

Foliensatz B

Shell-Programmierung, Kommandozeilenwerkzeuge

Prof. Dr.-Ing. Hans-Georg Eßer

esser.hans-georg@fh-swf.de
<http://swf.hgesser.de/>

8. Oktober 2020

FH Südwestfalen, Informatik, WS 2020/21

v1.3, 2020/10/08

Einleitung

Linux-Systemverwaltung

Lehrmethodik: So nicht ...

- „So tickt ein Linux-Admin“
- Problemlösungskompetenz
 - Problem untersuchen
 - Ursache feststellen
 - Aus Fundus der Unix-Tools Lösungsansatz überlegen
 - Lösung umsetzen und testen

1. Shell-Programmierung

- 1.1 Variablen und Konstanten
- 1.2 Mathematische und logische Operationen
- 1.3 Schleifen
- 1.4 Funktionen
- 1.5 Fehlerbehandlung
- ...

2. Standard-Tools

- 2.1 awk
- 2.2 bc
- 2.3 cat
- 2.4 clear
- 2.5 find
- 2.6 grep
- ...

Herangehensweise

- Präsentation eines Problems
- Schrittweises Erarbeiten und Verbessern einer Lösung
- dabei: Sprachelemente der Bash kennenlernen
- am Ende: Überblick, was an Neuem dabei war

Problem 1: Verzeichnisse überwachen

3

Problem 1: Trivialbeispiel

Lösung 1, Version 1

- Admin möchte den Inhalt von drei Ordnern
 - /var/log,
 - /var/spool/mail
 - und /var/spool/cupsüberwachen.
- Bisherige Vorgehensweise:
 - `ls -l /var/log`
 - `ls -l /var/spool/mail`
 - `ls -l /var/spool/cups`

- Die drei Befehle in eine Skriptdatei `info.sh` packen:

```
1 #!/bin/bash
2 # info.sh, Beschreibung folgt
3 ls -l /var/log
4 ls -l /var/spool/mail
5 ls -l /var/spool/cups
```

- `info.sh` ausführbar machen und an geeigneten Ort kopieren (Pfade!)

```
chmod a+x info.sh
mv info.sh /root/bin/
```

(setzt voraus, dass /root/bin im Pfad \$PATH steht)

4

5

Beobachtungen/Fragen

- Ausgabe von `/var/log` ist sehr lang, interessante Dateien (die sich zuletzt geändert haben) stehen mitten drin
- Skript ist nicht „konfigurierbar“ (z. B.: Option `-l` global ändern?)
- Skript skaliert nicht (mehr als die drei Ordner angucken)

- Studium der Manpage zu `ls` verrät: Es gibt hilfreiche Optionen

- `-t`: sortiert nach Zeitpunkt der letzten Änderung
- `-r`: kehrt Sortierreihenfolge um

Mit `ls -ltr` erhalten wir die Dateien mit den jüngsten Änderungen am Ende der Ausgabe

- Zur Konfigurierbarkeit führen wir eine Variable `LS_OPTS` ein, die die Optionen speichert:

```
LS_OPTS="-ltr"
```

(Zugriff darauf mit `$LS_OPTS`)

6

7

- Angepasste Skriptdatei `info.sh` :

```
1 #!/bin/bash
2 # info.sh, Beschreibung folgt
3 LS_OPTS="-ltr"
4 ls $LS_OPTS /var/log
5 ls $LS_OPTS /var/spool/mail
6 ls $LS_OPTS /var/spool/cups
```

- Ziel: Auch die Liste der Verzeichnisse in eine Variable packen, geht z. B. mit

```
FOLDERS="/var/log_/var/spool/mail_/var/spool/cups"
```

Aber: Wie darauf zugreifen?

- Einfachste Lösung an dieser Stelle:

```
ls $LS_OPTS $FOLDERS
```

(Das geht auch besser, später mehr.)

8

9

- Angepasste Skriptdatei `info.sh` :

```

1 #!/bin/bash
2 # info.sh, Beschreibung folgt
3 LS_OPTS="-ltr"
4 FOLDERS="/var/log /var/spool/mail /var/spool/cups"
5 ls $LS_OPTS $FOLDERS

```

- Das Skript führt dann effektiv nur einen einzigen Befehl aus, nämlich:

```
ls -ltr /var/log /var/spool/mail /var/spool/cups
```

10

- Es fehlt noch: Kommentierung des Skripts
→ Praktikum, Dokument „Bash Style Guide und Kodierungsrichtlinie“

```

1 #!/bin/bash
2 #=====
3 #
4 #     FILE:  info.sh
5 #
6 #     USAGE: info.sh
7 #
8 #     DESCRIPTION:  Execute ls -ltr for three directories.
9 #                   The list of directories can be changed via the FOLDERS variable
10 #
11 #     OPTIONS:  none
12 #     REQUIREMENTS:  ---
13 #     BUGS:  ---
14 #     NOTES:  ---
15 #     AUTHOR:  Hans-Georg Eber (hge), esser.hans-georg@fh-swf.de
16 #     COMPANY:  FH Südwestfalen, Iserlohn
17 #     VERSION:  1.4
18 #     CREATED:  29.09.2019 - 21:29
19 #     REVISION:  08.10.2020 - 22:42
20 #=====

```

11

Problem/Lösung 1: Neues Wissen

- Skripte starten mit `#!/bin/bash`
- Mehrere Befehle in Shell-Skript zusammenfassen (Hintereinanderausführung)
- `chmod a+x` macht Dateien ausführbar (mehr dazu später)
- Lesen von Manpages ist immer ein guter Plan
- `ls`-Optionen `-t` (sortiert nach Änderungsdatum) und `-r` (kehrt Reihenfolge um)
- Einsatz von Variablen

12

Problem 2: Dateien umbenennen

- Ein Server erstellt täglich um 0:00 Uhr eine Log-Datei und speichert sie in `/var/log/server/`.
- Admin möchte in diesem Verzeichnis mehrere Dateien mit Namen der Form `log-29.09.2020.txt`, `log-30.09.2020.txt` usw. in `log-2020-09-29.txt`, `log-2020-09-30.txt` usw. umbenennen → bessere Sortierung.
- Bisherige Vorgehensweise: 1x pro Woche manuell die Dateien umbenennen:

```
mv log-25.09.2020.txt log-2020-09-25.txt
mv log-26.09.2020.txt log-2020-09-26.txt
mv log-27.09.2020.txt log-2020-09-27.txt
mv log-28.09.2020.txt log-2020-09-28.txt
mv log-29.09.2020.txt log-2020-09-29.txt
...
```

13

- Ziel: Prozess der Umbenennung automatisieren, bereits korrekt benannte Dateien in Ruhe lassen
- Schritte zur Lösung
 - Wir werden eine *Schleife* brauchen
 - Bestandteile des Dateinamens (Jahr, Monat, Tag) identifizieren und separat verarbeiten
 - Wenn Skript fertig ist, für automatische Ausführung sorgen, z. B. um 1:00 Uhr

