# Lösungsvorschläge Shell-Programmierung ksh und bash

*Überarbeitet von Christine Wolfinger*

# Inhaltsverzeichnis

# 1 Übungen zum Aufwärmen aus dem Einführungskurs

Bei Ankreuzaufgaben können mehrere Lösungen richtig sein! Falls Sie eine Frage nicht beantworten können, lassen Sie diese einfach aus. __

Sie möchten in Ihr Home-Verzeichnis wechseln.

Welche Befehle sind richtig?

**cd $HOME**

**cd ~**

**cd**

Sie möchten alle Dateien sehen, welche die Endung 'txt' haben. Wie können Sie die Dateinamen dieser Dateien in Ihrem aktuellen Verzeichnis feststellen?

**ls *.txt**

**find . -name "*.txt"**

Kennen Sie noch andere Wildcards?

**? * [ab ] [a-c] [! ]**

Was bewirkt das folgende Kommando? $ grep '^t.*:' /etc/passwd

Es werden die Einträge von Benutzern, die mit t beginnen aus der Datei /etc/passwd ausgegeben.

Sie möchten alle Dateien auf Ihrem Rechner auflisten, die dem Benutzer 'team04' gehören. Welche Befehle sind richtig?

find $HOME -name team04 -falsch

ls /home/team04 - falsch

**find / -user team04**

**find / -user team04 2> /dev/null (***besser***)**

Mit welchem Kommando können Sie feststellen, welche Prozesse auf Ihrem Rechner aktiv sind?

**ps -ef**

Mit welchem Kommando können Sie einen Prozeß beenden/, mit welchem stoppen?

**kill -9** *PID*

**kill -STOP** *PID (s. kill -l - entsprechende Nummer z.B. 15)*

**kill -SIGSTOP** *PID (bash) (s. kill -l - entsprechende Nummer z.B. 19)*

Das Kommando 'ls -l myfile' ergibt folgende Ausgabe:

-rwxr-xr-x team01 staff ... myfile

Welche Kommandos können Sie absetzen, damit nur der Eigentümer Lese-, Schreib- und Ausführungsberechtigung bekommt?

**chmod 700 myfile**

**chmod go-rx myfile**

Das Kommando echo dient dazu, Texte oder Variableninhalte auf der Standardausgabe auszugeben. Was glauben Sie wird bei den folgenden beiden Kommandos ausgegeben?

**echo 'Tell,Wilhelm' | cut -c 1-4** *Ergebnis*: *Tell*

**echo 'Tell,Wilhelm' | cut -d, -f 2** *Ergebnis*: *Wilhelm*

Versuchen Sie die folgenden Taste(n) der entsprechenden vi-Funktionen zuzuordnen: cw, u, R, ESC, D, :q!, J, A, i, cc, dw, a, x, dd, C, ZZ, :wq, yw, yy, p oder P, r

| | |
|---|---|
| Einfüge-Modus verlassen | **<ESC>** |
| Text vor dem Cursor einfügen | **i** |
| Text nach dem Cursor einfügen | **a** |
| Text am Ende der Zeile einfügen | **A** |
| Ein Wort ändern | **cw** |
| Gesamte Zeile ändern | **cc** |
| Zeile vom Cursor bis zum Ende ändern | **C** |
| Ein Wort löschen | **dw** |
| Gesamte Zeile löschen | **dd** |
| Zeile vom Cursor bis zum Ende löschen | **D** oder **d$** |
| Ein Zeichen löschen | **x** |
| Ein Wort kopieren | **yw** |
| Eine Zeile kopieren | **yy** |
| Kopierten/Gelöschten Text einfügen | **p** oder **P** |
| Ein Zeichen ersetzen | **r** |
| Ab der Cursor-Position ersetzen | **R** |
| Letzte Aktion rückgängig machen | **u** |
| Zwei Zeilen zusammenfügen | **J** |
| vi verlassen mit Speichern der Datei : | **:wq** oder **ZZ** |
| vi verlassen ohne Speichern der Datei | **:q!** |

# 2 Übung 2. Shell-Grundlagen

## 2.1 Teil 1 – Umgebung anpassen

Erstellen Sie eine Environment-Datei, die folgendes leistet

- Überprüfen Sie ob de emacs Mode zur Kommandowiederholung eingetragen ist, wenn nicht setzen Sie ihn.
- Definieren Sie einen Alias ll, der das Kommando ls -l aufruft (bei der Bash bereits vordefiniert)
- Definieren Sie einen Alias l, der das Kommando ls -F aufruft
- Setzen Sie ihr primäres Promptsymbol so, daß stets das aktuelle Verzeichnis im Prompt angezeigt wird.

Wenn Sie mit der **Korn-Shell** (ksh) arbeiten, kontrollieren Sie, ob

in .**dtprofile** die letzte Zeite aktiviert ist (nicht mehr mit Kommentarzeichen) **DTSOURCEPROFILE=TRUE** (bzw. ob die Variable anderweitig bereits gesetzt wurde und ob die ENV-Variable gesetzt ist (für ksh: export ENV=~/.kshrc). Auch die Datei **.kshrc** sollte bereits Eintragungen z. B. **set -o emacs** enthalten und die Definitionen der Cursortasten. (Kontrollieren Sie .kshrc deshalb nur mit **cat -v** oder mit dem **vi**, da hier Sonderzeichen verwendet wurden (<Ctrl>)

Wenn Sie mit der **Bash** (bash) arbeiten, kontrollieren die Datei .bashrc bzw. ergänzen Sie sie.

Nachfolgende Einträge gelten nun sowohl für **ksh** als auch für **bash**:

**alias ll= ls -l**

**alias l= ls -F**

**export PS1='$PWD: '**

*für bash auch:* **export PS1='\w: '**

Kontrollieren Sie mit **man bash** welche weiteren Möglichkeiten zur Promptgestaltung vorhanden sind (Suchen **/prompting**).

Melden Sie sich zunächst ab und dann wieder an.

Betrachten Sie zunächst Ihren Prompt.

Erscheint das aktuelle Verzeichnis?

Wechseln Sie in das Verzeichnis /tmp. Erscheint /tmp im Prompt? cd /tmp

Testen Sie die neudefinierten Aliase!

***ll; l***

Testen Sie, ob die Kommandowiederholung funktioniert.

## 2.2 Teil 2: Job-Kontrolle

Starten Sie zunächst das Kommando `sleep` mit dem Argument `1111` im Hintergrund und anschließend mit dem Argument `2222` im Vordergrund.

```
sleep 1111 &
sleep 2222
```

Stoppen Sie den zuletzt gestarteten sleep Befehl.

```
<STRG>-Z
```

Überprüfen Sie den Zustand ihrer Jobs

```
jobs
```

Setzen Sie den gestoppten `sleep` Befehl im Hintergrund fort.

**bg** *PID* (oder **bg** %*nummer*)

Halten Sie den zuerst gestarteten `sleep` Befehl, der im Hintergrund läuft an.

**kill SIGSTOP** *%nummer (bash) oder stop %nummer (ksh)*
und führen Sie den Prozess im Vordergrund weiter.
**fg** %*nummer*

Überprüfen Sie den Zustand ihrer Jobs.

```
jobs
```

Beenden Sie den zweiten gestarteten `sleep` Befehl

**kill** %*nummer*

**Ende der Übung**

# 3 Übung 3. Prozeduren

## 3.1 Teil 1: Mustervorlage erstellen

Erstellen Sie eine Vorlage (Beispiel einer Vorlage)

#!/bin/bash

```
# Aufruf:
# Kruzbeschreibung:
# Autor:
# Abteilung:
# Telefon:
# erstellt am:
# Änderungen:

# ============================ hier kommt das Shell-Skript =================




# ============================ Ende des Shell-Skripts ======================
# Übergabe
# Die Autorin/der Autor übernimmt keine Haftung für Schäden und
# Folgeschäden (Hardware, Datenverluste etc.), die durch die
# Benutzung dieses Scripts entstehen.
#
# Getestet und für richtig befunden.
# Datum:
# Unterschrift:
```

Beispiel unter AIX (Kursunterlage mit what -String – nur bei ksh/AIX)

```
#!/usr/bin/ksh
# @(#) aktiv.sh V1.0 Aktive Benutzer anzeige
# Autor: team01 Datum:
# Aufruf: aktiv
```

## 3.2 Teil 2: Alle Dateien eines Benutzers ausgeben

- suche_user.sh

```
#!/usr/bin/ksh                  oder    #!/bin/bash
#
# @(#) suche_user (Kapitel 2) Suche alle Dateien des Benutzers team01 (nur AIX)
#
# Autor: team01                         Datum:
#
# Aufruf: suche_user
#

# find ...
#          Alle Dateien des Benutzers team01 (-user team01) auflisten.
#          Alle Fehlermeldungen sind zu unterdrcken (2>/dev/null).

find / -user team01 -print 2>/dev/null
```

Durch Angabe von **2> /dev/null**

Ändern Sie **suche_user.sh** als ausführbar. Notieren Sie sich hier das Kommando, das Sie hierfür benötigen:

**chmod +x** - *doch wenn Sie die Vorlage auf "executable" gesetzt haben, ist hier chmod nicht mehr erforderlich*

In welcher **Variable** stehen die Verzeichnisse, in denen die Shell nach Kommandos sucht? Wie ist in Ihrer Umgebung diese Variable gesetzt?

