Die Programmierung der bash - Shell

Michael Kugler

Unter DOS gibt es die Datei **command.com**, die die Eingaben der Tastatur bzw. von bat-Dateien verarbeitet. Unter Linux gibt es verschiedene Kommandozeilenprogramme, die man *Shell* nennt. Shell-Programme sind einfache Textdateien mit einigen Linux- und/oder **bash**-Kommandos. Nach dem Start eines derartigen Programmes werden die Kommandos der Reihe nach ausgeführt. Genau so wie bei DOS können Parameter übergeben werden und vom Programm ausgewertete werden.

Die bash (born again shell) ist die am meist verbreitete Shell.

Neben der sequentiellen Abarbeitung unterstützt die bash die Shellprogrammierung durch Schleifen und Verzweigungen. Sie ist damit wesentlich flexibler als die Programmierung von Batch-Dateien unter DOS. Ich möchte ein Programm haben, das mir ein geordnetes Directory-listing erstellt (Die zuletzt veränderten Dateien sollen am Ende des Listings erscheinen.)und am Ende eine kleine Zusammenstellung liefert. Aus der man-Page (Aufruf: **man ls**) erfahre ich, dass die notwendigen Optionen **l,r,t** sind. Der einzutippende Befehl lautet: **ls -ltr**. Mit meinem Lieblingseditor (es gibt unter Linux viele derartige !) erstelle ich eine Textdatei mit dem folgenden Inhalt

```
ls -ltr $*
```

und speichere diesen Text unter der Bezeichnung **lt** (so soll mein neues Programm heißen) ab. Die in der **bash** definierte Variable **$***, enthält alle an sie übergebenen Parameter. Es ist also z.B. möglich, sich alle Dateien anzusehen, die die Endung **.txt** haben.

Damit dieser Text zu einem Programm wird, ist es notwendig, den Zugriffmodus auf ausführbar zu schalten.

```
chmod +x lt
```

Um dieses kleine Programm zu testen, gebe ich das folgende Kommando ein.

```
./lt *.txt
```

danach ergibt sich das folgende Listing

```
-rw-r--r--   1 michi    users       70373
Nov 29 14:21
DE-DOS-nach-Linux-HOWTO.txt
-rw-r--r--   1 michi    users       32179
Dez 14 10:37 f2.txt
-rw-r--r--   1 michi    users       31424
Dez 19 11:36 l2d.txt
-rw-r--r--   1 michi    users         369
Dez 19 12:07 bash-programmierung.txt
```

Da das Arbeitsverzeichnis (aus Sicherheitsgründen) kein Bestandteil des Suchpfades ist, muss die Pfadangabe (hier **./**) dem Programmnamen vorangestellt werden. Wenn das Programm dann so funktioniert, wie es soll, kann es in ein Verzeichnis gestellt werden (z.B. **/bin**), welches im Suchpfad enthalten ist.

Das Programm soll auch funktionieren, wenn eine andere Shell dieses Programm aufruft. Zu diesem Zweck wird unserem Programm **lt** in der ersten Zeile mitgeteilt, welche Shell dieses Programm bearbeiten soll. Die erste Zeile in dem Programm lautet:

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

Nun zum zweiten Teil der Aufgabe. Das Programm soll uns am Ende des Verzeichnislistings eine Zusammenfassung (die Anzahl der angezeigten Dateien und der gesamte Speicherbedarf)anzeigen.

Dazu wird die Verzeichnisinformation neben der Ausgabe auf dem Bildschirm auch in eine temporäre Datei geschrieben. Den Namen dieser Datei verbinde ich mit der Prozessnummer. (Die Shellvariable **$$** beinhaltet diese Nummer). Da es keinen Sinn macht, diese Datei anzuzeigen, schließe ich sie vom Anzeigen aus. Der neue Befehl für das Anzeigen der Dateien lautet:

```
ls -ltr -I tmp.$$ $* | tee tmp.$$
```

(**-I** schließt die Datei **tmp.$$** von der Anzeige aus, **|tee** bedeutet, dass neben der Anzeige auf dem Bildschirm in die Datei **tmp.$$** geschrieben wird. **$$** wird durch die aktuelle Prozessnummer ersetzt.)

