Thema heute:

- Shell-Skripten

Shell-Skripten

- Ein Shell-Skript ist die Zusammenfassung von Kommandos in einer Textdatei.
- Diese Textdatei wird wie ein ablaufbares Programm behandelt.
- Es werden von einer Shell auch programmierähnliche Konstrukte, wie Schleifen, Bedingungen, Parameter und Variablen zur Verfügung gestellt.

Ein Shell-Skript wird mit einem normalen ASCII-Editor (z.B. `emacs`) erstellt. Das Skript *muss* in der ersten Zeile den Hinweis auf die verwendete Shell in folgender Form enthalten:

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

Dabei teilen die beiden ersten Zeichen dem Betriebssystem mit, dass es sich um ein Shell-Skript handelt, und der folgende Pfad inklusive Shell teilen dem System mit, für welche Shell das Skript erstellt wurde.

Kennt man den Pfad zur Shell nicht, so kann man zwei Kommandos ausprobieren, die einem den Ort der Shell im Dateibaum anzeigen:

```
type <Shell-Name>
```

oder

```
locate <Shell-Name>
```

Beide Befehle lassen sich im Übrigen auch auf alle anderen ausführbaren Programme. die auf einem System installiert sind, anwenden (dennoch muessen beide Befehle nicht zwingend auf einem System vorhanden sein)!

Da Unix unter den Zugriffsrechten auch ein Ausführungsrecht besitzt, das normalerweise beim Anlegen von Textdateien (nichts anderes ist ein erstelltes Shell-Skript zunächst) nicht automatisch gesetzt wird, muss dieses Recht erst noch aktiviert werden:

```
chmod u+x <Shell-Skript>
```

Einfaches Beispiel

```
#!/bin/sh

echo Ich bin ein einfaches Shell-Skript!

exit 0
```

Die eingefügten Leerzeilen sind nicht verpflichtend. Ebenso kann die letzte Anweisung weggelassen werden. Bei komplexen Skripten kann der `exit`-Status aber nützlich sein, um Informationen darüber zu erhalten, ob das Skript korrekt abgearbeitet wurde.

Kommandos werden entweder durch Zeilenwechsel oder durch Semikolon getrennt.

Parameterübergabe beim Aufruf

Man kann einem Skript beim Aufruf (beliebig) viele Parameter übergeben, die innerhalb des Skriptes verarbeitet werden können.

```
skript [p1] [p2] ... [pn]
```

Diese Parameter werden als Text interpretiert, der innerhals des Skriptes in den Variablen

```
$1, $2, ... $9
```

gespeichert wird. Auch hier ist das Dollarzeichen $ wieder die Dereferenzierung auf den Inhalt einer Variablen (wie auch bei den Umgebungsvariablen).

Das Shell-Skript selbst kann nur die Parameter 1-9 ansprechen.

Das Kommando `shift`

Mittels der Angabe von

```
shift n
```

Kann der Zugriffsbereich innerhalb des Skriptes um den Betrag `n` vershoben werden, sodass auch Parameter jenseits von 9 angesprochen werden können. Der Parameter `$1` entspricht dann dem `n+1`-ten beim Aufruf angegebenen Parameter.

Beispiel

```
#!bin/sh

echo $2 $1
shift 9
echo $1

exit 0
```

Dabei spielt es keine Rolle bzw. es führt zu keinem Fehler, wenn mehr oder weniger Parameter übergeben werden als innerhalb des Skriptes verarbeitet werden.

Umdefinieren von Parametern innerhalb des Skriptes

Mit dem Kommando

```
set Parameter1 Parameter2 ...
```

können die übergebenen Parameter neu definiert werden. Dabei entspricht die Reihenfolge nach dem `set`-Kommando der Reihenfolge bei der Übergabe.

Beispiel

```
#!/bin/sh

echo $1 $2
set Hallo Welt!
echo $1 $2

exit 0
```

Besondere Parameter innerhalb des Skriptes

`$0` Name des Skriptes

`$#` Anzahl der übergebenen Parameter

`$*` Die übergebenen Parameter werden als eine Zeichenkette interpretiert.

`$$` PID-Nummer des aktuellen Prozesses.

Variablen in Shell-Skripten

Durch den Zuweisungsoperator = können auch Variablen definiert und mit Werten besetzt werden:

```
#!/bin/sh

variable="Hallo Welt!"
echo $variable

exit 0
```

Wichtig sind die Anführungszeichen, da sonst durch das Leerzeichen `Welt!` als nicht zu `Hallo` gehörend betrachtet würde. Eine Fehlermeldung wäre die Folge.