**$PATH**

.....

Ermitteln Sie jetzt mit Hilfe der Prozedur **suche_user.sh** die Anzahl der Dateien, die dem Benutzer **team01** gehören. Ändern Sie dazu aber nicht die Prozedur, sondern nehmen Sie das **wc**-Kommando zur Hilfe. Was würde passieren, wenn Sie die Fehlerausgabe nicht unterdrückt hätten?

**suche_user.sh teamxx | wc -l**

'Debuggen' Sie die Prozedur **suche_user.sh**. Während des Debuggens sollte immer die Zeilennummer angezeigt werden. Mit welchem Kommando können Sie innerhalb einer Prozedur einen 'Trace' anschalten?

**export PS4='$LINENO'**

**set -x**

**set +x**

# 4 Übung Argumente (Variable)

## 4.1 Teil 1: Attribute von Dateien/Verzeichnissen formatiert ausgeben

Erstellen Sie unter Zuhilfenahme Ihres Musterprologes `kopf.sh`, eine Shell-Prozedur mit dem Namen `dinfo.sh`.

4.2a Beim Aufruf von `dinfo.sh` wird ein Argument übergeben. Dieses Argument spezifiziert eine existierende Datei (kann also auch ein Verzeichnis sein). Erzwingen Sie daß `dinfo.sh` endet, wenn kein Argument angegeben wurde. `dinfo.sh` soll die Ausgabe des Kommandos '`ls -ld`' für die jeweilige Datei in der folgenden Art darstellen:

```
$ dinfo /etc/passwd
-----------------------------------------------------
Datei : /etc/passwd
Rechte : -rw-r--r--
Größe : 217
Besitzer : root
Gruppe : security
Letzte Änderung : Mar. 16
-----------------------------------------------------
$ dinfo /tmp
-----------------------------------------------------
Datei : /tmp
Rechte : drwxrwxrwt
Größe : 1536
Besitzer : bin
Gruppe : bin
Letzte Änderung : Mar. 20
-----------------------------------------------------
```

**dinfo**

```
#!/usr/bin/ksh                          nicht für bash!
# @(#) dinfo V1.0 Datei=Informationen ausgeben
# Autor: team01              Datum: 24.01.97
# Aufruf: dinfo Datei#
# $1 muss beim Aufruf mitgegeben werden.
# Wir benutzen dazu die Schreibweise :?
Datei=${1:?"Bitte Dateinamen angeben"}#
# ls =ld fuer die angegebene Datei aufrufen
# und mit read die einzelnen Felder, Variablen zuweisen:
# ls -ld <Datei> | read V1 V2 ...
ls -ld $Datei | read Rechte LN User Group Size DD MM Time Name
# Nun die Datei-Informationen ausgeben:
echo - "-----------------------------------------------------"
echo "Datei              : $1"
echo "Rechte             : $Rechte"
echo "Größe              : $Size"
echo "Besitzer           : $User"
echo "Gruppe             : $Group"
echo "Letzte Änderung    : $DD. $MM"
echo - "-----------------------------------------------------"
```

4.2b Gleiche Aufgabe für bash mit Hilfe der while -Schleife

```
#!/bin/bash
#
# @(#) dinfo V1.0 Datei=Informationen ausgeben
#
# Autor: team01                Datum: 24.01.97
#
# Aufruf: dinfo Datei
#

#
# $1 muss beim Aufruf mitgegeben werden.
# Wir benutzen dazu die Schreibweise :?

Datei=${1:?"Bitte Dateinamen angeben"}
#set -x
#
# ls =ld fuer die angegebene Datei aufrufen
# und mit read die einzelnen Felder, Variablen zuweisen:
#
# ls -ld <Datei> | read V1 V2 ...
#

\ls -ld $* | while read Rechte Ln User Group Size dat Time Name
do
#
# Nun die Datei-Informationen ausgeben:
#

echo - "-------------------------------------------------------"
echo "Datei                : $Name"
echo "Rechte               : $Rechte"
echo "Größe                : $Size"
echo "Besitzer             : $User"
echo "Gruppe               : $Group"
echo "Letzte Änderung      : $dat $Time"
echo - "-------------------------------------------------------"
done
```

## 4.2 Teil 2: Passwörter einlesen (optional)

Erstellen Sie eine Prozedur mit Namen **`readpwd.sh`**. Diese kleine Prozedur soll eine Passworteingabe “simulieren”, d.h. die Eingabe des Passwortes soll auf dem Bildschirm nicht sichtbar sein:

$ readpwd

Passwort eingeben:

Das eingegebene Passwort war: start

Beachten Sie bitte folgende Hinweise: Es ist eine Shell-Konvention, daß ein Prompt immer auf die Fehlerausgabe (also auf den Deskriptor 2) geschrieben wird. Beachten Sie dies bei der Ausgabe desPrompts (im Beispiel oben “Passwort eingeben"'.

**Tipp:** Lenken Sie die Ausgabe des **echo**-Kommandos auf Fehlerausgabe um und testen Sie durch den Aufruf readpwd > /tmp/outputs. Die Eingabe des Benutzers darf nicht auf dem Bildschirm sichtbar sein! **Tipp:** Schauen Sie mal in die **man-Pages** Ihres Rechners unter dem Kommando

**stty,** speziell unter der Option **echo**.

```
#!/bin/bash
# Aufruf: readpwd
# Kruzbeschreibung:
# Autor:
# Abteilung:
# Telefon:
# erstellt am:
# Äderungen:

echo "Password: "
PS1="Passwort eingeben."
stty -F $( tty ) -echo
read passwd
echo das eingegebene Passwort war: $passwd
```

# 5 Kommandosubstitution

## 5.1 Teil 1: Informationen über Benutzer ausgeben

Erstellen Sie unter Zuhilfenahme Ihres Musterprologes **`kopf.sh`**, eine Shell-Prozedur mit dem Namen **`info_user.sh`**.

Die Prozedur **`info_user.sh`** soll folgende Ausgabe produzieren:

```
$ info_user.sh
******************** team01**********************
Der aktuelle Katalog: /tmp
Anzahl Dateien: 123
Das Home-Verzeichnis: /home/team01
Anzahl Prozesse: 12
```

Setzen Sie die Kommandosubstitution ein, um diese Informationen zu generieren.
**Tipps:** Die Anzahl der Zeilen können Sie mit dem Word-Count Kommando (**`wc`** mit der Option **`-l`**) zählen. Beachten Sie Überschriftszeilen! Setzen Sie ggf. das **`tail`**-Kommando ein, um diese auszuschneiden.

**info_user**

```
#!/bin/bash
# @(#) info_user V1.0 Benutzer-Informationen ausgeben
# Autor: team01
# Aufruf: info_user

# Persoenliche Kennung ($LOGNAME) ausgeben:

echo  "************ $LOGNAME ****************"
# Das aktuelle Verzeichnis ueber das pwd-Kommando ausgeben

echo  "Das aktuelle Verzeichnis ist : $(pwd)"
# Anzahl der Dateien ausgeben: ls -l | wc -l
# Die erste Zeile von ls -l duerfen wir aber nicht bercksichtigen:

echo  "Anzahl Dateien        : $(ls -l | tail +2 | wc -l)"
# Nun den Home-Verzeichnis ausgeben:
#
echo  "Das Home-Verzeichnis ist     : $HOME"

#
# Nun die Anzahl der Prozesse ausgeben: ps -u $LOGNAME | wc -l
# Aber die Überschriftenzeile wieder rausschmeissen:

echo  "Anzahl Prozesse       : $(ps -u $LOGNAME | tail +2 | wc -l)"
```

Warum müssen Sie die Überschriftszeile "*** ... ***" in Anführungszeichen setzen?