Als Nächstes wird diese temporäre Datei in einer `for`-Schleife bearbeitet. Die beiden lokalen Variablen `summe` und `anzahl` werden vorher auf 0 gesetzt. Das Pro-

pppd	Starten des Point to Point Dämon
-detach	der **pppd** soll zunächst nach dem Aufruf auch weiterhin im Vordergrund arbeiten
/dev/cua1	bezeichnet die serielle Schnittstelle, an der das Modem angeschlossen ist. **/dev/cua0** steht für **COM1** und **/dev/cua1** steht für **COM2**.
connect	steht vor dem Programm, welches mit dem Modem spricht. Hier wollen wir "**chat**" verwenden. Alles was wir **chat** mitgeben wollen, muss unter einfachen Anführungszeichen (') stehen.
'chat -v "" at\&f1 OK atdt rufnummer CONNECT'	**chat** wird gestartet, **-v** bewirkt, dass **chat** alles mitloggt -> mit "**tail -f /var/log/messages**" können wir zusehen, wie die Kommunikation abläuft. Beim Kommunikationsaufbau handelt es sich um ein Frage - Antwortspiel zwischen unserm PC und dem angerufenen Rechner (=Server). Zunächst wartet unser Modem auf nichts (""). Als nächstes wird das Modem mit **at&f** initialisiert, danach waret das Programm auf ein **OK** vom Modem. Danach wird mit Tonwahl die "**rufnummer**" gewählt und auf ein **CONNECT** gewartet.
38400	ist ein Vorschlag für die Kommunikationsgeschwindigkeit zwischen PC und Modem; es sollte hier auch schneller gehen: z.B. 115200. Anmerkung: die Übertragungsrate von Modem zu Modem ist abhängig von der Qualität der Telefonleitung und kann von Anwender nicht beeinflusst werden.
modem	zur Kommunikation verwenden wir ein Modem
crtscts	die Flusssteuerung erfolgt durch die Hardware
defaultroute	die Modemverbindung wird zum Standardweg aller Datenpakete die ins Internet gehen sollen.
user abc	hier muss genau der Username stehen, der in **/etc/ppp/pap-secrets** eingetragen wurde.
remotename xyz	hier muss das zweite Wort aus **/etc/ppp/pap-secrets** (die Kurzbezeichnung für den Server).

gramm bis zur Schleife sieht dann folgendermaßen aus.

```
#!/bin/sh
# Anzeige aller Dateien laut übergebenem
Muster
# geordnet nach der Änderungszeit, die
ältesten zuerst
#
ls -ltr -I tmp.$$ $* | tee tmp.$$
#
summe=0
anzahl=0
```

Aus der Datei **tmp.$$** wird nun die Information, wie groß die einzelnen Dateien sind mit Hilfe des Kommandos **cut** herausgeschnitten.

Die Information über die Größe der einzelnen Dateien steht an den Position 33 bis 41. Mit Hilfe von **cut** kann aus der Datei **tmp.$$** eine Liste von Zahlen erstellt werden, die der Größe der vorkommenden Dateien entspricht.(die notwendige Option **-b** steht in der man-page).

```
$(cut tmp.$$ -b 33-41)
```

Das führende **$** teilt der Shell mit, dass der Inhalt zwischen den beiden runden Klammern auszuwerten ist, und mit diesem Ergebnis der angegebene Ausdruck zu ersetzen ist. Die so erhaltene Liste der Zahlen wird nun mit einer for-Schleife verarbeitet. Für jedes Element der in der for-Anweisung angegebenen Liste werden die nachfolgenden Befehle verarbeitet. (Nachfolgende Befehle sind jene, die zwischen dem der for-Anweisung folgendem **do** und dem entsprechendem **done** stehen.)