Variablen mit der Ausgabe eines Kommandos besetzen

Soll beispielsweise eine Variable `Datum` mit dem aktuellen Datum des Systems besetzt werden, so kann `Datum` die Ausgabe des Kommandos `date` zugewiesen werden. Dazu muss das auszuführende Kommando innerhalb der Shell in sogenannte Backticks gesetzt werden:

```
Datum=‘date‘
```

For-Schleife (1)

Um den Teil eines Skriptes mehr als einmal, abhängig von einer gewissen Bedingung, durchlaufen zu können, gibt es die `for`-Anweisung:

```
for <variable>
do
        Kommandos ...
done
```

Hierbei wird die Schleife sooft durchlaufen, wie Argumente beim Aufruf des Skriptes übergeben wurden. Mit jedem Durchlauf wird die `variable` auf den Wert des nächstes Argumentes gesetzt.

Beispiel

```
#!/bin/sh

for parameter
do
        echo $parameter
done

exit 0
```

For-Schleife (2)

Man kann auch ein konkretes Ziel angeben, mit dem die `variable` in der Schleife verglichen bzw. gesetzt werden soll:

```
for <variable> in <argumente>
do
        Kommandos ...
done
```

Beispiel

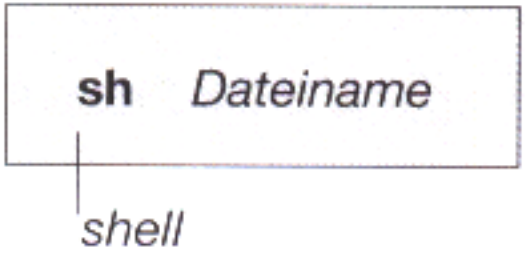
```
#!/bin/sh

for parameter in 9 8 7 6 5 4 3 2 1
do
        echo $parameter
done

exit 0
```

Was sind Shell-Prozeduren?

Shell-Prozeduren sind im Grunde nichts anderes als Kommandofolgen, abgelegt in einer Datei. Kommandos, die Sie bisher am Terminal aufgerufen haben, schreiben Sie in eine Datei. Mit dem Kommando **sh** können Sie die Kommandofolgen in dieser Datei von Ihrer Shell ausführen lassen.



