Betriebssysteme

Seminar 1

Inhalt

- Organisatorische Aspekte
- Shell-Skript
- Befehlszeile
- UNIX-Befehle
- Liste der Befehle
- Zusammengesetzte Befehle
- Umleiten
- UNIX-Kontrollstrukturen
- Aufgaben

Organisatorische Aspekte

- **Seminaraktivität:** 2 Stunden jede 2 Wochen
- **Mindestens 75%** aktive Teilnahme an den Seminarstunden (**mindestens 5** von 7 Seminarstunden)
- Während der Seminarstunden finden über die Moodle-Plattform Quizze statt

Shell-Skript

- = ausführbare ASCII Textdatei, die eine Folge von UNIX-Befehle enthält. Jede Anweisung aus der Datei kann von der Benutzer auch in der Befehlszeile benutzt werden.
- können die Dateiendung .sh haben
- Warum Shell-Skripting?
    - Der Hauptnutzen einer Shell ergibt sich aus der Tatsache, dass es sich um eine **Kombination bereits vorhandener Befehle** handelt. Dies bedeutet die Möglichkeit der **Automatisierung**. Wiederholte Aufgaben können in ein Shell-Skript geschrieben werden. Wann immer es notwendig ist, diese Aufgabe auszuführen, wird das Shell-Skript ausgeführt und diese wird erfüllt.

Shell-Skript

- in der ersten Zeile kann der Befehlsinterpreter angegeben werden

#!/bin/sh oder **#!/bin/bash**

- **#** - gibt den Anfang eines Kommentars an
- Das **#!** am Kopf des Skripts teilt dem System mit, dass diese Datei eine Reihe von Befehlen ist, die dem angegebenen Befehlsinterpreter zugeführt werden sollen.
- **/bin/sh** ist eine ausführbare Datei, die die Systemshell darstellt. Die System-Shell ist im Grunde die Standard-Shell, die das Skript verwenden sollte.
- **/bin/bash** ist die am häufigsten verwendete Shell als Standard-Shell für die Benutzeranmeldung des Linux-Systems.
- **#!/bin/sh**: Führt das Skript mit der Bourne-Shell oder einer kompatiblen Shell mit path/bin/sh aus
- **#!/bin/bash:** Führt das Skript mithilfe der Bash-Shell aus.

Shell-Skript

- Ausführung des Shell-Skripts:
  - ./name.sh
- Ausführung abgeschlossen:
  - natürlich
  - Gezwungen: **CTRL + C**

- Einfaches Beispiel: *Schreiben Sie ein Shell-Skript, das "Hallo, Welt" anzeigt*.

```
#!/bin/sh
echo “Hallo, Welt”
exit 0
```

- wir verwenden **echo**, um eine Zeichenfolge bei Standardausgabe anzuzeigen.
- **exit** wird verwendet, damit das Programm erfolgreich beendet wird
- die Verwendung des Befehls **exit** ist **optional**. Ein Shell-Skript gibt standardmäßig 0 zurück, wenn es das Ende des Programms erreicht.
- **exit 1:** Allgemeine Fehler, verschiedene Fehler wie "durch Null teilen" und andere unzulässige Operationen
- **exit 2:** Missbrauch von Shell-Builtins (laut Bash-Dokumentation)
- **echo $?** Befehl wird verwendet, um den letzten Rückgabestatus anzuzeigen.

Befehlszeile

= Kommandozeile (command-line oder command prompt)

Befehl arg1 arg2 … argn

$0 $1 $2 $n

- **$0** - speichert das erste Wort des eingegebenen Befehls (den Namen des Shell-Programms)
- **$#** - speichert die Anzahl der Befehlszeilenargumente, die an das Shell-Programm übergeben wurden
- **$?** - speichert den Exit-Wert des zuletzt ausgeführten Befehls
- **$*** - speichert alle Argumente, die in der Befehlszeile eingegeben wurden ($1 $2 ...).
- **$@** - speichert alle Argumente, die in der Befehlszeile eingegeben wurden, einzeln in Anführungszeichen ("$ 1" "$ 2" ...).

Beispiel

./command -yes -no /home/username

$# = 3

$* = -yes -no /home/username

$@ = array: {"-yes", "-no", "/home/username"}

$0 = ./command, $1 = -yes etc.

