8.Übung

Systemsoftware (SYS)

Christian Baun
cray@unix-ag.uni-kl.de

Hochschule Mannheim – Fakultät für Informatik
Institut für Robotik

23.11.2007

Wiederholung vom letzten Mal

- Textausgaben auf der Shell (`echo`, `printf`, `yes`, `seq`)
- Inhalt der Shell löschen (`clear`)
- Mustervergleiche (`sed`)
- Bearbeiten und Interpretieren von Texten (`awk`)

Heute

- Shell-Skripting (Teil 1)
  - Die Shell
  - Varianten der Shell
  - Kommentare
  - Auswahl der Shell
  - Shell-Skripte testen (`sh`)
  - Feste Variablen (`${var}`, `$0`, `$# $*`, `$@`, `$$`, `$-`, `$_`, `$?`, `&!`)
  - Kommandozeilenparameter verarbeiten
  - Tests für Zeichenketten, Zahlen und Dateien (`test`, `[`)
  - Rückgabewert setzen (`true`, `false`)
  - Shell-Skripte vorzeitig beenden (`exit`)

Die Shell

- Die Shell ist ein Programm, durch das das System die Anweisungen (Befehle) des Benutzers verstehen kann.
- Wegen Ihrer Funktion wird die Shell häufig als Befehls- oder Kommandointerpreter bezeichnet.
- Die Shell hat drei Hauptaufgaben:
  - Interaktive Anwendung (Dialog).
  - Anwendungsspezifische Anpassung des Systemverhaltens (Umgebungsvariablen definieren).
  - Programmierung (Shell-Skripting).
- Die Shell kennt einige Mechanismen, die aus Hochsprachen bekannt sind. Diese sind u.a. Variablen, Datenströme, Funktionen, ...

Einfache und komfortable Varianten der Shell

- Es gibt nicht *die* eine Shell, sondern es existieren mehrere Varianten.
- Jede Variante hat ihre Vor- und Nachteile.
- Es ist unter Linux/UNIX kein Problem den Kommandointerpreter auszutauschen. Aus diesem Grund stehen auf fast allen Systemen mehrere unterschiedliche Shells zur Verfügung.
- Welche Variante der Shell ein Benutzer verwenden möchte, ist reine Geschmackssache.
- Die existierenden Varianten der Shell können in eher einfache und eher komfortable Shells unterschieden werden

Einfache Varianten der Shell

- **Bourne**- oder **Standard-Shell** (`sh`):
  - Kompakteste und einfachste Form.
  - Bietet u.a. Umlenkung der Ein- oder Ausgaben, Wildcards zur Abkürzung von Dateinamen, Shell-Variablen, usw.
  - Steht auf praktisch allen Systemen zur Verfügung.
  - Shell-Skripte für die Standard-Shell sind sehr portabel.
- **Korn-Shell** (`ksh`):
  - Weiterentwicklung der Bourne-Shell.
  - Bietet u.a. History-Funktionen, eine Ganzzahl-Arithmetik und Aliase.
- **C-Shell** (`csh`):
  - Bietet ähnliche Features wie die Korn-Shell.
  - Syntax ist sehr stark an die Programmiersprache C angelehnt.
  - Geringe Portabilität. Darum eher ungeeignet für Shell-Skripte.

Komfortable Varianten der Shell

- **Bourne-Again-Shell** (`bash`):
  - Voll abwärtskompatibel zur Standard-Shell.
  - Bietet aber von allen Shells die komfortabelsten Funktionen für das interaktive Arbeiten.
  - Standard-Shell auf allen Linux-Systemen.
  - Steht auf den meisten anderen UNIX-Systemen zur Verfügung.
- **TENEX-C-Shell** (`tcsh`):
  - Verhält sich zur C-Shell wie die Bourne-Again-Shell zur Standard-Shell.
  - Voll kompatibel zur C-Shell. Bietet aber zusätzliche Komfort-Funktionen.
- Es existieren noch viele weitere Varianten der Shell.
- Eine Übersicht: `http://de.wikipedia.org/wiki/Unix-Shell`
- Für Shell-Skripte optimal: Standard-Shell oder Bourne-Again-Shell.

Warum schreibt man Shell-Skripte?

- Shell-Skripte sind immer da hilfreich, wo:
  - Ständig wiederkehrende Kommandos zusammengefasst werden sollen. Diese können dann mit einem einzelnen Aufruf gestartet werden.
  - Schnell kleine Programme entwickelt werden sollen.
  - Regelmäßige Systemüberwachung notwendig ist.
  - Umfangreiche Protokoll- und Servicedaten (Log-Daten) anfallen, die überwacht werden müssen.
  - Automatisierung notwendig ist um Fehler zu vermeiden und Ressourcen zu sparen.
- Typische Einsatzgebiete für Shell-Skripte sind Administrationsaufgaben (z.B. Backup).