sh – Kommando, um Dateien mit Kommandofolgen auszuführen

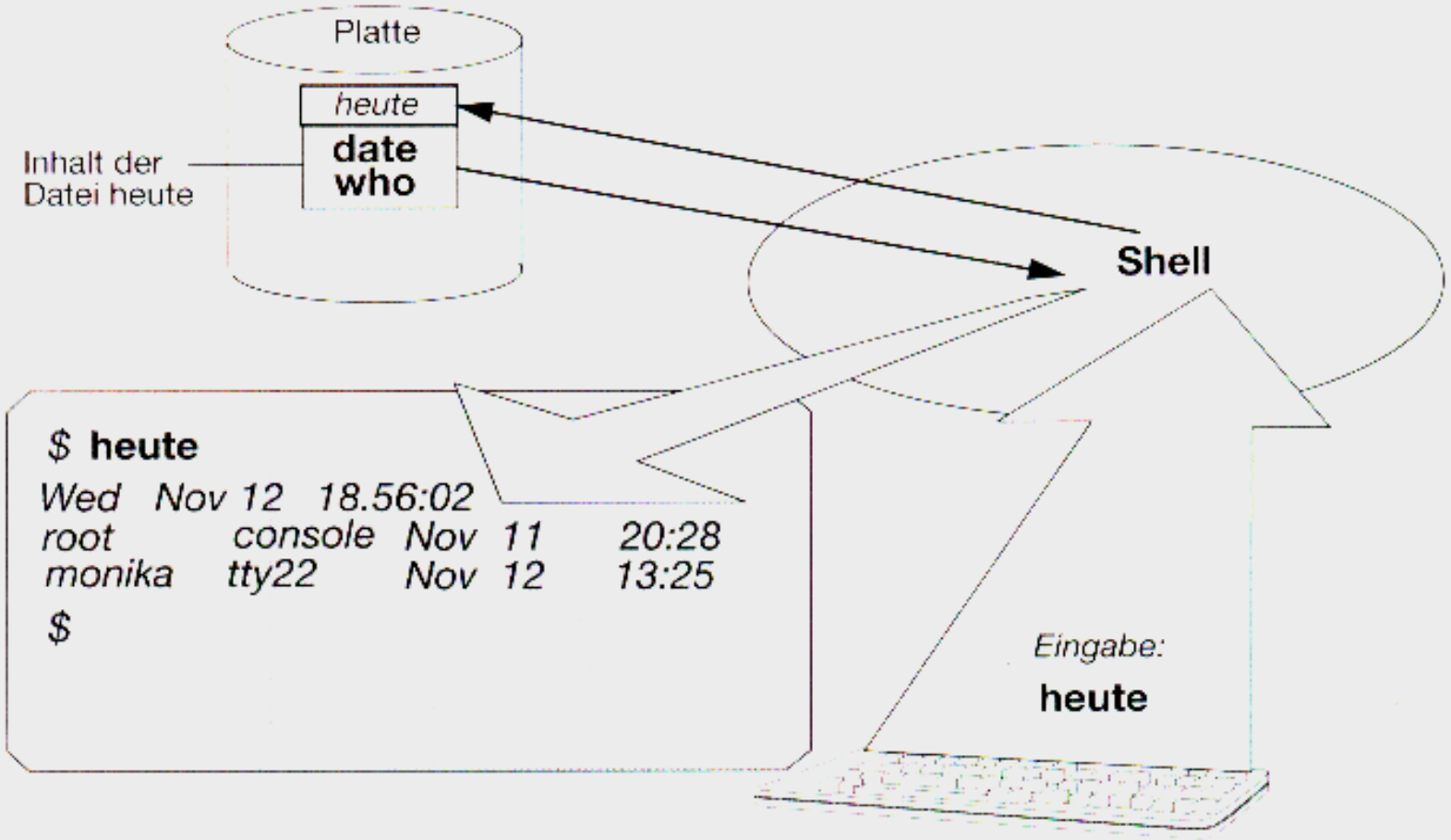


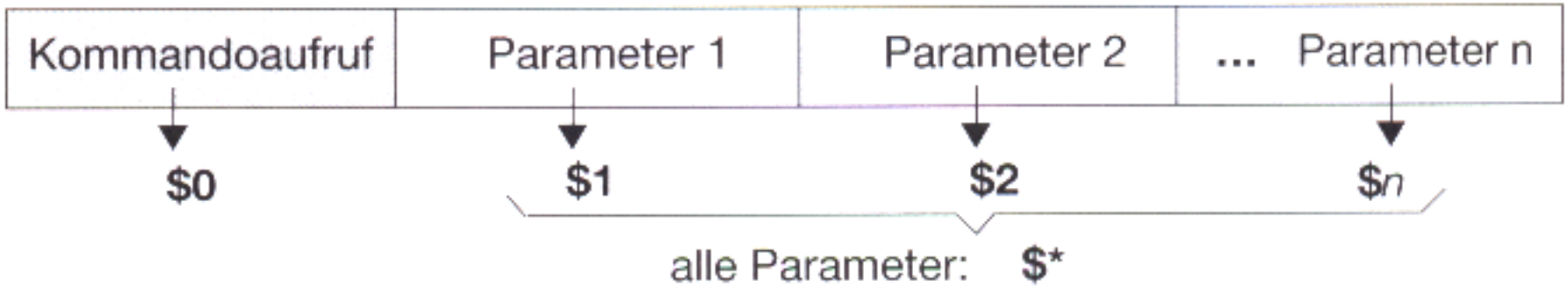
Bild 3-160: Ablauf einer Shell-Prozedur grafisch dargestellt

Alle Kommandos, also Programme, Shell-interne Kommandos und Shell-Prozeduren sind gleichberechtigt.

Aufruf des Kommandos	Kommandoname [Parameter 1 *(z.B. Option -l)*] \ [Parameter 2] ... im Vorder- oder Hintergrund (**&**) Die Parameter können hierbei über Metazeichen (*, ?, \, [ ]) und die Ihnen bereits bekannten Ersetzungsmechanismen (" ", ´ ´ , $, `*Kommando*`) expandiert werden.
Abbruch des Kommandos	(Unterbrechung des laufenden Programms): bei Vordergrundprozessen je nach Rechner entsprechende Funktionstasten z.B. **<CTRL> + c** oder bei Hintergrundprozessen mit dem Kommando ***kill -9*** *Prozeßnummer* Hier werden Sie in der Korn-Shell noch weitere Möglichkeiten kennenlernen
Umleitung	der Standardeingabe und Standardausgabe: **<, >, >>, 2>**
Verkettung	von Kommandos mit **;**
Pipe-Mechanismus	mit |

Eingabemöglichkeiten von Kommandos

Innerhalb einer Shell-Prozedur stehen Ihnen weitere Shell-Variable zur Verfügung. Die Shell analysiert Ihren Kommandoaufruf und erkennt, getrennt durch ein oder mehrere Leerzeichen, folgende Teile:



Die einzelnen Teile eines Kommandoaufrufs werden je nach Position des Kommandoteils der ***Variablen 0, 1, 2 bis 9*** zugewiesen. Den Namen des Kommandoaufrufs können Sie innerhalb Ihrer Shell-Prozedur mit ***$0*** erhalten, den Parameter 1 mit der Variablen ***$1***. Alle Namen der Parameter werden durch die Eingabe von ***$**** ersetzt. Zusätzlich können die Anzahl der Parameter oder die Prozeßnummer abgefragt werden. Diese Variablen werden auch **Positionsparameter** genannt. In der nachstehenden Aufstellung sind sie zusammengefaßt:

Positionsparameter

Positions-parameter	Bedeutung
$0	Name der Shell-Prozedur
$1 **$2** **...** **$9**	Wert des 1. Parameters Wert des 2. Parameters ... Wert des 9. Parameters
$*	Werte aller angegebenen Parameter
$#	Anzahl der Parameter
$?	Exit-Status des letzten Kommandos
$$	Prozeßnummer der Shell-Prozedur

Übersichtstabelle Positionsparameter

Die einfachste Form, den Ablauf von Kommandos zu steuern, ist das nächste Kommando nur dann zu starten, wenn das vorherige **erfolgreich** war. Hierfür werden die Kommandos mit **&&** verknüpft.

Kommando1 **&&** *Kommando2*

&& – Zeichen für bedingte Ausführung von Kommandos:
nur wenn das vorherige erfolgreich war,
wird das nachfolgende ausgeführt

Kommando1 || *Kommando2*

|| – Zeichen für bedingte Ausführung von Kommandos:
nur wenn das vorherige nicht erfolgreich war,
wird das nachfolgende ausgeführt

Mehr Möglichkeiten, in Abhängigkeit von dem Erfolg oder Nichterfolg eines Kommandos weitere Kommandos zu steuern, bietet die **if-Bedingung**.

```
if  Kommando(Bedingung)  ____________  wenn das und das zutrifft
    then  Kommandofolge1  _________  dann tue ....
  [ else  Kommandofolge2  ]  _____  [sonst tue ...]
fi  ____________________________  fertig (Ende der Abfrage)
```

if then else fi – internes Shell-Kommando, um den Ablauf zu steuern

Mit **fi** wird die if-Abfrage beendet *(es ist die Umkehrung von if)*. Die Worte **if, then, fi müssen vorhanden** sein und jeweils in einer **eigenen Zeile** stehen. Schreiben wir unsere Forderung im Programmierstil:

```
if test -f $Antwort  ____________  Wenn die angegebene Datei ($Antwort)
                                   existiert und es sich um eine nor-
                                   male Datei handelt (test -f)
  then pr -n $Antwort | pg  _____  dann soll sie am Bildschirm angezeigt
                                   werden (pg)
  else echo "$Antwort ist keine Datei"  sonst soll eine entsprechende Nachricht
  exit                             ausgegeben und die Prozedur be-
                                   endet werden (echo; exit)
fi  ____________________________  Ende (fi) Ende der Abfrage
```

```
if Kommando (Bedingung)
   then  if  Kommando (Bedingung)
             then  Kommandofolge
             (evtl. weitere  Kommandofolgen)
              if  Kommando (Bedingung)
                 then  Kommandofolge

              fi

          fi
   (weitere Kommandofolgen)
fi
```

Verschachtelung von if-Bedingungen

Bei Verschachtelungen kann man leicht den Überblick verlieren. Es empfiehlt sich deshalb, bei Prozeduren mit verschachtelten *if*-Bedingungen jeweils einzurücken, um die Zusammenhänge sichtbar hervorzuheben. Wird eine weitere **if-Bedingung in Verbindung mit *else*** eingegeben, so kann sie als ***elif*** zusammengezogen werden. ›***elif***‹ verlangt ebenso ein ***then*** wie ***if***. Allerdings benötigt ***elif* keinen eigenen Abschluß mit *fi*.**

Eingabe mit **else if**	alternativ **elif**
if *Kommando (Bedingung)* **then** *Kommandofolge* **else if** *Kommando (Bedingung)* **then** *Kommandofolge* **fi**	**if** *Kommando (Bedingung)* **then** *Kommandofolge* **elif** *Kommando (Bedingung)* **then** *Kommandofolge* keine Eingabe
fi	**fi**

Alternative Eingabe mit ›elif‹ statt ›else if‹

Schleifenverarbeitung

Sobald eine (oder mehrere) Anweisungen nicht nur einmal durchgeführt werden sollen, sondern mehrmals, benötigt man eine Schleife, so etwa, wenn eine Folge von Kommandos so oft durchlaufen werden soll, wie Parameter mitgegeben wurden.