14

Lösung 2: Schleifen

- Shell bietet **for**-Schleife:

```
1 #!/bin/bash
2 cd /var/log/server          # Wechsel in rich-
3                             # tigen Ordner
4 for dateiname in log*.txt  # alle Log-Dateien
5 do
6     echo $dateiname
7 done
```

- `dateiname` ist Variable, Zugriff mit `$dateiname`
- Konvention:
 - Variablen (die wirklich viele verschiedene Werte annehmen) klein schreiben;
 - Konstanten (wie in Problem 1 `LS_OPTS`) GROSS

15

Lösung 2: Schleifen

- Skript erzeugt Ausgabe:

```
log-2020-09-18.txt      # alt, schon umbenannt
log-2020-09-19.txt
log-2020-09-20.txt
log-21.09.2020.txt     # ab hier neu
log-22.09.2020.txt
```

- Wir wollen nur die neuen Dateien verarbeiten, darum Muster `log*.txt` anpassen
- Dateinamen haben die Form `log-DD.MM.YYYY.txt` → Muster: `log-???.???.?????.txt`
- Wie können wir jetzt die Komponenten DD, MM, YYYY aus dem Dateinamen ziehen?

16

- Der *Stream Editor* **sed** kann so was
- Erste Tests direkt im Terminal:

```
echo log-22.09.2020.txt | sed -E \  
  's/log-(..)(..)(....).txt/log-\3-\2-\1.txt/'
```

 erzeugt gewünschte Ausgabe `log-2020-09-22.txt`.
- Das bauen wir schon mal in das Skript ein ...

- Neue Version mit **sed**:

```
1 #!/bin/bash  
2 cd /var/log/server  
3 for dateiname in log-???.???.?????.txt  
4 do  
5   echo -n "$dateiname_>_>"  
6   echo $dateiname | sed -E \  
7     's/log-(..)(..)(....).txt/log-\3-\2-\1.txt/'  
8 done
```
- gibt Zeilen der folgenden Form aus:

```
log-22.09.2020.txt -> log-2020-09-22.txt
```

Einschub zur sed-Syntax

- `(...)` definiert eine Gruppe
- `\1, \2, \3` usw.: Bezug auf 1., 2., 3. Gruppe
- Achtung: Zweierlei Syntax für reguläre Ausdrücke, unterschieden durch Option `-E` oder `-e`. Im Beispiel von eben (vereinfacht):

```
sed -E 's/log-(..)(..).txt/log-\2-\1.txt/'  
sed -e 's/log-\(..\)\.\(..\).txt/log-\2-\1.txt/'
```

`-e` ist die „klassische“ Option, hier muss man die Klammern *escapen* (`\` voranstellen)

Einschub: Reguläre Ausdrücke

sed -E	sed -e	
.	.	beliebiges Zeichen
^	^	Zeilenanfang
[xyz]	[xyz]	eines der Zeichen x, y, z
[^xyz]	[^xyz]	beliebiges Zeichen <i>außer</i> x, y, z
\$	\$	Zeilenende
(...)	\(... \)	Gruppenbildung
?	?	„einmal oder keinmal“
*	*	beliebig oft (auch 0-mal)
+	\+	beliebig oft (aber mindestens 1-mal)
{n}	\{n\}	genau n-mal
{i,j}	\{i,j\}	zwischen i- und j-mal
R ₁ R ₂	R ₁ R ₂	R ₁ oder R ₂

- Weiter mit der Lösung: Skript gibt bisher alten und neuen Namen aus
- Wir brauchen aber zwei Argumente für mv
- Lösung: \$(...)

```
bash:~$ echo Hallo
Hallo
bash:~$ wert=$( echo Hallo )
bash:~$ echo $wert
Hallo
```

→ anwenden auf das Skript führt zu Lösung

21

- Neue Version mit \$(...):

```
1 #!/bin/bash
2 # rename.sh
3 cd /var/log/server
4 for dateiname in log-???.???.?????.txt
5 do
6     neuename=$( echo $dateiname | sed -E \
7     's/log-(..)(..)(....).txt/log-\3-\2-\1.txt/' )
8     mv $dateiname $neuename
9 done
```

22

- Automatisieren: Google-Suche nach „Skript regelmäßig aufrufen“ → cron
- Manpage zu cron → /etc/crontab

```
1 SHELL=/bin/sh
2 PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
3 MAILTO=root
4 #
5 # check scripts in cron.hourly, cron.daily, cron.
6 #   weekly, and cron.monthly
7 */15 * * * * root test -x /usr/lib/cron/run-crons
   && /usr/lib/cron/run-crons >/dev/null 2>&1
```

23

```
*/15 * * * * root test -x /usr/lib/cron/run-
   crons && /usr/lib/cron/run-crons >/dev/null 2>&1
```

Felder durch Leerzeichen getrennt

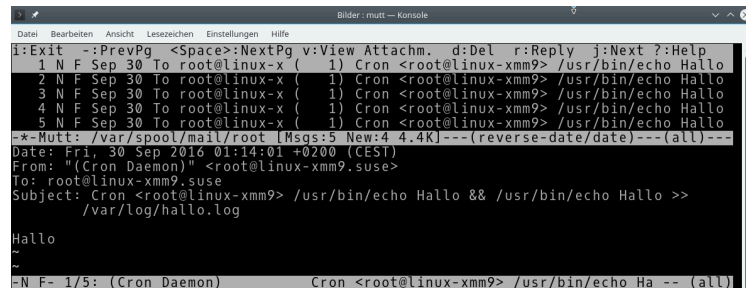
1. */15 – Minute (hier: alle 15 Minuten)
2. * – Stunde (hier: zu jeder Stunde)
3. * – Tag (1–31, hier: jeder Tag)
4. * – Monat (1–12, hier: jeder Monat)
5. * – Wochentag (0–7, 0 = 7 = Sonntag, hier: jeder)
6. root – Benutzer, mit dessen Rechten der Befehl läuft
7. test -x ... – Befehl

Also neuer Eintrag für täglich 1:00 Uhr:

```
0 1 * * * root /root/bin/rename.sh >/dev/null 2>&1
```

