ...

- Felder müssen durch Leerzeichen oder Tab ("Whitespaces") getrennt sein

- letzte Variable enthält den "Rest" bis zum Zeilenende

Einlesen von Daten in Variablen (2)

```
$> read V1 V2 V3
Eins Zwei und ab hier der Rest

$> echo $V1
Eins

$> echo $V2
Zwei

$> echo $V3
und ab hier der Rest
```

Eine Schleife

So oft durchlaufen, bis die Bedingung erfüllt ist

Unser Freund:

```
while (BEDINGUNG)
do
  # So oft wie nötig hier durchlaufen
done
```

Anmerkung: Jeder Befehl ist eine Bedingung!

backup.sh - Teil 2

```
while read ARCHIVNAME VERZ ZUSICHERN

do

  echo $ARCHIVNAME
  echo $VERZ
  echo $ZUSICHERN
  echo

done <$CFGFILE
```

tar - Erstellen von Archiv-Dateien

Tar kennt u.a. folgende Optionen

- c - Archiv anlegen (Create)
- x - Archiv entpacken (eXpand)
- t - Archiv ansehen

- z - Archiv packen mit gzip
- I / j - Archiv packen mit bzip2

- v - (Verbose)

- f - Angabe des Archivnamens

tar czf archiv.tar.gz datei verzeichnis/ ...

backup.sh - Teil 3

```
while read ARCHIVNAME VERZ ZUSICHERN

do

  cd $VERZ

  tar cIf $ZIELORT/$ARCHIVNAME.tar.bz2 \
        $ZUSICHERN

done <$CFGFILE
```

Anlegen der Konfigurationsdatei

```
$> cat /etc/backup.cfg

/daten textdokumente abiword/ \
        soffice/ word/

/daten spreadsheets gnumeric/ \
        excel/

/home/svelt meinhome ./
```

Zum Nachlesen

- http://lusc.de
- man bash
- diverse Einführungen im Internet