Wie schreibt man Shell-Skripte?

- Um ein einfaches Shell-Skript zu erzeugen, startet man einen beliebigen Editor und führt ein paar Kommandos hintereinander zeilenweise auf.
- Ein einfaches Beispiel:

```
# Mein erstes Shell-Skript
echo "Test"
date
whoami
```

- Diese Zeilen werden unter dem Namen `shellskript` gespeichert.
- Die Datei muss noch ausführbar gemacht werden:

```
$ chmod u+x shellskript
$ ls -l shellskript
-rwxr--r-- 1 testuser testuser 51 2007-10-23 17:06 shellskript
```

Das erste Shell-Skript

- Ergebnis der Ausführung des ersten Shell-Skripts:

```
$ ./shellskript
Test
Di 23. Okt 17:09:40 CEST 2007
testuser
```

Die Shell auswählen

- In der ersten Zeile eines Shell-Skriptes sollte immer definiert werden, mit welcher Shell das Skript ausgeführt werden soll.
- In diesem Fall öffnet das System eine Subshell und führt das restliche Shell-Skript in dieser aus.
- Die Angabe der Shell erfolgt über eine Zeile in der Form:
  - Für die Standard-Shell

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

  - Für die Bourne-Again-Shell

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

- Der Eintrag wirkt nur, wenn er in der ersten Zeile des Shell-Skripts steht.

Kommentare

- Kommentare in der Shell beginnen immer mit dem Zeichen #.
- Es spielt keine Rolle, ob das Zeichen am Anfang der Zeile steht, oder hinter Kommandos.
- Alles ab dem #-Zeichen bis zum Zeilenende wird beim interpretieren von der Shell ignoriert.

```
# Das ist eine Kommentarzeile!
```

- Beim Schreiben von Shell-Skripten sollte man nicht mit Kommentaren geizen, um die Lesbarkeit zu erhöhen.
- Der Einzige Fall, in dem der Text hinter dem # nicht ignoriert wird, ist bei der Auswahl der Shell.

Shell-Skripte testen

- Zum Testen eines Shell-Skripts empfiehlt sich das Kommando `sh -x`.
- Beim Aufruf mit `sh -x` wird jedes Kommando im Shell-Skript ausgeführt und das Ergebnis direkt ausgegeben.

```
$ cat shellskript
#!/bin/bash
# Mein erstes Shell-Skript

echo "Test"
date
whoami
```

```
$ sh -x shellskript
+ echo Test
Test
+ date
Mi 31. Okt 10:28:46 CET 2007
+ whoami
bauni
```

Feste Variablen bei Shell-Skripten

`${var}`	Wert der Variablen `var`.
`$0`	Name der Programms (Shell-Skripts).
`$#`	Anzahl der Argumente auf der Kommandozeile.
`$1 $2 $2`	Erstes, zweites, drittes ... Argument.
`$*`	Alle Argumente auf der Kommandozeile (`$1 $2 $3` ...).
`$@`	Wie `$*`.
`"$@"`	Expandiert im Unterschied zu `$*` zu `"$1" "$2" "$3"` ....
`$$`	Prozessnummer (PID) der Shell.
`$-`	Die aktuellen Shell-Optionen.
`$_`	Name der Datei, für die diese Shell gestartet wurde.
`$?`	Rückgabewert (Return-Code) des zuletzt ausgeführten Kommandos. (Normalerweise `0` bei erfolgreicher Durchführung).
`&!`	Prozeßnummer des zuletzt gestarteten Prozesses.

Feste Variablen und Kommandozeilenparameter

```
$ cat variablen_skript
echo Anzahl der Übergabeparameter: $#
echo Übergabeparameter: $*
echo Benutzer ist: $USER
echo Shell ist eine: $SHELL
echo Erster Parameter: $1
echo Zweiter Parameter: $2
echo Dateiname des Shell-Skripts: $0
echo Prozessnummer \(PID\): $$

$ ./variablen_skript eins zwei drei
Anzahl der Übergabeparameter: 3
Übergabeparameter: eins zwei drei
Benutzer ist: testuser
Shell ist eine: /bin/bash
Erster Parameter: eins
Zweiter Parameter: zwei
Dateiname des Shell-Skripts: ./variablen_skript
Prozessnummer (PID): 9444
```

