Betriebssysteme

Labor 3

Shell-Programmierung

- **`Shell`** = ein spezielles Programm, das eine Schnittstelle zwischen dem Benutzer und dem Kern des Betriebssystems bereitstellt
- shell: **`sh`** (Bourne shell), **`csh`** (C shell), **`ksh`** (Korn shell), **`bash`** (GNU Bourne-again shell)
- **`Skript`** = eine Textdatei mit Befehlen (intern oder extern)

- **`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.

Shell-Programmierung

- ein erstes Beispiel für ein Bash-Skript:

```
#!/bin/bash

pwd
ls
```

- **Bemerkungen:**
  - der Ablauf `#!` aus der ersten Zeile des Skripts ist **KEIN** Kommentar (es heißt Shebang)
  - nach dieser Sequenz wird der absolute Pfad zum auszuführenden Programm für alle anderen Zeilen des Skripts festgelegt

Shell-Programmierung

- Ausführen eines Bash-Skripts:

```
chmod +x script_1.sh
./script_1.sh
```

- Ausführung abgeschlossen:
  - natürlich
  - gezwungen: **`CTRL + C`**
- Kommentare beginnen mit dem #-Zeichen
- Variablen:
  - der Variablenname kann Buchstaben, Zahlen und Unterstriche „ _” (underscore) enthalten
  - das erste Zeichen muss ein Buchstabe sein
  - reservierte Wörter können nicht als Variablennamen verwendet werden
  - Shell unterscheidet zwischen Klein- und Großschreibung (case-sensitive)

Shell-Programmierung

- Beispiele:

```
n=45
name=Ana
msg="Enter a number:"
```

- reservierte Wörter (`keywords`):

```
if then else elif fi
for while until do done
case in esac
```

- interne Befehle(`built-in commands`):
  - die Liste der internen Befehle anzeigen: help
  - Befehlinformationen anzeigen: help **command**
  - Beispiele: echo read printf test

Befehlszeile

= Kommandozeile (command-line oder command prompt)

```
Befehl arg1   arg2   …   argn
 $0     $1    $2    …    $n
```

- **$0**: Dateiname
- **$1,..., $9**: die in der Befehlszeile angegebenen Argumente
- **$#**: speichert die Anzahl der Befehlszeilenargumente, die an das Shell-Programm übergeben wurden
- **$***: speichert alle Argumente (Reihe der Argumenten)
- **$@**: speichert alle Argumente, einzeln in Anführungszeichen (`"$1" "$2" ...`)
- **$?**: speichert den Exit-Wert des zuletzt ausgeführten Befehls
- **$$**: `PID` des aktuellen Prozesses
- **$!**: `PID` des letzten im Hintergrund gestarteten Befehls

Beispiel

```
./command -yes -no /home/username
```

```
$# = 3
$* = -yes -no /home/username
$@ = array: {"-yes", "-no", "/home/username"}
$0 = ./command, $1 = -yes etc.
```

Arithmetische Ausdrücke mit ganzen Zahlen

- **Befehl `expr`**

`expr` Ausdruck

- wertet den Wert eines ganzzahligen arithmetischen Ausdrucks aus und zeigt ihn an der Standardausgabe an
- Operanden:
  - **`+ - \* / %:`** Summe, Differenz, Produkt, Quotient oder Modulo
  - **`= !=:`** Numerische Vergleiche
  - **`\> \>=:`** den Wert 1, wenn die Beziehung zwischen s und d wahr ist
  - **`\< \<=:`** ansonsten Wert 0
  - **`\( \):`** Klammern für die Definition der Teilausdrücken
  - **`S \| D:`** Wert S, wenn S weder NULL noch 0 ist, sonst Wert D
  - **`S \& D:`** der Wert S, wenn S und D weder NULL noch 0 sind, sonst 0
  - **`length S:`** Länge der Folge S
  - **`index S CHARS:`** Position des ersten Auftretens in S oder 0 (Nummerierung beginnt bei 1)
  - **`substr S P L:`** die Teilzeichenfolge, die in S an Position P beginnt und die Länge L hat

Arithmetische Ausdrücke mit ganzen Zahlen

- **Befehl `let`**
  - wertet den Wert eines ganzzahligen arithmetischen Ausdrucks aus und zeigt ihn an der Standardausgabe an
  - Operanden: `++ -- ! ~ ** * / % + - << >> <= >= < > == != & ^ | && ||`
- **Doppelte Klammern**
- **Befehl `test`**