24

- Was macht `>/dev/null 2>&1` ?
 - 1 und 2 sind sog. Dateideskriptor-Nummern für Standardausgabe (1, stdout) und Standardfehlerausgabe (2, stderr)
 - `>/dev/null` leitet die Standardausgabe nach `/dev/null`
 - `2>&1` leitet Standardfehlerausgabe in die Standardausgabe um, also auch nach `/dev/null`
- Und wozu?
 - Ohne die Umleitung würde der Befehl evtl. Ausgaben erzeugen, die dann an unerwarteten Stellen auftauchen



```

i:Exit  -:PrevPg  <Space>NextPg v:View Attachm. d:Del  r:Reply  j:Next ? :Help
1 N F Sep 30 To root@linux-x ( 1) Cron <root@linux-xmm9> /usr/bin/echo Hallo
2 N F Sep 30 To root@linux-x ( 1) Cron <root@linux-xmm9> /usr/bin/echo Hallo
3 N F Sep 30 To root@linux-x ( 1) Cron <root@linux-xmm9> /usr/bin/echo Hallo
4 N F Sep 30 To root@linux-x ( 1) Cron <root@linux-xmm9> /usr/bin/echo Hallo
5 N F Sep 30 To root@linux-x ( 1) Cron <root@linux-xmm9> /usr/bin/echo Hallo
--Mutt: /var/spool/mail/root [Msgs:5 New:4 4.4K]---(reverse-date/date)---(all)---
Date: Fri, 30 Sep 2016 01:14:01 +0200 (CEST)
From: "(Cron Daemon)" <root@linux-xmm9.suse>
To: root@linux-xmm9.suse
Subject: Cron <root@linux-xmm9> /usr/bin/echo Hallo && /usr/bin/echo Hallo >>
/var/log/hallo.log

Hallo
~
-N F- 1/5: (Cron Daemon) Cron <root@linux-xmm9> /usr/bin/echo Ha -- (all)

```

25

- Wildcards * und ?
- `\` am Ende umbricht Befehlszeilen
- **echo** gut für erste Tests
- **sed** mit regulären Ausdrücken verwenden
- Befehlsergebnis mit `$(...)` verwenden
- **for**-Schleife programmieren
- cron-Job über crontab erstellen
- Umleitung mit `>` und `>&`; Standardausgabe stdout (1), Standardfehlerausgabe stderr (2)

26

Problem 3: Kein Editor

Problem 3: Kein Editor

- Nach Login auf obskurer Unix-Maschine: kein Editor (nicht mal vi)
- Ziel: Mit Bordmitteln der Shell selbst einen Editor schreiben
- Werkzeuge: **read**, wc, Ein- und Ausgabe-Umleitungen
- Teilaufgaben:
 - Datei zeilenweise einlesen
 - Zeilen in der Shell in Variablen speichern
 - Bearbeiten einzelner Zeilen (im Speicher)
 - Speichern und beenden

27

- Erste Zeile lesen durch Umleitung (<):

```
$ cat a.txt
Zeile 1
Zeile 2
Zeile 3
$ read v < a.txt
$ echo $v
Zeile 1
```

- Mehrfaches Lesen auf diese Weise: klappt nicht

```
$ read v1 < a.txt; read v2 < a.txt;
$ echo $v1; echo $v2
Zeile 1
Zeile 1
```

28

- Lösung: Gruppieren der **read**-Befehle mit {}
- Wichtig: Jeden Befehl in Gruppe (auch den letzten!) mit ; abschließen

```
$ { read v1; read v2; } < a.txt;
$ echo $v1; echo $v2
Zeile 1
Zeile 2
```

29

- **read** varname liest eine Zeile aus Standardeingabe und schreibt sie nach varname
- Problem:

```
$ read testvar
Eingabe mit vielen Blanks
$ echo $testvar
Eingabe mit vielen Blanks
$ echo "$testvar"
Eingabe mit vielen Blanks
```

→ **read** entfernt führende Blanks; **echo** ohne "" entfernt alle

30

Lösung (findet man durch Lesen der Manpage und Ausprobieren):

- **read** ohne Variablennamen aufrufen; speichert Ergebnis in \$REPLY
- Dort sind auch führende Leerzeichen enthalten
- Bei Zugriff immer "\$REPLY" verwenden

```
$ read; testvar="$REPLY"
Eingabe mit vielen Blanks
$ echo "$testvar"
Eingabe mit vielen Blanks
```

31

- **read** entfernt Backslashes (\)

```
$ read; testvar="$REPLY"
"String mit \"String\" in der Mitte"
$ echo "$testvar"
"String mit "String" in der Mitte"
```

- Lösung: Option -r

```
$ read -r; testvar="$REPLY"
"String mit \"String\" in der Mitte"
$ echo "$testvar"
"String mit \"String\" in der Mitte"
```

32

- Nächste Schwierigkeit: Anzahl der Eingabezeilen variabel
- Anzahl mit wc feststellen:

```
NAME
  wc -- word, line, character, and byte count

SYNOPSIS
  wc [-clmw] [file ...]

DESCRIPTION
  The wc utility displays the number of lines, words, and bytes contained in each input file, or standard input (if no file is specified) to the standard output. A line is defined as a string of characters delimited by a <newline> character. Characters beyond the final <newline> character will not be included in the line count.

  A word is defined as a string of characters delimited by white space characters. White space characters are the set of characters for which the iswspace(3) function returns true. If more than one input file is specified, a line of cumulative counts for all the files is displayed on a separate line after the output for the last file.

  The following options are available:

  -c   The number of bytes in each input file is written to the standard output. This will cancel out any prior usage of the -m option.

  -l   The number of lines in each input file is written to the standard output.
```

```
lines=$( wc -l < filename )
```

33

- Shell bietet auch Array-Variablen
- Setzen mit arrayvar[0]=..., arrayvar[1]=... usw.
- innerhalb Schleife (über i): arrayvar[\$i]=...

- Erster Ansatz:

```
for i in 0 1 2 3 4; do
  read -r
  zeile[$i]="$REPLY"
done < filename # Umleitung: für ganze Schleife
```

- Problem: variable Zeilenzahl ...

34

- Tool seq erzeugt Sequenz von a bis b:

```
$ seq 0 3
0
1
2
3
```

- Ausgabe von seq wieder mit \$(...)-Trick verarbeiten:

```
$ for i in $( seq 0 3 ); do echo Zeile $i; done
Zeile 0
Zeile 1
Zeile 2
Zeile 3
```

35

- Wenn wc mit `lines=$(wc -l < filename)` die Zeilenzahl geliefert hat, erzeugt `for i in $(seq 0 $lines)` einen Eintrag zu viel!
- Rechnen in der Shell: `$(lines-1)` (zwingend mit `[]`)
`$ echo $lines vs. $(lines-1)`
`3 vs. 2`
`$ seq 0 $(lines-1)`
`0`
`1`
`2`
- Alles zusammen: `for i in $(seq 0 $(lines-1))`

36

```
#!/bin/bash
filename=$1          # erstes Argument: Dateiname
lines=$( wc -l < $filename)
for i in $( seq 0 $(lines-1) ); do
    read -r          # zeilenweise einlesen ...
    zeilen[$i]=$REPLY # ... und zuweisen
done < $filename    # Umleitung gilt für ganze
                   # Schleife

# Testweise wieder ausgeben:
for i in $( seq 0 $(lines-1) ); do
    printf "%3d_\u00a0" $i; echo "${zeilen[i]}"
done

→ ${...} bei Array-Zugriff nötig! Falsch: $zeilen[$i]
```