Vergleichsoperationen

- Das Kommando `test` ist Bestandteil der Shell und wertet einfache Boolesche Ausdrücke aus, die aus Zahlen und Strings bestehen kommen.
- Entsprechend der Auswertung von `test` ist der Rückgabewert (Exit-Status) 0 (true) oder 1 (false).
- Für `test` gibt es den Alias `[`. Wenn Sie diesen Alias verwenden, müssen Sie als letztes Argument von `test` ein `]` angeben.
- Die Vergleichsoperatoren müssen von Leerzeichen umgeben sein, sonst werden Sie von der Shell nicht erkannt. Das Gilt auch für die Klammern.

Ist 10 größer als 5?

```
$ test 10 -gt 5
$ echo $?
0
```

Ist 10 größer als 5?

```
$ [ 10 -gt 5 ]
$ echo $?
0
```

Vergleichsoperationen (Tests für Zeichenketten)

`"s1" == "s2"`	Wahr, wenn die Zeichenketten gleich sind.
`"s1" != "s2"`	Wahr, wenn die Zeichenketten ungleich sind.
`-z "s1"`	Wahr, wenn die Zeichenkette leer ist (Länge $= 0$).
`-n "s1"`	Wahr, wenn die Zeichenkette nicht leer ist (Länge > 0).

Sind die beiden Strings gleich?

```
$ test "TEST" == "TEST"
$ echo $?
0
```

Sind die beiden Strings ungleich?

```
$ test "String" != "TEST"
$ echo $?
0
```

Vergleichsoperationen (Tests für Ganze Zahlen)

`n1 -eq n2`	Wahr, wenn die Zahlen gleich sind.
`n1 -ne n2`	Wahr, wenn die Zahlen ungleich sind.
`n1 -gt n2`	Wahr, wenn `n1` $>$ `n2` sind.
`n1 -ge n2`	Wahr, wenn `n1` $\geq$ `n2` sind.
`n1 -lt n2`	Wahr, wenn `n1` $<$ `n2` sind.
`n1 -le n2`	Wahr, wenn `n1` $\leq$ `n2` sind.

Ist 15 kleiner als 10?

```
$ test 15 -lt 10
$ echo $?
1
```

Hat der String `Test` eine Länge $>$ 0

```
$ test -n "Test"
$ echo $?
0
```

Vergleichsoperationen (Tests für Dateien)

`-d Name`	Wahr, wenn `Name` ein Verzeichnis ist.
`-f Name`	Wahr, wenn `Name` eine reguläre Datei ist.
`-L Name`	Wahr, wenn `Name` ein symbolischer Link ist.
`-r Name`	Wahr, wenn `Name` existiert und lesbar ist.
`-w Name`	Wahr, wenn `Name` existiert und schreibbar ist.
`-x Name`	Wahr, wenn `Name` existiert und ausführbar ist.
`-s Name`	Wahr, wenn `Name` existiert und die Größe > 0 ist.
`f1 -nt f2`	Wahr, wenn `f1` jünger als `f2` ist.
`f1 -ot f2`	Wahr, wenn `f1` älter als `f2` ist.

Vergleichsoperationen (Sonstige Tests)

`!`	Negation.
`-a`	Logisches *und*.
`-o`	Logisches *oder*.
`\(...\)`	Gruppierung. Die Klammern müssen jeweils durch einen Backslash geschützt werden.

Wahr (`true`) und Falsch (`false`)

- In der Shell existieren die Kommandos `true` und `false`, die ihren Rückgabewert (Exit-Status) entsprechend setzen.

```
$ true
$ echo $?
0
```

```
$ false
$ echo $?
1
```

- Die Kommandos `true` und `false` sind besonders bei Bedingungen und Schleifen nützliche Werkzeuge.

Shell-Skripte vorzeitig beenden mit `exit`

- Shell-Skript beenden sich automatisch, sobald ihre letzte Zeile ausgeführt wurde.
- Es ist aber auch möglich, ein Shell-Skript vorzeitig selbst beenden zu lassen.
- Zum vorzeitigen Abbruch eines Shell-Skript existiert das Kommando `exit`.

```
exit
```

- `exit` kann eine ganze Zahl als Argument mitgegeben werden, um den Rückgabewert (Exit-Status) festzulegen.
  - `exit 0` bedeutet so viel wie *alles ok*.
  - `exit 1` bedeutet *Fehler!*.
- Der Rückgabewert kann später über die Variable `$?` ausgelesen werden.

Nächste Übung:

30.11.2007