Einführung in die Betriebssysteme

Martin Spörl

Shellprogrammierung III – Bash Script

Grundlagen I

Grundlagen „Shell“

- Bildet die Benutzerschnittstelle zum Computer
- Grundidee:
  - User gibt Kommandos in Eingabezeile
  - Shell „interpretiert“ den Befehle
    - Darum auch „Kommandozeileninterpreter“
- Mehrere Shells verfügbar
- Benutzer hat Wahl welche Shell er nutzt
- Voreingestellt Shell wird nach Login gestartet

Bekannte Shells

- Bourne-Shell (Grundlage vieler heutiger Shells)
  - /bin/sh
- Bourne-Again-Shell
  - /bin/bash
- Kornshell
  - /bin/ksh
- Debian-Almquist-Shell
  - /bin/dash

Beispiel aus Vorlesung für diese Shell

Grundlagen II

Default Shell festlegen

- Datei */etc/passwd* editieren

```
pi@raspberrypi:~ $ grep "pi" /etc/passwd
pi:x:1000:4:,,,:/home/pi:/bin/bash
```

Default Shell

Aktuelle Shell ermitteln

- *echo $0*

```
pi@raspberrypi:~ $ echo $0
-bash
```

Shell wechseln

- */bin/<shell>*
- *exit* um Shell zu verlassen

```
pi@raspberrypi:~ $ /bin/sh
$ echo $0
/bin/sh
$ exit
pi@raspberrypi:~ $ echo $0
-bash
```

Häufige Befehle

Befehl	Erklärung	(Basic) Syntax	Beispiel
cp	Kopiert Datei(en)	cp <source> <target>	cp /tmp/MyFile /tmp/My_New_File
mv	Verschiebt eine Dateien/Ordner (auch umbennen	mv <source> <target>	mv /tmp/MyFile /tmp/My_New_File
rm	Löscht Datei(en) (auch rem)	rm <file>	rm /tmp/MyFile
cd	Wechselt das Verzeichnis	cd <directory name>	cd /tmp
ls	Listet Dateien/Ornder im Verziechnis	ls (aktuelles Verzeichenes) / ls <directory name>	ls /tmp
echo	Gibt einen Text aus	echo "<Text>"	echo "Hello World"
date	Gibt / setzt das aktuelle datum	date	date
cat	Gibt den Inhalt einer Datei aus (alles)	cat <filename>	cat /tmp/MyFile
less	Gibt den Inhalt einer Datei („soviel wie in Fenster passt“)	less <filename>	less /tmp/MyFile
mkdir	Erstellt einen Ordner	mkdir <directory name>	mkdir /tmp/MyDir

Allgemeine Hinweise

- Bei Leerzeichen Pfaden: "<Pfad>" schreiben
- *man <befehl>* gibt i.d.R. die Hilfe aus
- mehrere Befehle mit &&, ||, oder ; verknüpfbar
  - && - Befehle werden ausgeführt, wenn vorheriger erfolgreich
  - ; - Befehle werden ausgeführt, egal ob erfolgreich oder nicht
  - || - Befehl wird ausgeführt, wenn vorheriger nicht erfolgreich
- mit < (input) und > (output) kann der I/O Strom umgelenkt werden
- | (Pipe) verbindet Befehle (Stdout des ersten ist Input des zweiten)
- |& verbindet Befehle (Stdout und Stderr des ersten sind Input des zweiten (nicht alle Shells!))
- \ ist escape-Zeichen

Fehler von Aufrufen werden am Exit-Value erkannt:
Success bei “0”
Failure bei “>= 1”

Grundaufbau

„Shebang“ / „Hash-bang“
gibt Interpreter an

```
#!/bin/bash
#gibt Ordnerinhalt an
ls
```