37

- Es geht auch eleganter ohne wc
- Shell kennt **while**-Schleife:

```
#!/bin/bash
filename=$1          # 1. Argument: Dateiname
i=0                 # Zähler initialisieren
while read -r; do   # Schleife läuft, bis
    read fehlschlägt (EOF)
    zeilen[$i]=$REPLY # gelesenen Wert zuweisen
    i=$((i+1))       # und erhöhen
done < $filename
lines=$i            # Zeilenzahl merken
# Testweise wieder ausgeben:
...                # wie vorher
```

38

- Jeder Befehl erzeugt einen Exit-Code
- lässt sich über `$?` abfragen: 0 (ok) oder $\neq 0$ (nicht ok)

```
$ cat datei.txt # vorhandene Datei
...
$ echo $?
0
$ cat bhdvbjdfh.txt # nicht vorhanden
cat: bhdvbjdfh.txt: No such file or directory
$ echo $?
1
```
- **read** gibt 0 zurück, solange Zeile gelesen wurde
- Bei EOF: Rückgabe 1
- **while command** wertet Exit-Code aus. Abbruch wenn $\neq 0$

39

Ergebnis einer Programmausführung mit `&&` und `||` verwerten:

- `cmd1 && cmd2`
= Kommando `cmd2` nur dann ausführen, wenn `cmd1` erfolgreich war (Exitcode 0)
- `cmd1 || cmd2`
= Kommando `cmd2` nur dann ausführen, wenn `cmd1` *nicht* erfolgreich war (Exitcode \neq 0)

Beispiele:

- `test -f README && cat README`
(Datei `README` nur ausgeben, wenn sie existiert)
- `zeile=$(read) || zeile="Leere_Eingabe"`
(Versuch, eine Zeile einzulesen. Falls keine Eingabe erfolgt, Variable auf den String "`Leere_Eingabe`" setzen)

40

- Alternativ zu
`x=$((x+1))`
ist auch
`((x++))` oder `((x=x+1))`
möglich → vgl. Praktikum, Kap. 3.4, `create_dirs.sh`

41

Lösung 3: Editor-Funktionen (1)

- Idee: Endlosschleife mit Abfrage einer Zeilennummer
- Zeile ausgeben und mit `read` neue Version einlesen

```
...
while true; do # true gibt immer 0 (ok) zurück
  echo -n "Zeilennummer:_"; read input
  echo "Alt:_${zeilen[input]}"
  echo -n "Neu:_"; read -r
  zeilen[$input]="$REPLY"
done
```

- `echo`-Option `-n`: kein Zeilenumbruch

42

Lösung 3: Editor-Funktionen (2)

- Speichern, Abbruch, Ausgabe (zur Kontrolle)
- Programm soll Befehle `wq`, `q!` und `show` verstehen
- Test mit `if ... then ... fi`

```
...
echo -n "Zeilennummer:_"; read input
if [ $input == "wq" ]; then
  ...
fi
if [ $input == "q!" ]; then
  ...
fi
```

- `[...]` ist Kurzform für `test ...` (→ `help test`)

43

```

File operators:
-a FILE      True if file exists.
-b FILE      True if file is block special.
-c FILE      True if file is character special.
-d FILE      True if file is a directory.
-e FILE      True if file exists.
-f FILE      True if file exists and is a regular file.
-g FILE      True if file is set-group-id.
-h FILE      True if file is a symbolic link.
-L FILE      True if file is a symbolic link.
-k FILE      True if file has its 'sticky' bit set.
-p FILE      True if file is a named pipe.
-r FILE      True if file is readable by you.
-s FILE      True if file exists and is not empty.
-S FILE      True if file is a socket.
-t FD        True if FD is opened on a terminal.
-u FILE      True if the file is set-user-id.
-w FILE      True if the file is writable by you.
-x FILE      True if the file is executable by you.
-O FILE      True if the file is effectively owned by you.
-G FILE      True if the file is effectively owned by your group.
-N FILE      True if the file has been modified since it was last read.

FILE1 -nt FILE2 True if file1 is newer than file2 (according to modification date).

FILE1 -ot FILE2 True if file1 is older than file2.

FILE1 -ef FILE2 True if file1 is a hard link to file2.

```

44

```

String operators:

-z STRING    True if string is empty.

-n STRING    True if string is not empty.

STRING1 = STRING2
              True if the strings are equal.
STRING1 != STRING2
              True if the strings are not equal.
STRING1 < STRING2
              True if STRING1 sorts before STRING2 lexicographically.
STRING1 > STRING2
              True if STRING1 sorts after STRING2 lexicographically.

Other operators:

-o OPTION    True if the shell option OPTION is enabled.
! EXPR      True if expr is false.
EXPR1 -a EXPR2 True if both expr1 AND expr2 are true.
EXPR1 -o EXPR2 True if either expr1 OR expr2 is true.

arg1 OP arg2 Arithmetic tests. OP is one of -eq, -ne, -lt, -le, -gt, or -ge.

Arithmetic binary operators return true if ARG1 is equal, not-equal, less-than,
less-than-or-equal, greater-than, or greater-than-or-equal than ARG2.

```

45

Einschub: Beispiele zu test

```

$ test "A" == "B"; echo $?
1
$ test "A" == "A"; echo $?
0
$ test "A" -eq "A"; echo $?
-bash: test: A: integer expression expected
2
$ test "A" != "A"; echo $?
1
$ [ "A" != "A" ]; echo $?
1
$ [ 3 -lt 5 ]; echo $?
0
$ [ 3 -gt 5 ]; echo $?
1
$ [ 3 -eq 5 ]; echo $?
1

```

46

Lösung 3: Editor-Funktionen (3)

- Speichern mit wq (in Anlehnung an das vi-Kommando wq)
 - Einfach alle Zeilen raus schreiben und beenden
- ```

if [$input == "wq"]; then
 for i in $(seq 0 $[lines-1]); do
 printf "%s\n" "${zeilen[i]}"
 done > tmp.out # in temp. Datei speichern
 break # break beendet die Schleife
fi

```
- Nach Verlassen der Schleife endet auch das Programm

47

- Beenden mit q! (in Anlehnung an das vi-Kommando q!)
- Noch einfacher: **break** und Programmende
 

```
if [$input == "q!"]; then
 break # break beendet die Schleife
fi
```
- Kontrollausgabe mit show (implementieren wir als Funktion)
 

```
if [$input == "show"]; then
 show # ruft Funktion show() auf
 continue # diesen Schleifendurchlauf
fi # beenden
```

48

- Es fehlt noch die Funktion show()
- Funktionsdeklaration: zwei gleichwertige Varianten
 

```
function fname { cmd1; cmd2; ... ; }
fname() { cmd1; cmd2; ... ; }
```
- Funktion kann Parameter über \$1, \$2 etc. auswerten
 

```
function show {
 for i in $(seq 0 $[lines-1]); do
 printf "%3d:_" $i # Zeilennummer ausgeben
 echo "${zeilen[i]}"
 done
}
```

