Shell-Scripte:

Shell-Scripte sind so etwas ähnliches wie Batchdateien wie man sie unter Windows kennt. Also eine Datei in der untereinander Befehle aufgeschrieben sind. Diese Scripte kann man dann ähnlich wie Befehle direkt in der Shell ausführen. Dabei werden dann die Befehle, der Reihenfolge nach, abgearbeitet. Jedoch ist mit Shell-Script einiges mehr möglich als mit Batchdateien. Shell-Script ähnelt schon fast eine Programmiersprache.

Shell-Scripte werden folgendermaßen ausgeführt, hier hallo.sh:

bash ./hallo.sh
. ./hallo.sh
source ./hallo.sh
Oder das Script ausführbar machen und dann ausführen, dies sollte der Normalfall sein.

chmod +x ./hallo.sh; ./hallo.sh
In der Ersten Zeile eines Shell-Scripts sollte immer der Interpreter angegeben werden, da es nicht nur Shell sondern auch Perl, PHP, und andere Scripte gibt und das System wissen muß welcher Interpreter dieses Script ausführen soll. Wird der Interpreter jedoch nicht angegeben wird versucht dieses Script mit der Standard Shell „/bin/sh" auszuführen. Die Angabe des Interpreters Beginnt immer mit „#!" in der ersten Zeile. z.B.:

#!/bin/bash
Nun das Standard Hallo Welt Programm:

```
#!/bin/bash
clear
echo ""
echo "Hallo Welt!"
echo ""
```

Wird dieses Script ausgeführt, Löscht es den Bildschirminhalt, Stellt einer Leerzeile da, dann den Satz "Hallo Welt", und dann noch eine Leerzeile.

Benutzereingaben:

Um nun vom Benutzer (der der das Script ausführt) Informationen zu erfragen, gibt es den Befehl ***read***, durch den der Benutzer zur Eingabe aufgefordert wird. Hierbei wird die Eingabe in einer Variabel, die nach dem ***read*** Befehl angegeben wird, abgespeichert. z.B:

```
echo "Bitte geben Sie Ihren Namen an!"
echo -n "Name: "
read name
```

Nun kann der Eingegebene Name durch ***$name*** im Script weiter verwendet werden.

Bedingungsabfragen:

Die oft als Wenn, dann, sonst abfrage bezeichnete Funktion z.B.:

```
if [ $name = Marko ]
  then
    echo "Hallo Marko, wie geht's dir?"
  else
    echo "Dich kenn ich nicht."
fi
```

Wenn also der Inhalt der Variabel ***$name*** Marko ist, wird also "***Hallo Marko, wie geht's dir?***" ausgegeben. Wenn nicht dann wird "***Dich kenn ich nicht***" ausgegeben. Mit if kann mehr als nur Gleichheit überprüft werden:

```
if [ $i = "eins" ]  → Ist wahr wenn $i "eins" ist
if [ $i -eq 1 ]     → Ist wahr wenn $i 1 ist
if [ $i -lt 1 ]     → Ist wahr wenn $i kleiner als 1 ist
if [ $i -gt 1 ]     → Ist wahr wenn $i größer als 1 ist
if [ $i -le 1 ]     → Ist wahr wenn $i kleiner oder gleich 1 ist
if [ $i -ge 1 ]     → Ist wahr wenn $i größer oder gleich 1 ist
if [ $i -ne 1 ]     → Ist wahr wenn $i nicht 1 ist
if [ -z $i ]        → Ist wahr wenn $i leer ist
```

Returncode's

Weiterhin gibt es die Möglichkeit die Returncode's von Programmen abzufragen. Wenn ein Programm genau so abläuft, wie es geplant ist, ist der Returncode "0", sollte etwas unvorhergesehenes Passieren wird ein Programm eine Returncode größer als "0" ausgeben. z.B.:

```
grep root /etc/passwd >/dev/null
if [ $? -eq 0 ]
  then
    echo "Der Benutzer root ist schon vorhanden"
  else
    echo "Es gibt keinen Benutzer root"
fi
```

Hier wird die /etc/passwd nach der Zeichenkette "root" durchsucht, aber die Ausgabe nach /dev/null umgeleitet, Jedoch sollte grep einen Returncode von 0 zurückgeben wenn "root" gefunden worden wehre und einen größeren wenn "root" nicht gefunden wurde. Deshalb wird im nächsten Schritt der Returncode abgefragt und zwar mit $?. Wenn $? also gleich 0 ist, soll "Der Benutzer root ist schon vorhanden" ausgegeben werden. Sonnst soll "Es gibt keinen Benutzer root" ausgegeben werden.

Der Befehl test:

Desweiteren gibt es einen Befehl namens test mit dem sich einiges testen läßt: z.B.:

```
test DATEI1 -nt DATEI2   → Wahr wenn DATEI1 neuer als DATEI2 ist
test DATEI1 -ot DATEI2   → Wahr wenn DATEI1 älter als DATEI2 ist
```

```
test -d Verz1       → Wahr wenn das Verzeichnis Verz1 existiert
test -f Datei1      → Wahr wenn die Datei Datei1 existiert
test -r Datei1      → Wahr wenn die Datei Datei1 lesbar ist
test -w Datei1      → Wahr wenn die Datei Datei1 schreibbar ist
```

Weitere Optionen entnehmen Sie der ManPage.
Wenn test wahr ist, gibttest immer den Returncode 0 zurück.

Schleifen:

Die For-Schleife:

Eine for-Schleife die von 1 bis 10 Zählt.

```
for i in {1..10}; do
  echo $i
done
```

Die While-Schleife:

Eine while-Schleife die von 1 bis 10 zählt.

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

Hier wird eine Variabel i festgelegt deren Inhalt 1 ist, dann wird eine Schleife gestartet die solange laufen soll wie $i kleiner oder gleich 10 ist. Innerhalb der Schleife soll $i ausgegeben werden und dann soll $i plus 1 ausgeführt werden.

Die case Anweisung:

Ein Beispiel:

```
case $i in
     1)   echo "i ist 1"
          ;;
     2)   echo "i ist 2"
          ;;
     *)   echo "i ist was anderes"
          ;;
esac
```

Wenn $i 1 ist soll "i ist 1" ausgegeben werden, wenn $i 2 ist, soll "i ist 2" ausgegeben werden. Wenn $i irgendwas anderes als 1 oder 2 ist, soll "i ist was anderes" ausgegeben werden.

Übergabeparameter:

Es Ist auch möglich, bestimmte Parameter schon beim Starten an ein Shell-Scrit zu übergeben. z.B.:

./script.sh Param1 Param2 Param3

Diese Parameter können dann im Script verwendet werden.

$0	→ Der Name des Scriptes
$1	→ Der erste Parameter
...	
$9	→ Der neunte Parameter
${10}	→ Der zehnte Parameter
$$	→ Die PID des Shell-Scripts
$#	→ Anzahl der übergebenen Parameter
$@	→ Alle übergebenen Parameter in der Form: "$1" "$2" "$3"
$*	→ Alle übergebenen Parameter in der Form: "$1 $2 $3"

Der Shift Befehl:

Der shift Befehl verschiebt alle Übergabe Parameter un eine Stelle nach links (aus $2 wird $1, aus $3 wird $2 und $1 fällt weg)