Kommentar

Speichern als Basic_Script.sh Datei

Ausführbar machen:
chmod +x Basic_Script.sh

```
martin@ubuntu:~/Desktop/Vorlesung/ShellScripting$ ls -la
total 12
drwxr-xr-x 2 martin martin 4096 Nov  4 21:55 .
drwxr-xr-x 6 martin martin 4096 Nov  4 21:54 ..
-rw-r--r-- 1 martin martin   37 Nov  4 21:55 Basic_Script.sh
martin@ubuntu:~/Desktop/Vorlesung/ShellScripting$ chmod +x Basic_Script.sh
martin@ubuntu:~/Desktop/Vorlesung/ShellScripting$ ls -la
total 12
drwxr-xr-x 2 martin martin 4096 Nov  4 21:55 .
drwxr-xr-x 6 martin martin 4096 Nov  4 21:54 ..
-rwxr-xr-x 1 martin martin   37 Nov  4 21:55 Basic_Script.sh
martin@ubuntu:~/Desktop/Vorlesung/ShellScripting$ ./Basic_Script.sh
Basic_Script.sh
martin@ubuntu:~/Desktop/Vorlesung/ShellScripting$
```

Exkurs: Zugriffsrechte in Linux

r = **r**ead
w = **w**rite
x = e**x**ecute

r w x r w x r w x

Owner

Gruppe

Alle

Bitweise markiert

1 1 1 1 0 1 1 0 1

7

5

5

Variablen

Hinweise

- Keine Typisierung!
- Vordefinierte Variablen (Auszug)

Befehl	Erklärung
$0	Gibt den Scriptnamen aus
$?	Gibt exitstatus des vorherigen Befehls aus
$*	Alle Parameter als String
$@	Alle Parameter als Array
$#	Anzahl der Parameter
$UID	UID des Users aus /etc/passwd
$USER	Gibt den Benutzernamen aus
$HOME	Gibt das Homeverzeichnis des User zurück

In der Zuweisung sind keine Leerzeichen erlaubt!

```
#!/bin/bash
myvar=1234
echo $myvar
```

```
martin@ubuntu:~/ShellScripting/Variablen$ ./variablen.sh
1234
```

```
#!/bin/bash
echo "Hello $USER”
echo "Your home folder is at $HOME”
```

```
martin@ubuntu:~/ShellScripting/Variablen$ ./predefined_variables.sh
Hello martin
Your home folder is at /home/martin
```

Verwechslungsgefahr:
Bei Powershell starten Parameter mit 0 ($args[0])
Bei Shell/Bash mit 1 ($0 ist das Script selbst)!

Benutzerinteraktionen

Benutzereingabe lesen

```
#!/bin/bash
myvar=""
echo -n "Please enter your name: "
read myvar
echo "Hello $myvar"
```

-n gibt kein Newline bei Echo aus

Auf User warten

```
#!/bin/bash
echo let\'s do it!
echo ok first get some coffee
read -n1 -r -p "Press any key to continue..." key
echo ok - let\'s rock it!
```

-n = Anzahl der zu lesenden Zeichen
-r = RAW Mode (reagiert auf jedes Zeichen
-p = gibt Prompt Text an

Tipp: read kann mehrere Variablen einlesen (space-limited):

```
read myvar1 myvar2
```

String-Operationen

Substring

```
#!/bin/bash
mystr="Hello World!"
hello=${mystr:0:5}
echo $hello
world=${mystr:(-6)}
echo $world
world2=${mystr:6:5}
echo $world2
```

Startindex (0-Based)

Länge

Replace

```
#!/bin/bash
mystr="Hello World!"
newstr=${mystr/World/Universe}
echo $newstr
```

Search

Replace

Concat

```
#!/bin/bash
mystr1="Hello"
mystr2="World!"
constr="$mystr1 $mystr2"
echo $constr
```

Split

```
#!/bin/bash
mystr="Hello;World!"
echo "$(echo $mystr | cut -d';' -f1)"
echo "$(echo $mystr | cut -d';' -f2)"
```

Cut ist externes Programm

Length

```
#!/bin/bash
mystr="Hello World!"
echo "Der String is ${#mystr} Zeichen lang"
```

Schleifen & Verzweigung

Auf Leerzeichen achten!

while

```
#!/bin/bash
myvar=1
while [ $myvar -lt 10 ]
do
    myvar=$(($myvar+2))
    echo $myvar
done
echo "Done"
```

until

Läuft **bis** Bedingung wahr ist

```
#!/bin/bash
myvar=1
until [ $myvar -gt 10 ]
do
    myvar=$(($myvar+2))
    echo $myvar
done
```

if

Auf Leerzeichen achten!

```
#!/bin/bash
myvar=1
if [ $myvar -eq 30 ]
then
    echo "Yes!"
else
    echo "No!"
fi
```

if ! [… kehrt die Bedingung um

for

entspricht foreach

```
#!/bin/bash
echo "Variante 1"
for i in 1 2 3 4 5
do
        echo $i
done
echo "Variante 2"
for i in {0..10..1}
do
        echo $i
done
```

{Start..Ende..Inkrement}

Durch Brace-expand sind keine Variablen nutzbar!