**Weil Sie sonst die Dateinamen des aktuellen Verzeichnisses bekommen würden.**

## 5.2 Teil 2: Muster in Dateien suchen (optional)

Erstellen Sie eine Prozedur mit Namen **rgrep.sh**, die folgendes leistet:
**rgrep.sh** wird mit zwei Argumenten aufgerufen, einem Suchmuster und einem Startverzeichnis. **rgrep.sh** sucht mittels **find** und **grep** nach dem angegebenen Suchmuster in allen Dateien, die unterhalb des angegebenen Startverzeichnisses liegen:

```
$ rgrep.sh 'peter' /home
- - - - - - - - - - - - - - - - - - - - -
Suche nach dem Muster: peter
Anzahl der Treffer: 3
- - - - - - - - - - - - - - - - - - - - -
/home/peter/.profile
/home/peter/bin/personel
/home/thomas/brief_an_peter
```

Beachten Sie bitte folgende Hinweise:

- Achten Sie darauf, daß ein falscher Aufruf abgefangen wird
- Arbeiten Sie mit den Kommandos **find** und **grep**. Hierzu folgende Tipps: Setzen Sie das **find** so auf, dass nur in gewöhnlichen Dateien gesucht wird (**-type f**). Verwenden Sie die Option **-exec**, um in den gefundenen Dateien automatisch nach dem Suchmuster zu suchen.Verwenden Sie **grep** mit der Option -**l**, d.h. es sollen nur die Namen der Dateien ausgegeben werden, in denen das Suchmuster gefunden wurde.
- Die Namen der Dateien, in denen das Suchmuster gefunden wurde, sollen sortiert auf dem Bildschirm ausgegeben werden (**sort**-Kommando). Außerdem soll die Anzahl der Dateien ermittelt werden, in denen das Suchmuster gefunden wurde.

```
#!/bin/bash
# Aufruf: rgrep.sh 'muster' 'Verzeichnis'
# Kruzbeschreibung: Sucht nach allen Dateien ab 'Verzeichnis',
# deren Inhalt 'muster' enthält
# Autor:
# Abteilung:
# Telefon:
# erstellt am:
# Änderungen:


Muster=${1:?"Aufruf rgrep.sh 'Muster' 'Verzeichnis'"}
Verz=${2:?"Aufruf rgrep.sh 'Muster' 'Verzeichnis'"}

echo "Es wird nach Dateien mit dem Muster $Muster gesucht
ab dem Verzeichnis $Verz"

echo "Anzahl der Treffer: $( find $2 -type f -exec \
 grep -l $1 {} \; | wc -l)"

find $2 -type f -exec grep -l $1 {} \; 2>/dev/null 2>/dev/null | sort

Alternative Lösung (da das find verhältnismäßig lange braucht – die
```

Hälfte der Zeit kann eingespart werden, wenn man es wie folgt verändert:

```
!/bin/bash
# Aufruf: rgrep.sh 'muster' 'Verzeichnis'
# Kruzbeschreibung: Sucht nach allen Dateien ab 'Verzeichnis',
# deren Inhalt 'muster' enthält
# Autor:
# Abteilung:
# Telefon:
# erstellt am:
# Äderungen:


Muster=${1:?"Aufruf rgrep.sh 'Muster' 'Verzeichnis'"}
Verz=${2:?"Aufruf rgrep.sh 'Muster' 'Verzeichnis'"}

echo "Es wird nach Dateien mit dem Muster $Muster gesucht
ab dem Verzeichnis $Verz"

echo "Anzahl der Treffer: $( find $2 -type f -exec grep -l $1 {} \; |
tee /tmp/erg$$ | wc -l)"

#find $2 -type f -exec grep -l $1 {} \; 2>/dev/null 2>/dev/null | sort
sort /tmp/erg$$
```

# 6 set und die Kommandosubstitution

## 6.1 Teil 1: Benutzerinformationen aus der /etc/passwd ausgeben

### Schreiben Sie sich ein Skript "n" , dass ein neues Kommando erstellt und gleich den entsprechenden Editor aufruft

Skript n:

```
#!/bin/bash
# Aufruf: n neuesKommando
# Kruzbeschreibung: Kopiert die Befehlsvorlage und ruft den Editor auf
# Autor: CW
# Abteilung:
# Telefon:
# erstellt am:
# Änderungen:

cd $HOME/bin/
${1:?"Aufruf n Name_von_neuem_Kommando"}
cp Vorlage $1 # bzw. kopf
kate $1 &
read weiter
```

show-user.sh

```
#!/bin/bash
# @(#) show-user.sh V1.0 Benutzer-Informationen ausgeben
#
# Autor: team01
#
# Aufruf: show_user.sh BenutzerName
User=${1:?"Bitte Benutzernamen mitgeben"}
# Die Informationen, die benötigt werden, kommen aus
# der /etc/passwd gelesen. Trennzeichen in der Passwortdatei
# ist das Zeichen ":". Damit belegen wir die Variable IFS.
# IFS=x
IFS=:
# Nun benutzen wir set, zusammen mit einer Kommandosubstitution
# um die Argumente neu zu belegen.
# set ...
set $(grep $User /etc/passwd)
# Nun können wir die gewnschten Informationen ausgeben.
# Die Argumente $1 - $7 sind damit belegt:

echo - "-----------------------------------------------------"
echo - "Kennung:                 $1"
echo - "User-ID:                 $3"
echo - "Group-ID:                $4"
echo - "Kommentar:               $5"
echo - "Home-Verzeichnis:       $6"
echo - "Startprogramm:           $7"
echo - "-----------------------------------------------------"
```

Die Liste wird ausgegeben ohne Inhalt (Variablen $1 bis $7 leer)

Vorgezogen: Übersicht der Variablen-Expandierung
(wird später teilweise genauer erläutert und geübt)

**Variablen-Expandierung in geschweiften Klammern { }**
**Bei allen Beispielen ist die Variabel *name* mit "Hans.Muster" gesetzt, *n* hat keinen Wert,**

| | |
|---|---|
| **${***Variable***}*xyz***<br>**echo ${name}xyz**<br>***Hans.Musterxyz*** | Geschweite Klammer als Begrennzung<br>xyz gehört nicht zum Variablennamen. |
| **${#***Variable***}**<br>**echo ${#name}**<br>*11* | Gibt nur Anzahl Zeichen der Variablen aus. |
| **${***Variable***:-***Wert***}**<br>**echo ${name:-Hans}**<br>*Hans.Muster*<br>**echo ${n:-Hans}**<br>*Hans* | Ist die Variable leer, dann wird *Wert* zugewiesen. |
| **${***Variable***:=***Wert***}**<br>**echo ${name:=Hans}**<br>*Hans.Muster*<br>**echo ${n:=Hans}**<br>*Hans* | Ist die Variable nicht belegt, dann wird *Wert* zugewiesen (gleich wie **:-***Wert*). |
| **${***Variable***:+***Wert***}**<br>**echo ${name:+Hans}**<br>*Hans*<br>**echo ${n:+Hans}**<br>*Hans* | Erzwingt "*Wert*", wenn der Variablenwert nicht leer ist. |
| **${***Variable***:?"***text***"}**<br>**echo ${name:?"Kein Wert\ zugewiesen"}**<br>*Hans.Muster*<br>**echo ${n:?"Kein Wert\ zugewiesen"}**<br>*bash: n: Kein Wert zugewiesen* | Ist die Variable leer, dann eine Fehlermeldung "*text*" ausgegeben. |
| **${***Variable***#***Wert***}**<br>***$*{***Variable***##***Wert***}**<br>**echo ${name#Hans}**<br>*.Muster*<br>**echo ${name#*\.}**<br>*Muster* | Vergleicht den Variablenwert mit *Wert* beginnend am Anfang, der übereinstimmende Teil wird ausgeschnitten.<br>(auch Verwendung von Dateinamenexpansion: *?[ ] möglich).<br># - Entfernung von kürzestem Muster<br>##- Entfernung von längstem passenden Muster |

| | |
|---|---|
| ***${****Variable*%*Wert****}***<br>***${****Variable*%%*Wert****}***<br>**echo ${name%Muster}**<br>*Hans.*<br>**echo ${name%\.*}**<br>*Hans* | ähnlich wie ***{****Variable****#****Wert****}***, hier wird der *Wert* am Ende des Variablenwertes verglichen.<br>% - Entfernung von kürzestem Muster<br>%% - Entfernung von längstem passenden Muster |
| ***${****Variable****/***Muster**/**String***}***<br>***${****Variable****//***Muster**/**String***}***<br>**echo ${name/\./:}**<br>*Hans:Muster*<br>**echo ${name/s/ss}**<br>*Hanss.Muster*<br>**echo ${name//s/ss}**<br>*Hanss.Musster*<br>**echo ${name/Hans./Otto }**<br>*Otto Muster* | Vergleicht und ersetzt das Muster durch den String (/ - einmal, // mehrmals) |

