Übung 3 – Shell-Programmierung

Aufgabe 1: Mein erstes Skript

Schreibe ein Shell-Skript, welches den Inhalt des aktuellen Verzeichnisses, sortiert nach der i-node-Nummer, ausgibt:

1. Öffne die Datei `myls.sh` im vi mit `vi myls.sh`
2. Schreibe folgenden Inhalt in die Datei:

```
ls -i | sort -n
```

3. Speichere die Datei und verlasse den vi
4. Führe die Datei mit dem Befehl **bash** in der Shell aus:

```
bash myls.sh
```

Aufgabe 2: Variablen

Variablen werden bei der Shell-Programmierung per Konvention ausschließlich unter Verwendung von **Großbuchstaben** benannt. Variablen werden bei ihrer Definition (Wertzuweisung) im aktuellen Prozess angelegt. Um auf den Inhalt einer Variablen lesend zuzugreifen, muss das Symbol **$** verwendet werden. Beim Schreiben in die Variable erfolgt der Zugriff durch Verwendung des Variablennamen, ohne dem vorangestellten $.

a) Weise der Variablen MYVAR verschiedene Werte zu und gib diese aus.
Tipp: Ausgabe über den Befehl echo. Also zum Beispiel `echo MYVAR` , nachdem der Variable MYVAR ein Wert zugewiesen wurde. Experimentiere mit der Zuweisung von Zeichen, Wörtern und ganzen Sätzen (Strings bzw. Zeichenketten).

b) Was bewirkt die Ausführung der folgenden Befehle in der Konsole? Erkläre, was passiert.
Tipp: „`date`“ ist ein Programm, das auch ganz normal gestartet werden kann.

```
DATE=$(date)
echo $DATE
```

Aufgabe 3: Rechnen

Um arithmetische Operationen auszuführen, ist die Verwendung des `let`-Ausdrucks erforderlich. Beachte die Verwendung von $ in folgendem Konstrukt:

```
i=0
let i=$i+1
echo $i
```

a) Was wird ausgegeben?

b) Gib das Gleiche erneut ein, allerdings ohne dabei `let` zu verwenden.
Was wird nun ausgegeben?

Aufgabe 4: Bedingungen in der Shell

Gebe folgende Befehle in der Shell ein:

```
if true
then
echo wahr
fi
```

Was wird in der History (= Pfeil hoch) angezeigt?

Beachte: `;` (Semikolon) und Zeilenumbrüche sind identisch aber nicht optional!

Aufgabe 5: Bedingungen im Skript

a) Öffne die Datei `my.sh` im vi

b) Schreibe folgenden Inhalt in die Datei:

```
if true
then
      echo wahr
fi
```

c) Speichere die Datei und verlasse den vi

d) Führe das Skript mit folgendem Befehl aus:

```
bash my.sh
```

Aufgabe 6: Bool'sche Operatoren

Die Operatoren für logisches „und" und logisches „oder" lauten „&&" bzw. „||".

Gib die drei unten stehenden Ausdrücke ein und beobachte, was ausgegeben wird. Ist das Ergebnis überraschend? Warum verhält sich die Shell so?

1. `true && echo wahr`
2. `false && echo wahr`
3. `false || echo false`

Aufgabe 7: Vergleich von Zeichenketten

Mit den folgenden Operatoren lassen sich Zeichenketten vergleichen.

=	Gleichheit	**!=**	Ungleichheit
-z	ist leer („zero")	**-n**	ist nicht leer („not zero")

Gib die folgenden Befehle ein und experimentiere mit den weiteren Möglichkeiten:

```
[ a = b ] || echo false

[ -z "" ] && echo wahr
```

Beachte: Das Leerzeichen nach „[" und vor „]" muss zwingend da stehen.