(show erhält keine Argumente; mögliche Anpassung: Start- und Endzeile)

49

## Lösung 3: Alles zusammen (1)

```
1 #!/bin/bash
2 #
3 # Einfacher Zeilen-Editor
4
5 function show { for i in $(seq 0 $lines); do
6 printf "%3d:_" $i; echo "${zeilen[i]}"
7 done
8 }
9
10 function fehler { echo "$0: Fehler: $1"; exit 1; }
11
12 # Test auf Fehler: Kein Dateiname (oder zu viele Argumente)
13 if [$# -ne 1]; then
14 fehler "Aufruf: $0 filename."
15 fi
16
17 filename=$1
18
19 # Test auf Fehler: Datei nicht vorhanden
20 if [! -f $filename]; then
21 fehler "Datei \"$filename\" existiert nicht."
22 fi
23
24 lines=$(wc -l < $filename) # Anzahl Zeilen
25
26 for i in $(seq 0 $lines); do
27 read -r; zeilen[$i]=$REPLY # zeilenweise einlesen
28 done < $filename
```

50

## Lösung 3: Alles zusammen (2)

```
29 show
30
31 while true; do
32 echo -n "Zeilennummer oder 'wq', 'q!' oder 'show' eingeben:_"
33 read linenummer
34 if ["$linenummer" == "Z"]; then
35 continue
36 fi
37 if [$linenummer == "q!"]; then
38 break
39 fi
40 if [$linenummer == "wq"]; then
41 for i in $(seq 0 $lines); do
42 printf "%s\n" "${zeilen[i]}"
43 done > tmp.out
44 break
45 fi
46 if [$linenummer == "show"]; then
47 show
48 continue
49 fi
50
```

51

```

51 # Pruefen, ob Eingabe eine Zahl ist
52 zahlen='^[0-9]+$'
53 if ! [[$linenumber =~ $zahlen]] ; then
54 echo "Fehler: „Weder „Zeilennummer„noch„Editor-Kommando"
55 continue
56 fi
57
58 # Pruefen, ob Zeilennummer gueltig
59 if [$linenumber -gt $lines]; then
60 echo "Fehler: „Zeile" $linenumber "existiert„nicht."
61 continue
62 fi
63
64 echo Ersetze Zeile $linenumber
65 printf "%3d:„ $linenumber; echo "${zeilen[linenumber]}"
66 printf "%3d:„ $linenumber
67 read; zeilen[$linenumber]="$REPLY"
68 echo Zeile $linenumber ersetzt.
69 done

```

52

- „Bearbeiten“ der Zeile mit **read** unkomfortabel
  - Schöner wäre: Zeileneditor wie in der Shell, aber mit vorausgefülltem Inhalt
  - Ziel: einzelne Tastendrucke interpretieren, dabei auch Escape-Sequenzen verarbeiten,  
z. B. Cursor-nach-links: Esc, [, D
  - Folgende Folien präsentieren zwei mögliche Implementierungen eines Programms `editline.c`
    - Variante ohne Nutzung spezieller Bibliotheken
    - Variante mit Nutzung der `readline`-Bibliothek
- (C-Programme gehören *nicht* zum offiziellen Vorlesungsstoff.)

53

## Lösung 3: Ergänzung, editline.c (1)

```

#define cursorforward(x) printf("\033[%dC", (x))
#define cursorbackward(x) printf("\033[%dD", (x))
#define MAX_LEN 80

int main (int argc, char *argv[]) {
 struct termios old, new;
 tcgetattr(STDIN_FILENO, &old); new = old; new.c_lflag &= ~(ICANON | ECHO);
 tcsetattr(STDIN_FILENO, TCSANOW, &new);

 char c, buffer[81];
 if (argc==2) strncpy (buffer, argv[1], 80); else buffer[0] = 0;
 char len = strlen(buffer); int pos = 0; printf ("%s", buffer); cursorbackward (len);

 for (;;) {
 c = getchar ();
 if (c == 27) { // Esc-Sequenz verarbeiten
 getchar (); c = getchar ();
 switch (c) { case 'D': // Esc, [, D = cursor-left
 if (pos > 0) { cursorbackward (1); pos--; }
 break;
 case 'C': // ESC, [, C = cursor-right
 if (pos < len) { cursorforward (1); pos++; }
 break;
 default: break;
 }
 }
 else { // normales Zeichen verarbeiten
 ...

```

54

## Lösung 3: Ergänzung, editline.c (2)

```

... // normales Zeichen verarbeiten
switch (c) { case '\n': // enter = Bearbeitung fertig
 printf ("\n"); fprintf (stderr, "%s\n", buffer); // auf stderr schreiben
 tcsetattr(STDIN_FILENO, TCSANOW, &old); exit (0);
 case '\t': // tab = ein Zeichen einfüegen
 if (len < MAX_LEN) {
 for (int i = len+1; i >= pos; i--) buffer[i+1] = buffer[i]; // move ->
 buffer[pos] = '\t'; len++;
 printf ("%s", buffer+pos); cursorbackward (len-pos);
 }
 break;
 case 127: // delete = ein Zeichen loeschen
 if (pos > 0) {
 for (int i = pos-1; i < len; i++) buffer[i] = buffer[i+1]; // move <-
 len--; pos--; cursorbackward (1);
 printf ("%s", buffer+pos); cursorbackward (len-pos+1);
 }
 break;
 default:
 if (pos < MAX_LEN) {
 printf ("%c", c); buffer[pos++] = c;
 if (pos > len) len++;
 }
 break;
}
}
}
}

```

55

... oder mit readline:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <readline/readline.h> // mit -lreadline kompilieren!
#define MAX_LEN 80

static char *deftext = NULL;
static int set_deftext () { if (deftext) rl_insert_text (deftext); return 0; }

char *my_readline (char *prompt, char *prefill) {
 if (prefill != NULL) {
 deftext = prefill; // Zeileninhalt vorbelegen
 rl_startup_hook = set_deftext; // readline soll set_deftext aufrufen
 }
 char *line = readline (prompt);
 rl_startup_hook = NULL; // readline wieder auf Normalbetrieb
 return line;
}

int main (int argc, char *argv[]) {
 char *buffer; if (argc==2) buffer = argv[1]; else buffer = NULL;
 buffer = my_readline ("", buffer);
 fprintf (stderr, "%s\n", buffer); // neue Zeile auf stderr schreiben
}
```

56

Wenn man noch folgende Funktionen ergänzt, hat man einen brauchbaren Zeileneditor:

- Zeilen löschen, Leerzeilen einfügen
- Copy & Paste (für Zeilen und Blöcke von Zeilen)
- Suchen und Ersetzen
- Speichern unter Originalname, vorher Backup anlegen

→ alles recht einfach umzusetzen

57

## Problem/Lösung 3: Neues Wissen (1)

- **read**, **read -r**, \$REPLY, wc, **printf**, seq
- stdin, stdout, stderr mit <, > und 2> umleiten
- Befehlsgruppe mit {}
- **function** {}
- Exitcode, \$?, **if ... then ... fi**
- [ ... ], **test**, [ -f datei ]
- Array-Variablen, array[\$index]=..., \${array[index]}
- **while ... do ... done**, **continue**, **break**
- Ganze Schleifen umleiten: < o. ä. hinter **done**
- \$var vs. "\$var"
- &&, ||
- \${x+1}, ((x++))

58

## Problem/Lösung 3: Neues Wissen (2)

**for-Schleife:**