## Teil 2: Arbeiten mit Arrays (optional)

### show_user.opt.sh

```
#!/bin/bash
#
# @(#) show-user.sh V1.0 Benutzer-Informationen ausgeben
# Autor: team01
# Aufruf: show-user.opt.sh BenutzerName
#
User=${1:?"Bitte Benutzernamen mitgeben"}
# Die benötigten Informationen, werden aus
# der /etc/passwd gelesen. Trennzeichen in der Passwortdatei
# ist das Zeichen ":". Damit belegen wir die Variable IFS.

IFS=:
#
# Nun erstellen wir eine Variable als Array:

pwz=($(grep $User /etc/passwd))
# unter ksh set -A pwz=($(grep $User /etc/passwd))
#
# Nun können wir die gewünschten Informationen ausgeben.
# Die einzelen pwz-Array sind damit belegt:

echo - "-----------------------------------------------------"
echo - "Kennung:                 ${pwz[0]}"
echo - "User-ID:                 ${pwz[2]}"
echo - "Group-ID:                ${pwz[3]}"
echo - "Kommentar:               ${pwz[4]}"
echo - "Home-Verzeichnis:       ${pwz[5]}"
echo - "Startprogramm:           ${pwz[6]}"
echo - "-----------------------------------------------------"
```

# 7 Erste Schritte mit der if-Anweisung

## 7.1 Überprüfen, ob Benutzer bekannt ist

cp show_user.sh list_user1.sh

list_user1.sh

```
#!/bin/bash
# @(#) list_user V.0 Erweiterte Benutzer-Informationen ausgeben
# Autor: team0
# Aufruf: list_user kennung
#
# Zuerst mal checken, ob ein Argument berhaupt mitgegeben wurde.
#
User=${1:?"Bitte Benutzernamen mitgeben"}
#
# Zuerst pruefen, ob "kennung" in /etc/passwd existiert:
# grep "^$User" ...
#
if grep -w "^$User" /etc/passwd >/dev/null
then
   #
   # Die Kennung existiert!
   # Nun benutzen wir set zusammen mit grep, um die gewuenschten
   # Informationen festzustellen:
   # set $(grep "^$User" ...)
   # Aber was duerfen wir hier nicht vergessen???

   IFS=:
   set $(grep "^$User" /etc/passwd)
   #
   # Nun mit echo alles ausgeben ...
   #

   echo "-----------------------------------"
   echo "Benutzer:               $User"
   echo "-----------------------------------"
   echo "User-ID:                $3"
   echo "Group-ID:               $4"
   echo "Kommentar:              $5"
   echo "Home-Verzeichnis:      $6"
   echo "Startprogramm:          $7"
   echo "-----------------------------------"
else
   #
   # "kennung" ist in /etc/passwd nicht vorhanden!
   # Hier geben wir eine Fehlermeldung aus und verlassen die
   # Prozedur mit einem Fehlercode.
   echo "Benutzer $User ist dem System nicht bekannt!"
   exit 1

fi
```

## 7.2 Teil 2: Überprüfen, ob Benutzer angemeldet ist (optional)

cp list_user1.sh list_user2.sh

```
#!/bin/bash
# @(#) list_user V.0 Erweiterte Benutzer-Informationen ausgeben
...

User=${1:?"Bitte Benutzernamen mitgeben"}
...

if grep -w "^$User" /etc/passwd >/dev/null
then

...

   IFS=:
   set $(grep "^$User" /etc/passwd)
...
   echo "-----------------------------------"
   echo "Benutzer:               $User"
   echo "-----------------------------------"
   echo "User-ID:                $3"
   echo "Group-ID:               $4"
   echo "Kommentar:              $5"
   echo "Home-Verzeichnis:      $6"
   echo "Startprogramm:          $7"
   echo "-----------------------------------"

   #
   # Nun schauen wir noch, ob der Benutzer momentan arbeitet.
   # Diese Information liefert uns who zusammen mit grep:
   # who | grep "..."
   if who | grep "$User" >/dev/null
   then
      #
      # Benutzer ist aktiv!
      #
      echo "Benutzer $User ist momentan aktiv."

   else
      #
      # Benutzer ist nicht aktiv!
      #
     echo "Benutzer $User ist momentan nicht aktiv."

   fi

else
..

   echo "Benutzer $User ist dem System nicht bekannt!"
   exit 1
fi
```

# 8 Bedingungen formulieren

## 8.1 Teil1: Dateien in den Papierkorb “löschen”

sdel.sh – Schritte 1-7

```
#!/bin/bash
# @(#) sdel V1.0 Sicheres Loeschen einer Datei
# Autor: team01
# Aufruf: sdel dateiname
# 1. Schritt: Argumente pruefen
if [[ $# -ne 1  ]]
then
   echo "sdel: Aufruf: sdel Dateiname"
   exit 1
fi
# 2. Schritt: Belegung der Variablen
Datei=$1
Papierkorb=$HOME/.Papierkorb
Verz=$(dirname $Datei) ||Verz=$PWD      # Vaterverzeichnis
DatName=$(basename $Datei) ||DatName=$1 # Dateiname ohne Pfad
# 3. Schritt: Papierkorb checken
if [[ ! -a $Papierkorb ]]
then
   mkdir $Papierkorb
else
   if [[ ! -d $Papierkorb ]]
   then
      echo "sdel: $Papierkorb existiert und ist kein Verzeichnis."
      exit 1
   fi
fi
# 4. Schritt: Kriterien zum Löschen der Datei abfragen:
if [[ ! -f $Datei || ! -w $Verz || ! -w $Datei || ! -r $Datei ]]
then
   echo "sdel: Fehler: $Datei kann nicht gelöscht werden."
   exit 1
fi
# 5. Schritt: Ist die Datei bereits im Papierkorb?
if [[ -f $Papierkorb/$DatName ]]
then
   echo -n "$DatName existiert bereits. überschreiben (j)?"
   read antwort
   if [[ "$antwort" != "j" ]]
   then
      echo "sdel: $Datei wurde nicht gelöscht"
      exit 1
   fi
fi
# 6. Schritt: Datei jetzt in den Papierkorb verlagern
#             und Prozedur verlassen.
mv $Datei $Papierkorb
exit 0
```

## 8.2 Teil 2: Dateien aus dem Papierkorb wiederherstellen

### 8.8(-8.14) **cp sdel.sh undel.sh**

```
#!/bin/bash
# @(#) undel.sh V1.0 Wiederherstellen einer Datei
# Autor: team01
# Aufruf:  undel.sh  dateiname
# 1. Schritt: Argumente pruefen
if [[ $# -ne 1  ]]
then
   echo "undel: Aufruf: undel.sh Dateiname"
   exit 1
fi
# 2. Schritt: Belegung der Variablen
Datei=$1
Papierkorb=$HOME/.Papierkorb#
# 3. Schritt: Papierkorb überprüfen
if [[ ! -d $Papierkorb ]]
then
   echo "es gibt keinen $Papierkorb"
   exit 1
fi
# 4. Schritt: Dateinamen überprüfen:
#             Pfadnamen sind nicht erlaubt!
if echo $Datei | grep "/" >/dev/null
then
   echo "undel: Pfadnamen sind nicht erlaubt."
   exit 1
fi

# 5. Schritt: Ist die Datei im Papierkorb vorhanden?
if [[ ! -e $Papierkorb/$Datei ]]
then
   echo -n "$Datei ist im Papierkorb nicht vorhanden."
      exit 1
fi
# 6. Schritt: Gibt es die Datei im aktuellen Verzeichnis?
if [[ -e $Datei ]]
then
    echo -n "$Datei existiert bereits. Ueberschreiben (j) ? \c"
    read antwort
    if [[ "$antwort" != "j" ]]
    then
       echo "$Datei wurde nicht wiederhergestellt."
       exit 1
    fi
fi#
# 7. Schritt: Datei jetzt vom Papierkorb ins aktuelle Verzeichnis
#             verlagern.
mv $Papierkorb/$Datei .
echo "$Datei wurde wiederhergestellt."
exit 0
```