Die Zeile lautet :

```
for i in $(cut tmp.$$ -b 33-41);
do
....
done
```

Leider produziert der **ls**-Befehl an den Positionen 33-41 nicht nur Zahlen, sondern manchmal auch unerwünschten Text. Um etwaige Fehlermeldungen vorzubeugen, ist es sinnvoll, mit Hilfe der Parametersubstitution etwaigen Text zu ersetzen. Die Syntax dafür lautet: **${var_muster}** (zu finden ist dies in der (sehr langen) man-page der **bash** (**man bash**)).Benötigt wird: falls ein Zeichen vorkommt, das keine Ziffer ist, oder falls ein Leerzeichen vorkommt, so soll der gesamte Ausdruck gelöscht werden. **${i##*[^0-9,' ']*}** erfüllt diese Forderung. Die Programmzeile lautet:

```
i=${i##*[^0-9,' ']*}
```

Nur wenn **i** nun irgendwelche Zeichen erhält, soll aufsummiert werden.

```
if [ i ] ; then
     summe = $[[$summe + $1]
     anzahl = $[anzahl + $1]
fi
done
```

Jedes `if` endet mit `fi`, nach dem Ende der Bedingung kommt ein `;` gefolgt vom Schlüsselwort `then`. Um in der **bash** auf den Wert einer Variablen zugreifen zu können, muss vor dem Variablennamen ein `$` vorangesetzt werden. Für die Zuweisung selbst, genügt die Angabe des Variablennamens. Das abschließende `done` gehört zum `for`.

Nun folgt die formatierte Ausgabe der beiden Werte. Da große Werte nicht leicht lesbar sind, sollen ein Punkt, bzw. ein Beistrich nach jeweils drei Ziffern geschrieben werden. `printf` wird verwendet, damit auch führende Nullen in Dreier-Gruppen angezeigt werden.

```
if [ $summe -ge 1000000 ] ; then
    #Speicherverbrauch ist größer als
1.000.000
    echo -n  $[$summe / 1000000]","
    printf '%.3d.' $[($summe %
1000000)/1000]
    printf '%.3d' $[($summe % 1000)]
elif  [ $summe -ge 1000 ] ; then
    #Speicherverbrauch ist größer als 1.000
    echo -n  $[$summe / 1000]","
    printf '%.3d' $[($summe % 1000)]
else
    #normal anzeigen
    echo -n $summe
fi
echo " Byte"
#und nun die Anzahl der Dateien ausgeben
echo "das sind $anzahl dateien"
```

Damit die temporäre Datei nicht zu einer permanenten wird, muss sie nun abschließend gelöscht werden.

```
rm tmp.$$
```

```
#!/bin/sh
# Anzeige aller Dateien laut übergebenem Muster
# geordnet nach der Änderungszeit, die ältesten zuerst
#
ls -ltr -I tmp.$$ $* | tee tmp.$$
#
summe=0
anzahl=0

#nun eine Liste des Speicherverbrauches erzeugen
for i in $(cut tmp.$$ -b 33-41);
do
    # für jedes Element prüfen ob es nur Ziffern enthält
    i=${i##*[^0-9,' ']*}

    #falls das Listenelement nicht leer ist, summiere auf
    if [ i ] ; then
        summe=$[$summe+$i]
        anzahl=$[$anzahl+1]
    fi
done

#und nun ausgeben
echo -n  "gesamter Speicherverbrauch: "

if [ $summe -ge 1000000 ] ; then
    #Speicherverbrauch ist größer als 1.000.000

    echo -n  $[$summe / 1000000]","
    printf '%.3d.' $[($summe % 1000000)/1000]
    printf '%.3d' $[($summe % 1000)]

elif  [ $summe -ge 1000 ] ; then

    #Speicherverbrauch ist größer als 1.000

    echo -n  $[$summe / 1000]","
    printf '%.3d' $[($summe % 1000)]

else

    #normal anzeigen

    echo -n $summe
fi
echo " Byte"

#die temporäre Datei entfernen
rm tmp.$$
```

Nichts auf der Welt ist so gerecht verteilt wie der Verstand: jeder glaubt, genug bekommen zu haben.

Jacques Tati