Aufgabe 8: Vergleich von Zahlen

-eq	gleich („equal")	**-ne**	ungleich („not equal")
-lt	kleiner als („less than")	**-gt**	größer als („greater than")
-le	kleiner gleich („less or equal")	**-ge**	größer gleich („greater or equal")

Gebe den folgenden Befehl ein und experimentiere mit den weiteren Möglichkeiten:

```
[ 3 -gt 1 ] && echo wahr
```

Aufgabe 9: Zahlenvergleiche im Skript

a) Öffne die Datei my.sh im vi

b) Schreibe folgenden Inhalt in die Datei:

```
i=0
if [ $i -eq 0 ]
then
echo wahr
fi
```

c) Speichere die Datei, verlasse den vi und führe das Skript aus

Experimentiere mit den weiteren Möglichkeiten.

Aufgabe 10: Mein erstes „sinnvolles“ Programm

Einem Skript kann man Parameter übergeben. Auf diese wird im Skript zugegriffen durch $1 für den ersten Parameter, $2 für den zweiten usw. Weiterhin kann ein Skript auch interaktiv gestaltet werden. Hierzu bietet sich die Shell Funktion `read` an, die Eingaben von stdin liest. Im folgenden Beispiel wird `read` mit dem Parameter `i` verwendet, dies ist der Name der Variable, in welche `read` die eingelesene Zeile ablegt.

Programmiere das folgende Skript und führe es mit verschiedenen Pfaden als Parameter aus

a) Öffne die Datei mytool.sh im vi

b) Schreibe folgenden Inhalt in die Datei:

```
cd $1
echo "Funktion ($PWD):"
echo " 1) ls"
echo " 2) ls -l"
echo " 3) ls -lR"
echo -n "Eingabe: "
read i
case $i in
1) ls;;
2) ls -l;;
3) ls -lR;;
esac
```

c) Speichere die Datei und verlasse den vi

d) Führe das Skript einmal ohne und einmal mit Parameter aus:

```
bash mytool.sh
bash mytool.sh /etc
```

Aufgabe 11: Schleifen

Auch Schleifen sind möglich:

```
for i in 1 2 3 4 ; do echo $i; done
for i in 3 2 5 9 ; do echo $i; done
for i in H a l l o ; do echo $i; done
for i in H a l l o ; do echo -n $i; done; echo ""
```

a) Gib die obigen Zeilen ein. Was geschieht hier? Verhält sich das `for`-Konstrukt wie in der Programmiersprache Java?

b) Wozu dient der zweite echo-Befehl im letzten Beispiel?

c) Gib die folgenden Zeilen ein und beobachte das Resultat:

```
i=0
while [ $i -lt 10 ]; do let i=$i+1; echo "$i"; done
```

Aufgabe 12: Schleifen im Skript

a) Öffne die Datei `count.sh` im vi

b) Schreibe folgenden Inhalt in die Datei:

```
for i in 1 2 3 4
do
case $i in
1) echo eins;;
2) echo zwei;;
*) echo $i;;
esac
done
```

c) Führe das Skript aus. Was passiert bei „*)“?

Aufgabe 13: Schleifen und Pipes

Schleifen lassen sich auch mit Pipes kombinieren, etwa um wie im folgenden Beispiel den Ausgabedatenstrom eines Programms Zeile für Zeile weiterzuverarbeiten:

```
ls | while read i; do echo "Line: $i"; done
```

Weshalb funktioniert das obige Beispiel?

Wo steckt die Abbruchbedingung für die `while`-Schleife?

Tipp: `read` liefert einen Rückgabewert. Führe zum besseren Verständnis den unten angegebenen Code aus (`$?` enthält den Rückgabewert des zuletzt ausgeführten Befehls). Der Befehl `read` wartet auf eine Eingabe. Gib zunächst eine beliebige Zeile ein. Welchen Wert liefert `read` zurück? Führe die Befehle erneut aus, beende die Eingabe an `read` durch Eingabe von der Tastenkombination *Strg-D* ohne vorher etwas einzugeben. Was liefert `read` nun zurück?

```
read j
echo $?
```