# 9 Kompakte Schreibweise mit &&, ||

## 9.1 Kompakte Schreibweise einsetzen

cp sdel.sh  sdel2.sh
cp undel.sh  undel2.sh

sdel2.sh

```
#!/bin/bash
# @(#) sdel.sh V1.0 Sicheres Loeschen einer Datei
# Autor: team01
# Aufruf: sdel2.sh dateiname
# 1. Schritt: Argumente pruefen
[[ $# -ne 1  ]] &&
{
   echo "sdel: Aufruf: sdel Dateiname"
   exit 1
}
# 2. Schritt: Belegung der Variablen
Datei=$1
Papierkorb=$HOME/.Papierkorb
Verz=$(dirname $Datei)||Verz=$PWD       # Vaterverzeichnis
DatName=$(basename $Datei) ||DatName=   # Dateiname ohne Pfad
# 3. Schritt: Papierkorb checken
if [[ ! -a $Papierkorb ]]
then
   mkdir $Papierkorb
else
    [[ ! -d $Papierkorb ]] &&
   {
      echo "sdel: $Papierkorb existiert und ist kein Verzeichnis."
      exit 1
  }
fi
# 4. Schritt: Kriterien zum Löschen der Datei abfragen:
 [[ ! -f $Datei || ! -w $Verz || ! -w $Datei || ! -r $Datei ]] &&
{
   echo "sdel: Fehler: $Datei kann nicht gelöscht werden."
   exit 1
}
# 5. Schritt: Ist die Datei bereits im Papierkorb?
if [[ -e $Papierkorb/$DatName ]]
then
   echo -n "$DatName existiert bereits. überschreiben (j)?"
   read antwort
  [[ "$antwort" != "j" ]] &&
  {
      echo "sdel: $Datei wurde nicht gelöscht"
      exit 1
  }
fi
```

```
#
# 6. Schritt: Datei jetzt in den Papierkorb verlagern
#             und Prozedur verlassen.

mv $Datei $Papierkorb
exit 0
```

## undel2.sh

```
#!/bin/bash
#
# @(#) sdel V1.0 Wiederherstellen einer Datei
# Autor: team01
# Aufruf:  undel  dateiname
# 1. Schritt: Argumente pruefen
[[ $# -ne 1  ]] &&
{
   echo "undel: Aufruf: undel.sh Dateiname"
   exit 1
}
# 2. Schritt: Belegung der Variablen
Datei=$1
Papierkorb=$HOME/.Papierkorb
# 3. Schritt: Papierkorb überprüfen
[[ ! -d $Papierkorb ]] &&
{
   echo "es gibt kein $Papierkorb"
   exit 1
}
# 4. Schritt: Dateinamen überprüfen:
#             Pfadnamen sind nicht erlaubt!
echo $Datei | grep "/" >/dev/null &&
{
   echo "undel: Pfadnamen sind nicht erlaubt."
   exit 1
}
# 5. Schritt: Ist die Datei im Papierkorb vorhanden?
[[ ! -e $Papierkorb/$Datei ]] &&
{
   echo -n "$Datei ist im Papierkorb nicht vorhanden."
   exit 1
}
# 6. Schritt: Gibt es die Datei im aktuellen Verzeichnis?
if [[ -e $Datei ]]
then
    echo -n "$Datei existiert bereits. Ueberschreiben (j) ? \c"
    read antwort
   [[ "$antwort" != "j" ]] &&
   {
       echo "$Datei wurde nicht wiederhergestellt."
       exit 1
   }
fi
```

```
# 7. Schritt: Datei jetzt vom Papierkorb ins aktuelle Verzeichnis
#             verlagern.

mv $Papierkorb/$Datei .
echo "$Datei wurde wiederhergestellt."

exit 0
```

# 10 Arbeiten mit Funktionen

## 10.1 Teil 1: Funktionenbibliothek: Ausstieg aus Prozeduren und Abfrage

cd ~/bin; mkdir funktionenbibliothek

Datei: func.lib (einschl. 10.3)

```
# @(#) func.lib V1.0 Funktionen-Bibliothek für sdel/undel
# Autor: team01
# Verwendung: . func.lib
# Funktion fatal
# Parameter:    $1 Fehlertext#
# Zweck:   Fehlertext ausgeben und Prozedur mit Fehlercode verlassen
# Rückgabewert: Keiner
function fatal
{
  echo $1
  exit 1
}
# Funktion checkyn#
# Parameter:    $1 Ausgabetext zum Bestätigen              #
# Zweck:        Text ausgeben und um Bestätigung bitten#
# Rückgabewert:  0 -> Ja
#                1 -> Nein
#                2 -> Alles andere

function checkyn
{
  while  true
  do
     echo -n "$1"
     read Antwort
     case "$Antwort" in
       [jJ] | [jJ][aA])     return 0 ;;# auch ja|Ja|J
       [nN] | [nN]ein)  return 1 ;; # auch nein|Nein|N
     esac
  done
}
```

## 10.2 Teil 2: Verwendung der Funktionen aus Bibliothek in den Prozeduren

10.5 cp sdel2.sh sdelf.sh; cp undel2.sh undelf.sh

## 10.3 10.5 a sdelf.sh

```
#!/bin/bash
# @(#) sdel V1.0 Sicheres Loeschen einer Datei
# Autor: team01
# Aufruf: sdelf.sh dateiname
# Funktionen-Bibliothek func.lib zuerst einbinden:
. ~/bin/Au23/functions/func.lib
# 1. Schritt: Argumente pruefen
[[ $# -ne 1  ]] && fatal "sdel: Aufruf: sdel Dateiname"
# 2. Schritt: Belegung der Variablen
Datei=$1
Papierkorb=$HOME/.Papierkorb
Verz=$(dirname $Datei) ||Verz=$PWD      # Vaterverzeichnis
DatName=$(basename $Datei)||DatName=$1  # Dateiname ohne Pfad
# 3. Schritt: Papierkorb checken
if [[ ! -a $Papierkorb ]]
then
   mkdir $Papierkorb
elif [[ ! -d $Papierkorb ]]
  then
       fatal "sdelf: $Papierkorb existiert und ist kein Verzeichnis."
fi
# 4. Schritt: Kriterien zum Löschen der Datei abfragen:
 [[ ! -f $Datei || ! -w $Verz || ! -w $Datei || ! -r $Datei ]] \
  && fatal "sdelf: Fehler: $Datei kann nicht gelöscht werden."
# 5. Schritt: Ist die Datei bereits im Papierkorb?
if [[ -a $Papierkorb/$DatName ]]
then
   checkyn "$Datname existiert bereits. überschreiben (j.n)?"

  if [[ "$?" -ne 0 ]]
   then
      fatal "sdelf: $DatName wurde nicht gelöscht"
  fi
fi
#
# 6. Schritt: Datei jetzt in den Papierkorb verlagern
#             und Prozedur verlassen.

mv $Datei $Papierkorb
```

10.5b undelf.sh

```
#!/bin/bash
# @(#) sdel V1.0 Wiederherstellen einer Datei
# Autor: team01
# Aufruf:  undel  dateiname
# Funktionen-Bibliothek func.lib einbinde
. ~/bin/Au23/functions/func.lib
# 1. Schritt: Argumente pruefen

[[ $# -ne 1  ]] && fatal "undel.sh: Aufruf: undel Datei"

# 2. Schritt: Belegung der Variablen

Datei=$1
Papierkorb=$HOME/.Papierkorb

# 3. Schritt: Papierkorb überprüfen
[[ ! -d $Papierkorb ]] && fatal "undel: kein Verzeichins $Papierkorb"
# 4. Schritt: Dateinamen überprüfen:
#             Pfadnamen sind nicht erlaubt!
echo $Datei | grep "/" >/dev/null && fatal "undel: Pfadnamen sind
nicht erlaubt."

# 5. Schritt: Ist die Datei im Papierkorb vorhanden?

[[ ! -e $Papierkorb/$Datei ]] &&
  fatal "undel: $Datei ist im Papierkorb nicht vorhanden."
#
# 6. Schritt: Gibt es die Datei im aktuellen Verzeichnis?
if [[ -e $Datei ]]
then
    checkyn "$Datei existiert bereits. Ueberschreiben (j) ? \c"

   if [[ "$?"  -ne 0 ]]
   then
       fatal "$Datei wurde nicht wiederhergestellt."
     fi
fi
#
# 7. Schritt: Datei jetzt vom Papierkorb ins aktuelle Verzeichnis
# verlagern.
mv $Papierkorb/$Datei .
#
echo "$Datei wurde wiederhergestellt."
```

