Einführung Unix Shell

Prof. Dr. Rüdiger Weis

Beuth Hochschule Berlin

Wintersemester 2012/2013

Unix Shells

- Bourne Shell (`/bin/sh`)
- bash (Bourne-again-shell)
- ash (Almquist shell)
- dash (Debian Almquist shell)
- Korn Shell
- csh
- tcsh
- ...

- Wikibook: Shellprogrammierung
  `http://de.wikibooks.org/wiki/Linux-Kompendium:_Shellprogrammierung`
- Übersicht Shells
  `http://en.wikipedia.org/wiki/Comparison_of_computer_shells`

Starten und Verlassen

sh	startet (weitere) Shell
exit <nummer>	Beendet die Shell mit Rückgabewert <nummer>

- Startzeile für Skripte in der ersten Zeile (Shebang)

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

- # für Kommentare

Dokumentation

Dokumentation

- man
- info
- apropos

Nützliche Komandos

wc	zählt Zeilen, Worte, Zeichen
more less	seitenweise Ausgabe
sort	sortieren von Zeilen
uniq	Entfernt identische aufeinanderfolgende Zeilen
grep	Programmierbarer Filter
sed	stream editor
tr	übersetzt Zeichen
ps	Zeigt Prozessinformationen
pstree	Prozessinformationen als Baum

Datei Komandos

ls	Liste Verzeichnisinhalt
cp	copy
mv	move
cat	Datei Ausgabe
cd	change dictory
pwd	print working directory
echo	Argument ausgeben
find	Datei finden
locate	Datei finden mit Index

stdin, stdout,stderr

0	stdin	Standardeingabe (Tastatur)
1	stdout	Standardausgabe (Terminal)
2	stderr	Standardfehler (Terminal)

Umlenkung

stdin	>	
(anhängen)	>>	
stdout	<	>&1
stderr	2 >	>&2

Beispiel: Umlenkung stderr

```
$ murks
bash: murks: command not found
$ murks 2> err
$ cat err
bash: murks: command not found
```

Beispiel: Ausgabe auf stderr

```
$ cat outerr.sh
#!/bin/bash
echo "Ausgabe auf stdout." >&1
echo "Ausgabe auf stderr." >&2

$ sh outerr.sh
Ausgabe auf stdout.
Ausgabe auf stderr.
$ sh outerr.sh 2>err
Ausgabe auf stdout.
$ cat err
Ausgabe auf stderr.
```

Pipes

Pipe

```
<prozess1>|<prozess2>
```

- Mittels des Pipesymbols | erreicht man die direkte Verknüpfung zweier Prozesse:
  - Ausgabe des ersten Prozesses bildet
  - Eingabe des zweiten Prozesses

Beispiel: Umlenkung und Pipes

```
$ echo "Hallo Welt."|wc
      1       2      12
$ echo "Hallo Welt.">hallo.txt
$ ls
hallo.txt
$ ls|wc
      1       1      10
```

Wildcards

*	beliebige Zeichenkette (incl. leere)
?	ein beliebiges Zeichen
$[c_1, c_2]$	Zeichen c_1 oder c_2
$[c_1 - c_2]$	Zeichen mit Code zwischen c_1 und c_2

Kommando-Gruppen

- ; trennt Kommandos
- () Gruppiert Kommandos

Steuert Umlenkungen.

Umgebungsvariablen

- Umgebungsvariablen binden Variablennamen an Zeichenketten.
- Weitergabe an Kindprozesse möglich.
- Shell ist Vaterprozess von dort aufgerufen Programmen.

Setzen und Löschen

- Setzen von Umgebungsvariablen
  - <name>=<string>
    String an Variable zuweisen
  - <name>=$(<cmd>)
    <name>= `<cmd>`
    Ausgabe von <cmd> an Variable zuweisen
- Löschen von Umgebungsvariablen
  - unset <name>

Zugreifen

- set
  zeigt alle Umgebungsvariablen an.
- export <name>
  macht Umgebungsvariablen für Kindprozesse sichtbar.
- $<name>
  liefert Wert der Umgebungsvariablen zurück.

Wichtige Umgebungsvariablen

- HOME : Homeverzeichnis
- HOSTNAME :Rechnername
- PS1 : Shell-Prompt
- PATH : Suchpfad für ausführbaren Programme

Vordefinierte Variablen

- ERRNO : Fehlernummer des letzten Systemaufrufes
- PWD : Aktuelles Verzeichnis
- OLDPWD : Vorheriges Verzeichnis

Vordefiniete Variablen

$0	Name des Skriptes
$1 ...$9	Aufrufparameter mit der Nummer n, $1 <= n <= 9$
$*	Alle Aufrufparameter als zusammenhängender String
$@	Alle Aufrufparameter als Folge von Strings
$#	Anzahl der Aufrufparameter
$?	Rückgabewert des letzten Kommandos
$$	Prozessnummer der aktiven Shell
$!	Prozessnummer des letzten Hintergrundprozesses

Quotierungen

'...' keine Ersetzung

"..." deaktiviert nur Wildcard-Ersetzung,
erlaubt Variablen-Ersetzung

`...` (Backticks) Verwendet Ausgabe des Kommandos alternativ:
$(...)