UNIX-Befehle

- **nützliche Befehle in der Shell- Kontext:** shift, read, readonly, sleep, exit, echo, test (äquivalent mit "[..]"), export, expr, basename, (umgekehrte Apostrophe)
- **Befehle zum Arbeiten mit Ordner:** ls, pwd, cat, find, locate, file, more, less, rm, mkdir, rmdir, cp, mv, cd, chmod, chown, ln, touch, du, cut, sort, uniq, cmp, diff, head, tail, wc, split
- **Netzwerkbefehle:** netstat, ping, hostname, host, ftp, ftpwho
- **Andere Befehle:** clear, date, mail, uptime, df, fg, bg

UNIX-Befehle

[*int1* -eq *int2*] *# int1 = int2*

[*int1* -ge *int2*] *# int1 >= int2*

[*int1* -gt *int2*] *# int1 > int2*

[*int1* -le *int2*] *# int1 <= int2*

[*int1* -lt *int2*] *# int1 < int2*

[*int1* -ne *int2*] *# int1 != int2*

UNIX-Befehle

[*file1* -nt *file2*]	*# True, wenn Datei1 neuer ist als (je nach Änderungszeit) Datei2*
[*file1* -ot *file2*]	*# True, wenn Datei1 älter als Datei2 ist*
[-d *file*]	*# True, wenn die Datei ein Verzeichnis ist*
[-e *file*]	*# True, wenn eine Datei vorhanden ist*
[-f *file*]	*# True, wenn eine Datei vorhanden ist und eine reguläre Datei ist*
[-L *file*]	*# True, wenn die Datei ein symbolischer Link ist*
[-r *file*]	*# True, wenn file eine von Ihnen lesbare Datei ist*
[-w *file*]	*# True, wenn file eine von Ihnen schreibbare Datei ist*
[-x *file*]	*# True, wenn file eine von Ihnen ausführbare Datei ist*

UNIX-Befehle

[-z *string*]	*# True, wenn die Zeichenfolge leer ist.*
[-n *string*]	*# True, wenn die Zeichenfolge nicht leer ist.*
[*string1* = *string2*]	*# True, wenn string1 gleich string2 ist.*
[*string1* != *string2*]	*# True, wenn string1 nicht gleich string2 ist.*
[!*E*]	*# Negierung des Ausdrucks E*
[*E1* -a *E2*]	*# AND-Zusammensetzung der E1- und E2-Ausdrücke*
[*E1* -o *E2*]	*# OR-Zusammensetzung der E1- und E2-Ausdrücke*

Liste der Befehle

= sind Befehlszeilen, die durch Pipes (“|”) verbunden sind und von einem der Operatoren getrennt sind

- **&&** (c1 && c2)
  - **AND** - c2 wird nur ausgeführt, wenn c1 erfolgreich beendet wurde (gibt einen Exit-Status von Null zurück)
- **||** (c1 || c2)
  - **OR** - c2 wird nur ausgeführt, wenn c1 fehlgeschlagen ist (Gibt einen Exit-Status ungleich Null zurück)
- **;** (c1;c2;c3)
  - c1, c2 und c3 werden nacheinander ausgeführt. Der Rückkehrcode dieser Sequenz ist der des letzten Befehls
- **&** (c1 & c2)
  - c1 wird im Hintergrund und c2 im Vordergrund in einer Subshell ausgeführt

Zusammengesetzte Befehle

- **(***Liste_der_Befehle***)** = *Liste_der_Befehle* wird in einem Subshell ausgeführt, dass heißt die verwendeten Variablen und die Befehle in der Liste, die normalerweise die Shell-Umgebung beeinflussen würden, werden nach Abschluss dieser Sequenz nicht mehr wirksam
  - Eine Subshell ist ein untergeordneter Prozess, der von einer Shell gestartet wird
  - Ein Shell-Skript kann selbst Unterprozesse starten. Mit diesen Subshells kann das Skript parallel verarbeitet werden, sodass mehrere Unteraufgaben gleichzeitig ausgeführt werden.
- **{***Liste_der_Befehle***}** = Eine solche Sequenz wird als Gruppenbefehl bezeichnet. Ein Codeblock in geschwungenen Klammern startet keine Subshell.
- **(**Ausdruck**)** = gibt den Wert des Ausdrucks zurück
- **!** Ausdruck **=** negiert den Wahrheitswert des Ausdrucks