```
for variable in liste; do
 cmd1
 cmd2 $variable
 ...
done
```

**if-then-else:**

```
if testcmd1; then
 cmd1
elif testcmd2; then # optional
 cmd2
else # optional
 cmd3
fi
```

**while-Schleife:**

```
while testcmd; do
 cmd1
 cmd2
 ...
done
```

- Schleife abbrechen: **break**
- einen Durchlauf abbrechen: **continue**

59



### Problem 4: Verzeichnis-Abgleich

---

- Sie verwalten auf zwei Rechnern gleichartige Ordnerstrukturen unterhalb ~/bs2/
- Diese werden öfter manuell (teil-)synchronisiert
- Frage: Welche Dateien unterscheiden sich?

60

### Problem 4: Verzeichnis-Abgleich

Ziel: Ausgabe der Form

```
[1] tex/tmp/swf-bs2-ws2020-b.tex
[1] pc1: 43772 13 0kt 11:43 94c55ba72a0289579c1
[1] pc2: 48102 20 0kt 21:12 a7d90cc7491966731a6
[2] tex/tmp/swf-bs2-ws2020-b.pdf
[2] pc1: 459549 19 0kt 20:01 678a767b87cdc6786dc
[2] pc2: 459549 20 0kt 21:35 a78676b87g6786fe786
[3] tex/tmp/swf-bs2-ws2020-b.log
[3] pc1: 9101 19 0kt 20:01 f10093176a786a67316
[3] pc2: <missing>
[4] tex/newtests/tmpfile.txt
[4] pc1: <missing>
[4] pc2: 701 20 0kt 23:12 9abbd78a9798a6sq133
```

61

### Problem 4: Verzeichnis-Abgleich

- Ausgabeformat mit [ 1], [ 2] usw. ist auf Situation mit mehr als zwei Rechnern erweiterbar
- Konfiguration für Tool soll wie folgt gespeichert werden:
 

```
machine=username1@pcname1:basedir1
machine=username2@pcname2:basedir2
...
```
- statt username1@pcname1 auch Angabe local möglich

62

- Zerlegen in Teilaufgaben
- Untersuchung eines Rechners
  - Dateiliste erstellen
  - Für jede Datei Hash berechnen und notieren
- Hash-Listen der Rechner sammeln und auswerten

Wir brauchen:

- Tool zur Hash-Berechnung (→ md5sum)
- Tools für Remote-Befehlsausführung und Remote-Dateizugriff (→ ssh, scp)

```
#!/bin/bash
CONFIG_FILE=machines.conf
MACHINE_PATTERN=machine=[^@]+@[^:]+:.*
example 1: machine=username1@pcname1:basedir1
example 2: machine=local:basedir1
i=0
while read; do
 if [[$REPLY =~ $MACHINE_PATTERN]]; then
 machine[$i]="$REPLY"; let i++
 fi
done < $CONFIG_FILE
machines=$i

for ((i=0; i<$machines; i++)); do
 get_data ${machine[i]} # <-- Funktion!
done
```

63

64

### Einschub: Alternative For-Schleife

```
#!/bin/bash

echo "C-style_for_loop:"
for ((i=0; i<10; i++)); do
 echo -n "$i_"
 if [$i -eq 4]; then
 i=7
 fi
done
echo

echo "bash-style_for_loop:"
for i in {0..9}; do
 echo -n "$i_"
 if [$i -eq 4]; then
 i=7
 fi
done
echo
```

- Programm erzeugt Ausgabe:  
 C-style for loop:  
 0 1 2 3 4 8 9  
 bash-style for loop:  
 0 1 2 3 4 5 6 7 8 9
- {0..9} ist Alternative zu \$( seq 0 9 )

65

### Lösung 4: Funktion get\_data (1)

get\_data muss ...

- das Argument \$1 zerlegen:  
 machine=username1@pcname1:basedir1  
 → username1, pcname1, basedir1  
 machine=local:basedir1 → basedir1
- Trennzeichen sind =@:
- Hier hilft die Shell-Variable \$IFS (internal field separator)  
 \$ string='machine=username1@pcname1:basedir1'  
 \$ IFS="=@:"; for i in \$string; do echo \$i; done  
 machine  
 username1  
 pcname1  
 basedir1

66

Typisches Beispiel für \$IFS-Einsatz: Arbeiten mit /etc/passwd

```
student@linux:~> tail -4 /etc/passwd
student:x:1000:100:Student:/home/student:/bin/bash
svn:x:485:483:user for svnserve:/srv/svn:/sbin/nologin
student01:x:1001:100::/home/student01:/bin/bash
student@linux:~> tail -1 /etc/passwd > pw; read var < pw
student@linux:~> echo $var
student01:x:1001:100::/home/student01:/bin/bash
student@linux:~> IFS=":"; for i in $var; do echo $i; done
student01
x
1001
100

/home/student01
/bin/bash
student@linux:~> IFS="$old_IFS"
```

67

Was steht standardmäßig in \$IFS? Ausgabe von echo \$IFS nicht hilfreich, aber ...

```
$ echo -n "$IFS" | hexdump -C
00000000 20 09 0a | ..|
```

- 0x20 = 32 = Leerzeichen
- 0x09 = 9 = Tabulator (\t)
- 0x0a = 10 = Zeilenumbruch (\n)

68

## Lösung 4: Funktion get\_data (2)

```
function get_data {
 old_IFS="$IFS"; IFS="=:@"
 index=0
 for arg in $1; do
 case $index in
 0) ;; # "machine"
 1) user=$arg;; # "username1"
 2) host=$arg;; # "pcname1"
 3) dir=$arg;; # "basedir1"
 esac
 let index++
 done
 IFS="$old_IFS"
 if [$index -eq 3]; then # local
 dir=$host; unset user host
 fi

 if test -n "$user"; then
 echo Remote-Funktion nicht installiert
 return
 fi
}
```

69

## Lösung 4: Funktion get\_data (3)