\ Sonderbedeutung für einzelnes Zeichen aufheben

Ersetzungen

```
$ echo date
date
$ echo "date"
date
$ echo ’date’
date
$ echo ‘date‘
Mo 26. Nov 22:55:42 CET 2007
$ echo $(date)
Mo 26. Nov 22:55:57 CET 2007
```

Ersetzungen

```
$ VAR=Hallo
$ echo $VAR
Hallo
$ echo "$VAR"
Hallo
$ echo '$VAR'
$VAR
$ echo "$VAR*"
Hallo*
$ echo '$VAR*'
$VAR*
$ echo \*
*
```

if Anweisung

if

```
if <command>
then
    <command>
[else
    <command>]
fi
```

- Wenn Rückgabewert von <command> in if Zeile 0 ist.
  - dann führe then-Zweig aus.
  - sonst führe else-Zweig aus.

test

test

```
test <command>

[ $<command> ]
```

Beachte: Leerzeichen hinter [und vor].

- Boolsche Ausdrücke
- Zahlen-Vergleiche
- String-Vergleiche

test Optionen I

-d file	Directory
-f file	File
-s file	nicht leeres File (size)
-r file	Read-Permission
-w file	Write-Permission
-x file	Execute-Permission
-z string	Leerstring (zero)
=	Strings gleich
!=	Strings ungleich

test Optionen II

-eq	Zahlen gleich (equal)
-ne	Zahlen ungleich
-gt	grösser (greater than)
-ge	grösser gleich
-lt	kleiner (less than)
-le	kleiner gleich
!	nicht
-a	und (and)
-o	oder (or)

Boolsche Ausdrücke

(<A>)	Ausdruck auswerten
<A> -o <B>	Logisches OR
<A> -a <B>	Logisches AND
!<A>	Logisches NOT

Zahlen-Vergleiche

<zahl$_1$> -eq <zahl$_2$>	Gleich
<zahl$_1$> -ne <zahl$_2$>	Ungleich
<zahl$_1$> -ge <zahl$_2$>	Grössergleich
<zahl$_1$> -gt <zahl$_2$>	Grösser
<zahl$_1$> -le <zahl$_2$>	Kleinergleich
<zahl$_1$> -lt <zahl$_2$>	Kleiner

String-Vergleiche

<string$_1$> = <string$_2$>	Gleich
<string$_1$> != <string$_2$>	Ungleich
-n <string>	String nicht leer
-z <string>	String leer

Beispiel: if Anweisung

```
#!/bin/sh
uhrzeit=$(date +%H)
if [ $uhrzeit -lt 14 ]
then
    echo "Guten Morgen."
else
    echo "Guten Tag."
fi
```

while Anweisung

while

```
while <command>
do
        <command>
done
```

- Wenn Rückgabewert von <command> in while Zeile 0 ist.

Beispiel: while Anweisung

```
i=0
while [ $i -le 10 ]
do
    i=$(expr $i + 1)
    echo "$i"
done
```

case Anweisung

case

```
case <text> in
    [<pattern>)
     <command>
     ;;]
esac
```

- Nur erster passender Fall wird ausgeführt
- *) für Default-Fall

Beispiel: case

```
$ cat rufe.sh
case $1 in
   Hund) echo "Wau!";;
   Katze) echo "Miau!";;
   *) echo "Hallo!";;
esac


$ sh rufe.sh Hund
Wau!
$ sh rufe.sh Katze
Miau!
$ sh rufe.sh
Hallo!
```

for Anweisung

for

```
for <variable> in <liste>
do
    <command>
done
```

- <variable> wird bei jedem Durchlauf das nächste Listenenelment zugewiesen.

Beispiele: for Schleife

```
for x in Technische Fachhochschule Berlin
    do echo $x
    done
```

Beispiel: Aufrufparameter

```
for para in $@
do
   echo "Aufrufparameter : " $para
done
```

Beispiel: txt Dateien finden

```
for i in $(find ~ -name "*.txt")
    do echo $i
    done
```

Harte Links

`ln quelle linkname`

- Zeiger auf Dateien
- Nicht möglich für Verzeichnisse.
- Keine harte Links über Gerätegrenzen
- Löschen, Umbenennen und Verschieben der Quelldatei haben keinen Einfluss auf Link.

Beispiel: Harte Links

```
$ echo "Dateiinhalt">datei
$ cat datei
Dateiinhalt
$ ln datei verweis
$ cat verweis
Dateiinhalt
$ ls
datei  verweis
$ rm datei
$ ls
verweis
$ cat verweis
Dateiinhalt
```

Sybolische Links

`ln -s quelle linkname`

- Zeiger auf Dateien oder Verzeichnisse.
- Löschen, Umbenennen und Verschieben der Quelldatei brechen den Link.
- Symbolische Links über Gerätegrenzen möglich.
- Anzeigen mittels `ls -F`

Beispiel: Symbolische Links

```
$ echo "Dateiinhalt">datei
$ ln -s datei verweis
$ ls -F
datei  verweis@
$ cat verweis
Dateiinhalt
$ rm datei
$ cat verweis
cat: verweis: No such file or directory
$ ls
verweis
$ ls -F
verweis@
```

©opyleft

©opyleft

- Erstellt mit Freier Software
- © Rüdiger Weis, Berlin 2005 – 20012
- unter der GNU Free Documentation License.