Vergleichsoperatoren

Operator	Bedeutung	Beispiel
-eq	gleich („**eq**uals“)	$a -eq 1
-ne	ungleich („**n**ot **e**quals“)	$b -ne “World”
-lt	kleiner („**l**ower **t**hen“)	$a -lt 5
-le	kleiner oder gleich („**l**ess or **e**equals“)	$b -le 4
-gt	größer („**g**reater **t**hen“)	$a -gt $b
-ge	größer oder gleich („**g**reater or **e**quals“)	$b -ge 7

Strings werden mit "==" verglichen!

Bash Conditional Expression

Datei prüfen

```
#!/bin/bash
if [ $# -gt 0 ]
then
        if [ -f $1 ]
        then
        echo "$1 is a regular file and
exists"
        fi
else
       echo "At least 1 folder needs to be
passed!"
fi
```

Expression	Erklärung
-a	Datei existiert
-f	Datei existiert und ist eine reguläre Datei
-d	Datei existiert und ist ein Ordner
-v varname	$varname existiert und hat einen Wert
-z string	Länge von *string* ist 0
-n string	Länge von *string* ist nicht 0
file1 -nt file2	file1 ist neuer als file2
file1 -ot file2	file1 ist älter als file2

Arithmetische Operation I

Addition

```
#!/bin/bash
myvar1=1
myvar2=2
sum=$(($myvar1+$myvar2))
echo $sum
```

((und)) bei arithmetischen Operation zwingend („ arithmetic expansion“)

Depricated Schreibweise: $[<EXPRESSION>]

Subtraktion

```
#!/bin/bash
myvar1=1
myvar2=2
diff=$(($myvar1-$myvar2))
echo $diff
```

Multiplikation / Potenz

```
#!/bin/bash
myvar1=4
myvar2=2
mul=$(($myvar1*$myvar2))
echo $mul
pot=$(($myvar1**$myvar2))
echo $pot
```

Division (ganzzahlig)

```
#!/bin/bash
myvar1=1
myvar2=2
quot=$(($myvar1/$myvar2))
echo $quot
```

Modulo

```
#!/bin/bash
myvar1=1
myvar2=2
quot=$(($myvar1%$myvar2))
echo $quot
```

Shift

Nicht mit „shift“ befehl verwechseln!

```
#!/bin/bash
shiftleft=$((1<<2))
echo $shiftleft
shiftright=$((4>>1))
echo $shiftright
```

Arithmetische Operation II

Bitwise OR

```
#!/bin/bash
myvar1=1
myvar2=2
sum=$(($myvar1|$myvar2))
echo $sum
```

Bitwise AND

```
#!/bin/bash
myvar1=1
myvar2=2
sum=$(($myvar1&$myvar2))
echo $sum
```

Bitwise XOR

```
#!/bin/bash
myvar1=1
myvar2=2
sum=$(($myvar1^$myvar2))
echo $sum
```

unterschiedliche Zahlensysteme

```
#!/bin/bash
#Hex 1
myvar1=0x1
sum=$(($myvar1 + $myvar1))
echo $sum
```

Dateioperationen

Kompatibel mit /bin/sh (standard)

Datei einlesen

```
#!/bin/bash
tmp=`cat FileRead.sh`
echo "I read $tmp"

#for bash and newer shells
tmp=$(<FileRead.sh)
echo "And new I read $tmp"
```

Datei (über-)schreiben

```
#!/bin/bash
tmp="This is not readable"
echo $tmp > Test.txt
tmp2="This is readable"
echo $tmp2 > Test.txt
```