Eine Schleife besteht aus drei Teilen:

- dem Schleifenkopf (Schleifenaufrufe: ***for, while, until***)
- dem Schleifenrumpf, dies sind die mehrmals auszuführenden Kommandos
- der Schleifenklammerung (***do .. done***).

Um eine Schleife, eine mehrmalige Wiederholung von Kommandos, einzuleiten, gibt es unterschiedliche Aufrufe, je nachdem in welcher Abhängigkeit die Wiederholung der Kommandofolgen steht. Die nachstehende Übersicht faßt die Möglichkeiten zusammen:

Kommando zur Einleitung der Schleife	Wiederholung der Kommandofolge abhängig von:
for *Name* **do** *Kommandoliste ....* **done**	der Anzahl der beim Aufruf der Shell-Prozedur übergebenen Parameter. Wurden z.B. 3 Parameter übergeben, wird die angegebene Kommandoliste von *do* bis *done* dreimal durchlaufen.
for *Name* **in** *Wort1 .. Wortn* **do** *Kommandoliste ....* **done**	der Anzahl der angegebenen Wörter. Geben Sie z.B. ein *for i* in otto hugo bernd iris wird die angegebene Kommandoliste viermal durchlaufen.
while *Kommando* **do** *Kommandoliste ....* **done**	dem Ergebnis des Kommandos nach *while*. Solange die Kommandofolge erfolgreich ist *(Exit-Status 0)*, wird die Kommandoliste durchlaufen. Z.B. solange die Variable Antw den Wert ja hat, wird die Kommandoliste zwischen *do .. done* wiederholt.
until *Kommando* **do** *Kommandoliste ....* **done**	dem Ergebnis des Kommandos nach *until*. Solange die Kommandofolge nicht erfolgreich ist *(Exit-Status ungleich 0)* wird die Kommandoliste zwischen *do* und *done* durchlaufen. Es handelt sich um die Umkehrung der *while*-Schleife.

Weitere nützliche Kommandos für Shell-Prozeduren

Die Schleifen-Kommandos ***while*** oder ***until*** werden auch gerne zu Testzwekken verwendet. Um eine ›*Endlosschleife*‹ zu starten, können Sie zwei Kommandos mit verwenden, deren Funktion nichts anderes ist, als sich ›**erfolgreich**‹ oder ›**nicht erfolgreich**‹ zurückzumelden.

Das Kommando **true** *(wahr)* liefert immer den Exit-Status 0, ist also immer erfolgreich, das Kommando **false** *(unwahr, falsch)* einen Exit-Status ungleich 0, immer nicht erfolgreich:

true

wahr – Exit-Status immer 0

true – Kommando, das nur den Exit-Status ›0‹ liefert

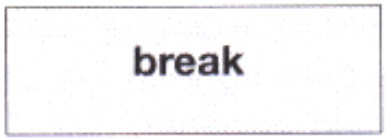
false

unwahr – Exit-Status immer ungleich 0

false – Kommando, das nur den Exit-Status ›ungleich 0‹ liefert

Wie können Sie eine Schleife abbrechen?

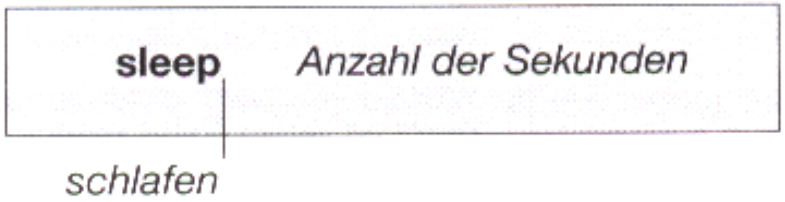
Sie können in der Schleife abfragen, ob ein bestimmer Zustand erreicht ist, ob eine Variable den Wert xy enthält, und mit dem Kommando break die Schleifenverarbeitung abbrechen.



break – Kommando, um Schleifen vorzeitig abzubrechen

Meistens wird bei Endlos-Tests irgendeine Nachricht auf dem Bildschirm ausgegeben, allerdings nicht fortwährend, sondern z.B. in einem Abstand von einer Minute. Zwischenzeitlich soll der Rechner ›schlafen‹.

Das Kommando, um ›Ruhepausen‹ in einer Shell-Prozedur einzulegen, nennt sich:



sleep – Kommando, um Wartezeiten in Sekunden zu setzen