## 10.4 Teil3 – nur optional – evtl. gemeinsam

# 11 Vertieftes Arbeiten mit Funktionen

**mycd**

*Testen Sie die* Prompting-*Funktionen der bash)*. Für ksh:

```
#
# functions mycd  # ksh
mycd ()  # bash
{

PS1='$(tput smso) $(date +%T), ${PWD}: $(tput rmso)'
```

Erstellen Sie eine Funktion mymkdir in Ihrem Home-Verzeichnis, die nachdem Sie ein Verzeichnis angelegt haben, sofort in dieses wechselt:

```
function mymkdir
{
mkdir $1
cd $1
}
```

Lesen Sie die Funktion in Ihre aktuelle Shell ein und testen Sie sie.

```
. $HOME/funktionen
mkdir Test1
pwd
```

# 12 Arbeiten mit der while-Schleife

## Anmeldung eines Benutzers überwachen

```
#
# @(#) blogin.sh: Anmeldung überwachen
#
User=${1:?"Bitte Benutzer angeben"}
until who | grep $User >/dev/null
do
sleep 30
done
echo "Anmeldung: $User, $(date)"
```

## Abmeldung eines Benutzers überwachen

```
#!/bin/bash

# @(#) blogoff V1.0 Logoff eines Benutzers ueberwachen
#
# Autor: team01                  #
# Aufruf: blogoff kennung#
# Argumente überprüfen!

[[ $# != 1 ]] &&
{
  echo "blogoff: Aufruf: blogoff kennung"
  exit 1
}
# Folgende Variablen initialieren:
# Schlafzeit auf 60 Sekunden setzen
# Protokoll mit "$HOME/.logprot" setzen
Schlafzeit=60
Protokoll=$HOME/.logprot
## Falls der Benutzer nicht angemeldet ist, verlassen wir
# die Prozedur mit einem Fehler.
# Wir benutzen hierzu who mit grep:
who | grep $1 >/dev/null ||
{
  echo "$1 ist zur Zeit nicht angemeldet."
  exit 1}
# Nachricht auf den Bildschirm schreiben, dass $1 nun
# "überwacht" wird ...
echo "Ausloggen von $1 wird überwacht..."
# Solange der Benutzer nun angemeldet ist, versetzen
# wir die Prozedur fuer $Schlafzeit in den "Schlafzustand":
while who | grep $1 >/dev/null
do
  sleep $Schlafzeit
done#
# An dieser Stelle, protokollieren wir nun das Ausloggen
# des Benutzers $1 in $Protokoll und auf dem Bildschirm:
echo "Abmeldung $1:     ($(date))" | tee -a $Protokoll
```

# 13 Arbeiten mit der for-Schleife

```
Veränderte Version sdel.sh (for in) *
#!/bin/bash
#
# @(#) sdelfSchleife V1.0 Sicheres Loeschen einer Datei
# Autor: team01
# Aufruf: sdelfSchleife dateiname
#
# Funktionen-Bibliothek func.lib zuerst einbinden:
#
. ~/bin/Au23/functions/func.lib  #ebenfals geändert fatal nicht exit!
#
# 1. Schritt: Argumente pruefen
 [[ $# -lt 1  ]] && fatal "sdel: Aufruf: sdel Dateiname(n)" #Änderung
# 2. Schritt: Belegung der Variablen durch for Datei in $*
# Datei=$1                        #entfernt
      Papierkorb=$HOME/.Papierkorb

#     Verz=$(dirname $Datei)           # Vaterverzeichnis
#     DatName=$(basename $Datei)       # Dateiname ohne Pfad entfernt
# 3. Schritt: Papierkorb checken
if [[ ! -a $Papierkorb ]]
then
   mkdir $Papierkorb
elif [[ ! -d $Papierkorb ]]
  then
       fatal "sdelf: $Papierkorb existiert und ist kein Verzeichnis."
fi
# 4. Schritt: Kriterien zum Löschen der Datei abfragen:
for Datei in $*                          #Änderung
 do                                     #Änderung
 DatName=$(basename $Datei)        # Dateiname ohne Pfad nun hier!!!
 Verz=$(dirname $Datei) || Verz=$(pwd)
 [[ ! -f $DatName || ! -w $Verz || ! -w $DatName || ! -r $DatName ]] \
  && fatal "sdelf: Fehler: $DatName kann nicht gelöscht werden."
#
# 5. Schritt: Ist die Datei bereits im Papierkorb?
#
      if [[ -a $Papierkorb/$DatName ]]
      then
         checkyn "$DatName existiert bereits. überschreiben (j.n)?"
        if [[ "$?" -ne 0 ]]
        then
         fatal "sdelf: $DatName wurde nicht gelöscht"
        fi
      fi
#
# 6. Schritt: Datei jetzt in den Papierkorb verlagern
#             und Prozedur verlassen.
mv $Datei $Papierkorb
done                                #Änderung
```

# 14 Endlosschleifen

## 14.1 Funktion checkyn akzeptiert nur gültige Eingaben

Geänderte func.lib

```
#
# Funktion checkyn
#
# Parameter:    $1 Ausgabetext zum Bestätigen
#
# Zweck:        Text ausgeben und um Bestätigung bitten
#
# Rückgabewert: 0 -> Ja
#               1 -> Nein
#          2 -> Alles andere

function checkyn
{

  while true            # Änderung bei Übung 14
  do

     echo -n "$1"
     read Antwort
     case "$Antwort" in
       [jJ] | [jJ][aA])     return 0 ;;
       [nN] | [nN]ein)  return 1 ;;
     esac

  done
}
```

bkdaemon

```
#!/bin/bash
# @(#) bkdaemon V1.0 Sicherungskopie von Dateien erstellen
# Autor: team01                 Datum: 07.04.98
# Aufruf: bkdaemon Verzeichnis
#
# Funktionen-Bibliothek func.lib einbinden:
#
. func.lib
#
# Argumente überprfen
[[ $# -ne 1 ]] && fatal "bkdaemon: Aufruf: bkdaemon Verzeichnis"
#
# Variablen belegen:
Verzeichnis=$1
Ext="BAK"
Schlafzeit=120
# Ins angegebene Verzeichnis wechseln:
# Falls cd nicht möglich, fatal "Falsches Verzeichnis"
cd $Verzeichnis 2>/dev/null || fatal "Fehler: Verzeichnis $1 ist
falsch."
# while true ...
while true
do
   #
   # Für alle Dateien in diesem Verzeichnis
   #
   for Datei in *
   do
     [[ -e $Datei ]] || break    # * konnte nicht aufgelöst werden bei
*?
     [[ -d $Datei ]] && continue # Datei ist Verzeichnis
                              # BAK-Dateien sollen nicht kopiert werden
     [[ $Datei = *.$Ext ]] && continue
     # Gibt's bereits eine BAK-Datei, so müssen diese verglichen
werden
     if [[ -e $Datei.$Ext ]]
     then
        if [[ ! $Datei -nt $Datei.$Ext ]] && continue
     fi

     # Dateien jetzt kopieren
     cp "$Datei" "$Datei.$Ext"

   done

   sleep $Schlafzeit

done
```