  `test` *`Bedingung`* oder **`[`** *`Bedingung`* **`]`**
  - wertet Bedingung aus und gibt 0 zurück, wenn die Bedingung wahr ist, andernfalls einen NULL-Wert
  - ermöglicht: Ganzzahlvergleich, Zeichenkettenvergleich oder Optionen zur Dateiüberprüfung

Ganzzahlen vergleichen

- Vergleichsoperatoren: `-lt -le -eq -ne -ge -gt`

- Vergleich von Zeichenfolgen:
  - `-z sir:` überprüft, ob die Zeichenfolge die Länge 0 hat
  - `-n sir:` überprüft, ob die Zeichenfolge eine Länge ungleich Null hat
  - `s1 = s2:` überprüft, ob die beiden Zeichenfolgen gleich sind
  - `s1 != s2:` überprüft, ob die beiden Zeichenfolgen unterschiedlich sind

Ganzzahlen vergleichen

- Optionen zur Dateiprüfung:
  - `-e fisier:` überprüft, ob die Datei existiert
  - `-s fisier:` überprüft, ob die Datei existiert und eine Länge ungleich Null hat
  - `-r fisier:` überprüft, ob die Datei existiert und gelesen werden kann
  - `-w fisier:` überprüft, ob die Datei existiert und geschrieben werden kann
  - `-x fisier:` überprüft, ob die Datei existiert und ausführbar ist
  - `-f fisier:` überprüft, ob die Datei existiert und eine normale Datei ist
  - `-d fisier:` überprüft, ob die Datei existiert und ein Verzeichnis ist
  - `-L fisier:` überprüft, ob die Datei existiert und ein symbolischer Link ist
  - `-p fisier:` überprüfen Sie, ob die Datei existiert und eine Pipe ist
  - `-c fisier:` überprüft, ob die Datei existiert und eine Sonderzeichendatei ist
  - `-b fisier:` überprüft, ob die Datei existiert und eine spezielle Blockdatei ist

IF-Struktur

```
if condition
then
    statement(s) to be executed
elif condition
then
    statement(s) to be executed
elif condition; then
    statement(s) to be executed
else
    statement(s) to be executed
fi
```

Beispiel

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

FOR-Struktur

```
for var in list
do
    statement(s) to be executed
done
```

- Festlegen eines Musters für einen Dateinamen (*filename wildcards*)
  - *****: beliebiger String, auch leer (nicht der erste Punkt im Dateinamen)
  - **?**: einzelnes Zeichen (nicht der erste Punkt im Dateinamen)
  - **[abc]**: beliebiges Zeichen in der Zeichenliste
  - **[!abc]**: jedes Zeichen, der nicht in der Zeichenliste ist
  - Beispiel:
    - Anzeige aller Dateien, die mit einem Buchstaben beginnen und eine Erweiterung von genau 2 Zeichen haben: `ls [a-zA-Z]*.??`

Beispiel

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

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

Beispiel

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

WHILE-Struktur

```
while condition
do
    statement(s) to be executed
done
```

```
until condition
do
    statement(s) to be executed
done
```

Beispiel

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

Beispiel

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

CASE-Struktur

```
case var in
    pattern_1)
        statement(s) to be executed if patern_1 is matched;;
    pattern_2)
        statement(s) to be executed if patern_2 is matched;;
    ...
    *)
        default condition to be executed;;
esac
```

Beispiel

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

Weitere nützliche Befehle

- Befehl `cut`

  ```
  cut -d: -f 1 /etc/passwd
  cut -d ":" -f 1 /etc/passwd
  who | cut -d " " -f 1
  ```

- Befehl `find`

  ```
  find . -type f -name "*.sh"
  find /tmp -type d -empty
  ```

- Befehl `shift`

  `shift [n]` Verschiebung nach links mit `n` Positionen der Befehlszeilenargumenten

- Befehl `sleep`

  `sleep [n]` setzt die Ausführung des aktuellen Prozesses für `n` Sekunden aus

- Befehl `exit`

  `Exit [n]` Abschluss der Ausführung und Rückkehr zu dem Prozess, von dem aus sie gestartet wurde

Ressourcen

- Programare shell:

  `https://ryanstutorials.net/bash-scripting-tutorial/`