Datei anhängen

```
#!/bin/bash
tmp="This is readable"
echo $tmp > Test.txt
tmp2="This is also readable"
echo $tmp2 >> Test.txt
```

>> = Anhängen
> = Ersetzen

Übergabeparameter I

Übergabe Parameter

```
#!/bin/bash
echo "Hello my name is $0"
```

„$0“ ist immer aufrufendes Kommando

Erweiterte Parameter

```
#!/bin/bash
echo "I got $# Parameter!"
echo "There are $@"
```

„$1“ – „$9“ sind Positionsparameter

Parameter	Erklärung
$1 - $9	1. – 9. Parameter
${n}	N-ter Parameter (bei mehr als 9)
$#	Anzahl der Paramter
$@	Alle Parameter (als Array)
$*	Alle Parameter (als 1 String)

Übergabeparameter II

Shift Befehl

```
#!/bin/bash
echo "I got $# Parameter!"
i=1
while [ $# -gt 0 ]
do
        echo "$i: $1"
        i=$(($i+1))
        shift
done
echo "Ok all parameters gone!"
```

Löscht ersten Parameter

```
martin@ubuntu:~/ShellScripting2/Uebergabeparameter$ ./shift.sh a b c d e f
I got 6 Parameter!
1: a
2: b
3: c
4: d
5: e
6: f
Ok all parameters gone!
```

Funktionen

```
#!/bin/bash
function test {
        testResult=$(($1+$2))
}

echo "I am adding 1 and 2"
test 1 2
echo "And its $testResult"
```

Reihenfolge der Parameter entspricht Reihenfolge im Aufruf

- Bash kennt keine Rückgabewert!
  - nur über Workaround
- Workaround 1: globale Variable
- Workaround 2: Exit-Value
  - „return“ definiert Exit-Wert
  - Exit Wert kann später abgerufen werden

```
#!/bin/bash
function test {
        return $(($1+$2))
}

echo "I am adding 1 and 2"
test 1 2
echo "And its $?"
```

$? Gibt den Exit-Value des letzten Aufrufs

sed (stream editor)

Text in Dateien ersetzen

#Ersetzte alle „World“ mit „Universe“
sed -i s/World/Universe/g MyFile.txt

Direkt in der Datei bearbeiten (keine STDOUT Ausgabe)

Ohne g nur erstes Vorkommen ersetzt

Zeile in Dateien löschen

#Löscht alle Zeilen die mit „ToDel“ beginnen
sed -i '/^ToDel/d' MyFile.txt

Zeile in Dateien einfügen

#Fügt vor der 2ten Zeile ein
sed -i '2iI am an new Line' MyFile.txt

Dateien / Ordner suchen

bestimmte Ordner suchen

```
#!/bin/bash
find /home -name $USER -type d
```

bestimmte Dateien suchen

```
#!/bin/bash
find /bin -name "bash" -type f
```

Anhand von Attributen suchen

```
#!/bin/bash
if [ $# -gt 0 ]
then
        find $1 -mtime +20
else
       echo "At least 1 folder needs to be
passed!"
fi
```

Dateien durchsuchen

Text in bestimmter Datei suchen

```
#!/bin/bash
#this comment will be searched
grep -n "comment" $0
```

Text in mehreren Dateien suchen

```
#!/bin/bash
#this comment will be searched
grep -n -r "comment" ./*.sh
```

Bitgenaues kopieren von Daten

Kopieren ein ganzen Datei / Partition

```
#!/bin/bash
dd if=/dev/sda5 of=/dev/sdb5 #Partition
dd if=/home/myuser/myfile of=/home/myuser/myfile2
```

Extrahieren von bestimmten Bereichen

```
#!/bin/bash
#MBR extrahieren
dd if=/dev/sda of=mbr bs=1 count=512
hexdump mbr
```

Speicherverbrauch betrachten

Dateisystem Verbrauch

```
#!/bin/bash
echo "Your filesystems are used as followed"
df -h
```

Speichergrößen auf KB, MB, GB usw. vereinfachen

Datei / Folder Verbrauch

```
#!/bin/bash
if [ $# -gt 0 ]
then
        du -sh $1
else
       echo "At least 1 folder needs to be passed!"
fi
```