# 15 Datenumlenkung mit Schleifen

## 15.1 Teil 1: Sichern von Home-Verzeichnissen ausgewählter Benutzer

```
# Datei: archive.list
# Die Home-Verzeichnisse folgender Kennungen werden
# bei der Datensicherung berücksichtigt:
#-------------------------------------------------------
# Kennung Kommentar
#-------------------------------------------------------
team01     /home/team01
team06     /home/team06
team02     /home/team02
```

archive1.sh

```
# @(#) archive V1.0: Archivieren von Benutzerdate
# Aufruf: archive1.s
# Autor: team01                        Datum: 23.04.9
# Anmerkung: Steuerungsdatei archive.list anpassen
# set -x # Aktivieren bei auftretenden Fehlern
(( $# != 0 )) && { echo "archive: Aufruf: archive1.sh"; exit 1; }
export Term=$(tty)
# Alle Fehlerausgaben auf Datei sich.err umleiten:
exec 2>sich.err
# Überschrift in sich.err schreiben.
# ACHTUNG: Nicht direkt in sich.err schreiben, sondern mit
#          File-Deskriptoren arbeiten:
echo "Fehlerprotokoll der Sicherung vom: $(date)" >&2
# Mit einem Kommando alle Zeilen mit # am Anfang ausfiltern:
# grep -v "#" archive.list | ...
# Und mit while read weiterverarbeiten ...
grep -v "#" archive.list |
while read Kennung Kommentar
do
   # Home-Verzeichnis aus /etc/passwd ausschneiden:
   # grep "^$Kennung" /etc/passwd | cut ...)
   Home=$(grep "^$Kennung " /etc/passwd | cut -d: -f6)
   # Existiert das Home-Verzeichnis ?
   [ ! -d "$Home" ] &&
   {
     # Fehler auf Bildschirm ausgebenWichtig >/dev/tty
     # Danach nächste Kennung bearbeiten ...
     echo "archive: Home-Verzeichnis fuer $Kennung existiert nicht." > $Term
     continue
   }
   # Meldung auf Bildschirm ausgeben:
   echo "Sicherung der Kennung: $Kennung $Kommentar" > $Term
   # Suche alle Dateien unterhalb $Home und sichere diese.
   # find schreibt auf die Standardausgabe (und damit auf die cpio-Pipe).
   find $Home -print
done | cpio -ocv > /tmp/Sicherung
echo Sicherung fertig
```

## 15.2 Teil 2: Beenden von Prozessen (optional)

mykill.sh

```
#!/bin/bash
#
# @(#) mykill: Alle Prozesse eines Benutzers killen
#
# Aufruf: mykill Benutzer
#
#set -x
function killprocess
{
  typeset Pid=$1
  typeset Cmd=$2
  echo "Folgender Prozess wurde gefunden: "
  echo "ProzessId: $Pid"
  echo "Kommando: $Cmd"
  echo -n "Prozess killen? (j,J): "
  read OK
  if [[ $OK = [jJ] ]]
  then
     kill -9 $Pid 2>/dev/null
  fi
  echo ""
}
User=${1:?"Bitte Benutzer angeben"}
#
# Beim ps nur Pid und Kommando ausgeben.
# Außerdem schreiben wir auf eine Datei von
# der wir nachher mit einem anderen Filedeskriptor
# lesen können:
ps -o pid,args -u $User | tail +2 > /tmp/mykill$$
#while read -u3 Pid Cmd   #nur ksh
while read Pid Cmd <&3
do
  killprocess "$Pid" "$Cmd"
done 3</tmp/mykill$$
rm /tmp/mykill$$
```

# 16 Die Select-Anweisung

aedit

```
#!/bin/bash
#
# @(#) aedit V1.0: Ausgewaehlte Datei editieren
#
# Autor: team01                       Datum:
#
# Aufruf: aedit.sh
#
# Falls die EDITOR-Variable nicht gesetzt ist, nehme den vi.
# Ist sie gesetzt, nehme den enstsprechenden Eintrag:
MyEditor=${EDITOR:-kate}
#
# Im folgenden müssen wir nun eine Liste generieren, die
# alle Dateien des aktuellen Verzeichnisses beinhaltet, aber
# Unterverzeichnisse ignoriert.
# Hierzu benutzen wir einen kleinen Trick:
# ls -F zeigt den Dateityp an der letzten Stelle an. Verzeichnisse
# sind dabei mit einem "/" gekennzeichet - wir mssen also nur
# ein ls -F mit grep -v "/$" kombinieren. Aber Vorsicht: Den "*"
# mssen wir entfernen, da wir sonst doppelte Dateien bekommen!
Liste=$(ls -F | grep -v "/$" | tr '*' ' ')
#
# Nun noch das Prompt-Zeichen PS3 belegen:
#
PS3="Ihre Wahl ? "
#
# Nun das Menü mit select anzeigen:
#
select Datei in $Liste Ende
do
  case "$Datei" in
    Ende) exit 0;;
    "")   echo "Falsche Ausgabe" ;;
    *)    "$MyEditor" $Datei;;
  esac
done
```

Teil2 – (optional – hier nicht weiter behandelt)

# 17 Arbeiten mit typeset/printf typeset mit -L nicht unter der bash

mydf.sh

```
#!/bin/ksh
# @(#) mydf: Ein eigenes df-Program
# Aufruf: mydf
# Übung 9A: Arbeiten mit typeset
# Variablen für die Kopfzeile belegen: # - nur unter ksh !!
# typeset -L20 KOPF1="Dateisystem:" # - nur unter ksh !!
# typeset -L15 KOPF2="Gesamt (MB):" # - nur unter ksh !!
# typeset -L15 KOPF3="Frei (MB):" # - nur unter ksh !!
# typeset -L12 KOPF4="Belegt (%):"  # - nur unter ksh !!
# Variablen für den Datensatz belegen: # - nur unter ksh !!
# typeset -L20 FILESYSTEM # - nur unter ksh !!
# typeset -L15 TBLOCKS_MB # - nur unter ksh !!
# typeset -L15 FBLOCKS_MB # - nur unter ksh !!
# typeset -L5  PROZENT # - nur unter ksh !!
export KOPF1="Dateisystem"
export KOPF2="Gesamt_MB"
export KOPF3="Frei_MB"
export KOPF4="Belegt_%"
typeset -i TBLOCKS
typeset -i FBLOCKS
typeset -i USEDBL
typeset -i TBLOCKS_MB
typeset -i FBLOCKS_MB
typeset -i PROZENT
# Eine Variable GRENZWERT mit dem Wert 90 belegen:
GRENZWERT=90
#
# Kopfzeilen gem□ □ungsbeschreibung nun ausgeben:
#echo "$KOPF1 $KOPF2 $KOPF3 $KOPF4"
#echo "================================================================================"
#set -x
printf "%-25s %15s %15s %15s\n" $KOPF1 $KOPF2 $KOPF3 $KOPF4
echo "================================================================================"
# while read LV TBLOCKS FBLOCKS PCT_USED IUSED PCT_IUSED FILESYSTEM #testen für ksh
# für bash
df -k | tail +2  | \
while read LV TBLOCKS USEDBL FBLOCKS PCT_USED FILESYSTEM
do
   TBLOCKS_MB="TBLOCKS/1024"
   FBLOCKS_MB="FBLOCKS/1024"
    PROZENT=$(echo $PCT_USED | tr "%" " ")
   # printf ("%-20s %-s15 %-s15 %-s5", $KOPF1, $KOPF2, $KOPF3, $KOPF4)
    # Nun die Zeile wie in der Übungsbeschreibung dargestellt,
    # ausgeben. Da noch diese Grenzwert-Sache kommt, hier den
    # Zeilenvorschub unterdrücken: echo -n
   # echo -n "$FILESYSTEM $TBLOCKS_MB $FBLOCKS_MB" # - nur unter ksh !!
   printf "%-25s %15d %15d %15s" "$FILESYSTEM" "$TBLOCKS_MB" \
                                                    "$FBLOCKS_MB" "$PROZENT"
    if [[ $PROZENT -gt $GRENZWERT ]]  #obere Zeile in eine Zeile bei printf
    then
       echo "   *** Achtung"
    else
       echo ""
    fi
done
echo "================================================================================"
```

Ausgabe df aufbereiten (optional):