Zusammengesetzte Befehle

- **((**Ausdruck**))** = Ausdruck wird arithmetisch ausgewertet. Der Rückkehrcode dieser Sequenz ist 0, wenn der ausgewertete Ausdruck ungleich Null ist, andernfalls 1.

```
#!/bin/bash
((sum=25+35))
echo $sum
```

- **[[** Ausdruck **]] =** Ausdruck ist eine Bedingung, die ausgewertet wird.

```
#!/bin/bash
echo "Geben Sie eine beliebige Nummer ein "
read n
if [[ ( $n -eq 15 || $n  -eq 45 ) ]]
then
echo "Du hast das Spiel gewonnen"
else
echo "Du hast das Spiel verloren"
fi
```

Umleiten

- Standardausgabe “>” “>>”

  ls > datei

- Standardeingabe "<"

  cat < datei

- Verbindung durch pipe “|” (p1 | p2 - Ausgang von p1 ist Eingang für p2)

  ls | cat

UNIX-Kontrollstrukturen **IF**-Struktur

if *Liste_der_Befehle_1*

then *Liste_der_Befehle_2*

elif *Liste_der_Befehle_3*

then *Liste_der_Befehle_4*

...

elif *Liste_der_Befehle_n*

then *Liste_der_Befehle_n+1*

else *Liste_der_Befehle_n+2*

fi

Aufgabe

```
if [ $# -lt 2 ]
    then
      echo "Sie müssen mindestens zwei Parameter eingeben!"
      exit 1
fi
```

UNIX-Kontrollstrukturen **CASE**-Struktur

case *wort* **in**

*layout_1***)** *Liste_der_Befehle_1***;;**

...

*layout_n***)** *Liste_der_Befehle_n***;;**

esac

Das Layout besteht aus einer Reihe allgemeiner Spezifikationen, die für Dateien verwendet werden (? - ein beliebiges Zeichen, * - eine beliebige Zeichenfolge, [...] - ein beliebiges angegebenes Zeichen), getrennt durch |

Aufgabe

```
case $1 in
    [a-z]|[A-Z]) echo "letter";;
    [0-9]) echo "digit";;
    *) echo "no letter, nor digit";;
esac
```

UNIX-Kontrollstrukturen **FOR**-Struktur

for *name* **in** [*durch_Leerzeichen_getrennte_Liste_von_Wörtern*]

do

Liste_der_Befehle

done

Aufgabe

```
for fis in ‘ls ‘
do
    cat $fis
done
```

```
s=0;
for i in ‘seq 1 10’
do
    s=’expr $s + $i’
done
echo $S
```

UNIX-Kontrollstrukturen **FOR**-Struktur

for ((*expr1* ; *expr2* ; *expr3* **))**

do

Liste_der_Befehle

done

Aufgabe

```
factorial=1;
N=4;
for (( i=2; $i<=$N; i++ ))
do
    factorial=$(( $factorial * $i ))
done
echo $factorial
```

UNIX-Kontrollstrukturen **WHILE**-Struktur

while *Liste_der_Befehle_1*

do

Liste_der_Befehle_2

done

Aufgabe

```
factorial=1;
N=4;
i=2;
while [ $i -le $N ]
do
    factorial=$(( factorial * i ))
    i=’expr $i + 1’
done
echo $factorial
```

UNIX-Kontrollstrukturen **UNTIL**-Struktur

until *Liste_der_Befehle_1*

do

Liste_der_Befehle_2

done

Aufgabe

```
factorial=1;
N=4;
i=2;
until [ $i -gt $N ]
do
    factorial=$(( $factorial * $i ))
    i=’expr $i + 1’
done
echo $factorial
```

Aufgaben:

1) Zeige alle Studenten aus eurer Gruppe an.
2) Zeige für jeden von der Tastatur gelesenen Namen die vom Finger angegebenen Informationen an.

Lösung Aufgabe 1

```
#!/bin/sh
for u in ‘ls -1 /home/scs/licenta/an1/gr71_’
do
    echo $u
done
```

Lösung Aufgabe 2

```
#!/bin/bash
while read name
do
    finger $name
done < name_datei
```