## 17.1 mydf (inkl Teil 2: Erweiterung (optional)

```
#!/bin/bash#
# @(#) mydf: Ein eigenes df-Programm#
# Aufruf: mydf#
# Variablen für die Kopfzeile belegen: # - nur unter ksh !!
# typeset -L20 KOPF1="Dateisystem:" # - nur unter ksh !!
# typeset -L15 KOPF2="Gesamt (MB):" # - nur unter ksh !!
# typeset -L15 KOPF3="Frei (MB):" # - nur unter ksh !!
# typeset -L12 KOPF4="Belegt (%):"  # - nur unter ksh !!
# Variablen für den Datensatz belegen: # - nur unter ksh !!
# typeset -L20 FILESYSTEM # - nur unter ksh !!
# typeset -L15 TBLOCKS_MB # - nur unter ksh !!
# typeset -L15 FBLOCKS_MB # - nur unter ksh !!
# typeset -L5  PROZENT # - nur unter ksh !!
export KOPF1="Dateisystem"
export KOPF2="Gesamt_MB"
export KOPF3="Frei_MB"
export KOPF4="Belegt_%"
typeset -i TBLOCKS
typeset -i FBLOCKS
typeset -i USEDBL
typeset -i TBLOCKS_MB
typeset -i FBLOCKS_MB
typeset -i PROZENT
# Eine Variable GRENZWERT mit dem Wert 90 belegen:
GRENZWERT=90
# Kopfzeilen ausgeben:
printf "%-25s %15s %15s %15s\n" $KOPF1 $KOPF2 $KOPF3 $KOPF4
echo "============================================================="
# für bash:
df -k | tail +2  | \
while read LV TBLOCKS USEDBL FBLOCKS  PCT_USED FILESYSTEM
do
   TBLOCKS_MB="TBLOCKS/1024"
   FBLOCKS_MB="FBLOCKS/1024"
    PROZENT=$(echo $PCT_USED | tr "%" " ")
   # Nun die Zeile wie in der Übungsbeschreibung dargestellt,
   # ausgeben. Da noch diese Grenzwert-Sache kommt, hier den
   # Zeilenvorschub unterdrücken: echo -n ...
   # echo -n "$FILESYSTEM $TBLOCKS_MB $FBLOCKS_MB"# nur unter ksh !!
   printf "%-25s %15d %15d %15s" "$FILESYSTEM"  "$TBLOCKS_MB"\
"$FBLOCKS_MB" "$PROZENT"
    [[ $PROZENT -gt $GRENZWERT ]] && echo " *** Achtung"\
    || echo ""
  fi
done
echo
"================================================================="
```

# 18 Arbeiten mit Mustern

## 18.1 Teil1: Sicherungskopie vor dem Editieren erstellen

editor

```
#!/bin/bash
# @(#) editor V1.0 Sicherungskopie erstellen und Datei mit Editor
bearbeiten
#
# Autor: team01
# Aufruf: editor <Dateiname>
#
#
# Argumente prüfen
#
(( $# != 1 )) && { print "editor: Falscher Aufruf."; exit 1; }
#
# Variablen definieren
#
Datei=$1
EDITOR="vi"
Ext="BAK"
#
# BAK-Dateien dürfen nicht editiert werden.
#
[[ $Datei = *.$Ext ]] &&
{
 print "$Datei: Sicherungskopien können nicht editiert werden."
 exit 1
}
# Nun Dateityp überprfen: Verzeichnisse können nicht editiert werden.
[ -d "$Datei" ] && { print "$Datei ist ein Verzeichnis."; exit 1; }
# Falls Datei eine reguläre Datei ist, Kopie erstellen.
# Ansonsten wird keine Kopie erstellt:
if [ -f "$Datei" ]
then
   #
   # Vorderen Teil ausschneiden:
   #
   Kopie=${Datei%%.*}
   cp $Datei $Kopie.$Ext

fi
$EDITOR $Datei

Die Ausgabe des UNIX-Kommandos df umformatieren
```

## 18.2 Teil 2: Arbeiten mit regulären Ausdrücken (optional)

checknum,

```
#
# @(#) checknum: Numerisch oder nicht ...
#
# Aufruf: checknum "Argument"
#

echo $1 | grep "^[0-9]*$" && echo $1 ist numerisch || echo $1 \
ist nicht numerisch
```

# 19 Bedingungen formulieren

## 19.1 Teil1: Signalbehandlung

bkdaemon (bis 19.3)

```
#!/bin/bash
# @(#) bkdaemon V1.0 Sicherungskopie von Dateien erstellen
# Autor: team01                 Datum: 07.04.98
# Aufruf: bkdaemon Verzeichnis
# Funktionen-Bibliothek func.lib einbinden:
. func.lib
# trap-Handler definieren
function cleanup
{
  print "$0: Proze◆wurde beendet: $(date)" | tee /tmp/bkdaemon.$$
  exit 1
}
trap "" 1                                          # Hangup
trap "print \"Abbruch nicht möglich\"" 2 3         # Interrupt, Quit
trap cleanup 15                                    # Termination
# Argumente überprüfen
[[ $# -ne 1 ]] && fatal "bkdaemon: Aufruf: bkdaemon Verzeichnis"
# Variablen belegen:
Verzeichnis=$1
Ext="BAK"
Schlafzeit=120
# Ins angegebene Verzeichnis wechseln:
# Falls cd nicht möglich, fatal "Falsches Verzeichnis"
cd $Verzeichnis 2>/dev/null || fatal "Fehler: Verzeichnis $1 ist
falsch."
# while true ...
while true
do
   #
   # Für alle Dateien in diesem Verzeichnis
   for Datei in *
   do
     [[ -a $Datei ]] || break    # * konnte nicht aufgel◆t werden
     [[ -d $Datei ]] && continue # Datei ist Verzeichnis
                                 # BAK-Dateien nicht kopiert werden
     [[ $Datei = *.$Ext ]] && continue
     # Gibt's bereits eine BAK-Datei,  diese vergleichen
     if [[ -a $Datei.$Ext ]]
     then
        cmp -s "$Datei" "$Datei.$Ext" && continue
     fi
     # Dateien jetzt kopieren
     cp "$Datei" "$Datei.$Ext"
   done
   sleep $Schlafzeit

done
```

# 20 Arbeiten mit eval

## 20.1 Teil 2: Arbeiten mit Subshells

20.2 cpdir

```
#!/bin/bash
#
# @(#) cpdir V1.0 Kopieren von Dateibäumen
#
# Autor: team01
#
# Aufruf: cpdir datei_baum1 [datei_baum2 ...] ziel_katalog
#
# Feststellen des letzten Arguments: Ziel
eval 'Zieldir=$'$#
echo es wird nach $Zieldir kopiert
# Solange noch ein Argument vorhanden ist ...
while [ $# -gt 1 ]
do
  echo "Kopieren von $1 nach $Zieldir "

  # Subprozess 1: Wechsle nach $1, tar auf Standardausgabe (-)
  # Subprozess 2: Wechsle nach $Zieldir , tar von Standardeingabe (-)
  (cd $1 ; tar cf - .) | (cd $Zieldir ; tar xf -)
  shift                # $2 wird zu $1, $# wird um eins reduziert;
done
```

# 21 Arbeiten mit awk

## 21.1 Teil1: Benutzerinformationen aufbereiten

uinfo

```
#!/bin/bash
set -x
sort /etc/passwd | grep -v "/bin/false" | awk '
BEGIN {
 FS=":"
printf ("%-10s %5s %5s %-15s %-15s\n", "Name","ID", "Group", "Verz",
"Programm")
}
# Main Loop (Aktionen)
{
#print $1, $4,$5
printf ("%-10s %5s %5s %-15s %-15s\n", $1, $3, $4, $6, $7)
}
'
```

## 21.2 Ausgabe df aufbereiten (optional):

mydf2

```
#!/bin/bash
# @(#) mydf2 V1.0 Verbessertes Disk-Free Command mit awk
# Autor: team01      Datum: 27.04.98
# Aufruf: mydf2
# df -k an awk "pipen". Achtung: Überschriftszeile von df muss raus:
#
df -k | tail +2 | awk '
BEGIN {
      print "----------------------------------------"
        printf("%-20s %10s\n", "Dateisystem:", "Frei:")
      print "----------------------------------------"
        freie_bytes = 0
        summe = 0
}
# Main Loop
{
       freie_bytes = $3/1024
       summe = summe + freie_bytes
       if ( freie_bytes < 10 )
         printf("%-20s %10.2f *** Achtung \n", $7, freie_bytes)
       else
         printf("%-20s %10.2f\n", $7, freie_bytes)
}
END {
      print "----------------------------------------"
        printf("%-20s %10.2f\n", "Summe:", summe)
}
'
```

# 22 Arbeiten mit sed

## 22.1 Dateiinformationen aufbereiten

dir

```
#!/bin/bash
# @(#) dir: Ein  ls
# Aufruf: dir ...

ls -lF $* | sed '
s/*$/ (exe)/
s/\/$/  (dir)/
s/@$/  (link)/
'
```

### 22.1a dirkurz

```
#!/bin/bash
#
# @(#) dir: Ein  ls
#
# Aufruf: dir ...
#

ls -lF $* | sed '
s/*$/ (exe)/
s/\/$/  (dir)/
s/@$/  (link)/
'
```