```
...
filename_prefix="${dir//\//:}"
current_dir="$PWD"
cd "$dir"
find . -type f | cut -c3- > ${current_dir}/${filename_prefix}.files
cd "$current_dir"; rm -f ${filename_prefix}.tmpdata
while read file; do
 sum=$(md5sum "$dir/$file" | cut -f1 -d" ")
 date=$(stat -c '%y' "$dir/$file" | cut -c1-16 | tr ' :' '-.')
 size=$(stat -c '%s' "$dir/$file")
 echo "$file!$sum!$date!$size" >> ${filename_prefix}.tmpdata
done < ${filename_prefix}.files

sort ${filename_prefix}.tmpdata > ${filename_prefix}.data
cut -f1,2 -d! < ${filename_prefix}.data > ${filename_prefix}.sums
}
```

- Warum while read file; ... < XXX.files statt for file in \$(cat XXX.files) ...?
- Was machen md5sum und stat?

70

```

student@linux:~> ls -l main.sh
-rwxr-xr-x 1 student users 1485 21. Okt 01:31 main.sh
student@linux:~> md5sum main.sh
c70036627f7800fb7a9f703832cc1c76 main.sh
student@linux:~> md5sum main.sh | cut -f1 -d" "
c70036627f7800fb7a9f703832cc1c76
student@linux:~> stat main.sh
 Datei: "main.sh"
 Größe: 1485 Blöcke: 8 EA Block: 4096 reguläre Datei
Gerät: 803h/2051d Inode: 131545 Verknüpfungen: 1
Zugriff: (0755/-rwxr-xr-x) Uid: (1000/ student) Gid: (100/ users)
Zugriff : 2020-10-21 01:31:32.452832973 +0200
Modifiziert: 2020-10-21 01:31:30.788832868 +0200
Geändert : 2020-10-21 01:31:30.788832868 +0200
Geburt : -
student@linux:~> stat -c '%y' main.sh
2020-10-21 01:31:30.788832868 +0200
student@linux:~> stat -c '%y' main.sh | cut -c1-16
2020-10-21 01:31
student@linux:~> stat -c '%y' main.sh | cut -c1-16 | tr ' :' '-'
2020-10-21-01.31
student@linux:~> stat -c '%s' main.sh
1485

```

71

Skript erzeugt z. B. für Ordner /home/student/tmp eine Datei  
:home:student:tmp.data mit Inhalt der Form

```

a/fstab!94016421866bedeeb88276d027108db9!2020-10-21-02.36!237
a/readme.txt!cfba969536b62ee7757892ca94755ea8!2020-10-21-02.38!1471
b/passwd!186bb83efb888e9699f6a85055aa46f1!2020-10-21-02.36!1818

```

(Aufbau: *Pfad:MD5-Summe:Zeitstempel:Größe*) und verkürzte  
Versionen :home:student:tmp.sums (Pfad + MD5) der Form

```

a/fstab!94016421866bedeeb88276d027108db9
a/readme.txt!cfba969536b62ee7757892ca94755ea8
b/passwd!186bb83efb888e9699f6a85055aa46f1

```

Schneller Vergleich zweier Ordner mit diff möglich:

```

student@linux:~> diff -y --suppress-common-lines *.sums
a/readme.txt!bcea167c1f22c622ac61e94d5a210955 | a/readme.txt!cfba969536b62ee7
b/passwd!e7b7d55f88b57148ff6563d99c5faa7b | b/passwd!186bb83efb888e9699f6

```

72

## Lösung 4: Beobachtungen

- Bisher erreicht: Vergleich von zwei lokalen Verzeichnissen  
→ dafür kein Skript nötig, rekursives diff:
 

```

student@linux:~> diff -rq tmp tmp2
Files tmp/a/readme.txt and tmp2/a/readme.txt differ
Files tmp/b/passwd and tmp2/b/passwd differ

```
- Ausgabe nicht im gewünschten Format (mit Datum und Größe)
- Kein paralleler Vergleich von mehr als zwei Ordnern
- Keine Remote-Ausführung

73

## Lösung 4: Remote-Zugriff (1)

- Idee: Skript kopiert sich auf alle Zielrechner und führt sich dort aus
- Skript-Parameter -r soll angeben, dass eine lokale Kopie (und nicht der „Master“) läuft
- alten Code-Block in get\_data()

```

...
if test -n "$user"; then
 echo Remote-Funktion nicht installiert
 return
fi
...

```

passend ersetzen

74

- Für Remote-Zugriff: ssh und scp (Secure Shell bzw. Copy)
  - ssh user@machine (Login auf machine)
  - ssh user@machine cmd (Kommando cmd auf machine ausführen)
  - Komplexere Kommandos: in "... " setzen
  - scp path1 user@machine:path2 bzw. scp user@machine:path1 path2 (Datei transferieren)
  - Bei identischem Usernamen auf beiden Rechnern: user@ weglassen
- Problem: ssh und scp fragen Passwort ab
  - automatischen Login über Public key einrichten
    - ssh-keygen -t rsa auf pc1 erzeugt Schlüssel .ssh/id\_rsa.pub,
    - diese Datei auf pc2 an .ssh/authorized\_keys anhängen

75

Neuer Code in get\_data():

```

if test -n "$user"; then
 # Remote!
 tmpfile=.tmpfile.txt # temporärer Dateiname
 scp -q $0 ${user}@${host}:/tmp/ # Skript auf Ziel-PC kopieren
 ssh ${user}@${host} chmod a+x /tmp/$0 # dort ausführbar machen
 # machines.conf auf Ziel-PC erzeugen: Trick mit SSH und Pipe
 echo machine=local:${dir} | ssh ${user}@${host} "cat >_/_/tmp/machines.conf"
 # Skript auf Ziel-PC ausführen (dort: mit Parameter -r !)
 ssh ${user}@${host} "cd /tmp; ;_/_/tmp/$0 -r"
 # rausfinden, welche Dateien auf Ziel-PC erzeugt wurden
 remote_files=$(ssh ${user}@${host} \
 "cd /tmp; ;_/_ls -tr1 *_data*.sums ;_/_tail -2")
 # Dateien vom Ziel-PC hierher kopieren und passend benennen
 for file in $remote_files; do
 scp -q ${user}@${host}:/tmp/${file} $tmpfile
 mv $tmpfile "${user}@${host}:${file}"
 done
 return
fi

```

76

Zwischenstand nach diesen Ergänzungen:

```

student@linux-us1d:~/prakt/problem4> cat machines.conf
machine=local:/tmp/dir1
machine=student@localhost:/tmp/dir2
machine=local:/tmp/dir3
machine=local:/tmp/dir4

student@linux-us1d:~/prakt/problem4> ./main.sh
student@linux-us1d:~/prakt/problem4> ls -l
insgesamt 52
-rw-r--r-- 1 student users 109 28. Okt 01:35 machines.conf
-rwxr-xr-x 1 student users 4835 28. Okt 02:08 main.sh
-rw-r--r-- 1 student users 334 28. Okt 02:08 student@localhost::tmp:dir2.data
-rw-r--r-- 1 student users 239 28. Okt 02:08 student@localhost::tmp:dir2.sums
-rw-r--r-- 1 student users 79 28. Okt 01:32 test.sh
drwxr-xr-t 3 student users 4096 28. Okt 01:33 tmp
-rw-r--r-- 1 student users 334 28. Okt 02:08 :tmp:dir1.data
-rw-r--r-- 1 student users 239 28. Okt 02:08 :tmp:dir1.sums
-rw-r--r-- 1 student users 269 28. Okt 02:08 :tmp:dir3.data
-rw-r--r-- 1 student users 193 28. Okt 02:08 :tmp:dir3.sums
-rw-r--r-- 1 student users 269 28. Okt 02:08 :tmp:dir4.data
-rw-r--r-- 1 student users 193 28. Okt 02:08 :tmp:dir4.sums
student@linux-us1d:~/prakt/p

```

77

Jetzt kommt der schwierigste Teil: die Auswertung

- Ziel: Dateien ausgeben, die
  - auf mind. einem Rechner verschieden sind (MD5-Summen!)
  - auf mind. einem Rechner fehlen
- Skript muss die \*.data-Dateien auswerten und zu jeder Datei (die es auf mind. einem Rechner gibt) Daten verwalten und vergleichen
- Gesucht: Mechanismus, der „Die Datei hab ich schon gesehen“ umsetzt
- Neuer Shell-Datentyp: Dictionary (auch: Hash, assoziatives Array), braucht bash-Version  $\geq 4$

78

- Deklaration der Dictionary-Variable dict mit **declare -A dict**
- Eintragen: `dict["key"]="value"`  
("key" und "value" sind Strings; auch Variablen nutzbar: `dict[$key]=$value`)
- Auslesen: **echo** `${dict["key"]}` oder (wenn der Key in einer Variablen steht) **echo** `${dict[$key]}`
- **Achtung:** Vergleich mit Arrays: **echo** `${array[index]}` – ohne \$ vor index (siehe Problem 3)  
→ Das geht nur bis bash-Version 3; in bash 4 auch hier: **echo** `${array[$index]}` !
- Über *keys* iterieren: **for** `i in "${!dict[@]}"; do ...`
- Über *values* iterieren: **for** `i in "${dict[@]}"; do ...`

79

Zur Erinnerung: Hauptprogramm bisher (vgl. Folie 64):

```
#!/bin/bash
CONFIG_FILE=machines.conf
MACHINE_PATTERN=machine=([^@]+@[^:]+\|local):.+
REMOTE_EXEC=$1 # bei Remote-Ausfuehrung: "-r" (neu)

if [! $BASH_VERSION \> 4]; then
 echo "Brauche_Bash_>=4.0"; exit # Dictionaries! (neu)
fi

i=0
while read; do
 if [[$REPLY =~ $MACHINE_PATTERN]]; then
 machine[$i]="$REPLY"; let i++
 fi
done < $CONFIG_FILE
machines=$i

Daten der verschiedenen Ordner sammeln
for ((i=0; i<$machines; i++)); do get_data ${machine[i]}; done
...
```

80

## Lösung 4: Datenabgleich (3)

Jetzt weiter:

```
...
if ["$REMOTE_EXEC" = "-r"]; then
 exit # Auswertung nur auf lokaler Maschine
fi

Header für Ausgabe erzeugen
for ((i=0; i<$machines; i++)); do
 echo "pc$i_=${machine[i]}"
done
echo
...
```

und dann

- .data-Dateien verarbeiten
- jeden Eintrag in Dateiname, Hash, Datum, Größe zerlegen
- Dateiname als *key* verwenden und damit Einträge in Dictionaries `hashes`, `dates`, `sizes` erzeugen

81

## Lösung 4: Datenabgleich (4)

Ansatz:

```
declare -A hashes dates sizes count # Dictionaries (bash 4)
for ((i=0; i<$machines; i++)); do
 # Eine Datei auslesen
 datafile=$(echo ${machine[i]} |
 sed -e s/^machine=// -e s/^local:// -e s/./data/ | \
 tr "/" ":")

 while read; do # Eine .data-Datei auswerten
 line=$REPLY
 filename=${line%%\!*}; tmp_rest=${line#\!*}
 hash=${tmp_rest%%\!*}; tmp_rest=${tmp_rest#\!*}
 date=${tmp_rest%%\!*}; size=${tmp_rest#\!*}
 ##### TO DO: Dictionaries aktualisieren
 done < $datafile
done
```

Was machen `${line%%\!*}` und `${line#\!*}`?

82

```
$ line='Daten/test1.txt!d41d8cd98f00b204e9800998ecf8427e!201
6-10-28-01.32!4192'
$ filename=${line%%\!*}; tmp_rest=${line#\!*}
$ echo $filename; echo $tmp_rest
Daten/test1.txt
d41d8cd98f00b204e9800998ecf8427e!2020-10-28-01.32!4192
$ hash=${tmp_rest%%\!*}; tmp_rest=${tmp_rest#\!*}
$ echo $hash; echo $tmp_rest
d41d8cd98f00b204e9800998ecf8427e
2020-10-28-01.32!4192
$ date=${tmp_rest%%\!*}; size=${tmp_rest#\!*}
$ echo $date; echo $size
2020-10-28-01.32
4192
```

- `${var%%muster}` : muster hinten abschneiden (*greedy*)
- `${var#muster}` : muster vorne abschneiden (*non-greedy*)
- Es gibt auch `${var%muster}` und `${var##muster}`

83

Das Aktualisieren der Dictionaries sieht zunächst einfach aus

- Nach dem Deklarieren gibt es keine Einträge,
- `${dict[key]}` liefert für jeden `key` "" zurück
- also einfach immer neuen Wert anhängen:
 

```
hashes[$filename]=${hashes[$filename]}"\n"$hash
dates[$filename]=${dates[$filename]}"\n"$date
sizes[$filename]=${sizes[$filename]}"\n"$size
((count[$filename]++))
```

 (mit Zeilenumbruch "\n" als Trenner)
- Diese Lösung funktioniert, wenn alle Dateien in allen Ordnern existieren
- Wenn nicht, gibt es für solche Dateien weniger Einträge als Maschinen

84

## Lösung 4: Datenabgleich (6)

Einheitliche Struktur erreichen wir durch manuell eingefügte Leerzeilen:

```
Dictionaries aktualisieren
if ["${hashes[$filename]}" = ""]; then
 # Datei taucht zum 1. Mal auf
 extra=""; num=1
 for ((j=0; j<$i; j++)); do extra=$extra"\n"; ((num++)); done
 hashes[$filename]=$extra$hash
 dates[$filename]=$extra$date
 sizes[$filename]=$extra$size
 count[$filename]=$num
else
 # Datei war schon da
 hashes[$filename]="${hashes[$filename]}"\n"$hash # append
 dates[$filename]="${dates[$filename]}"\n"$date # append
 sizes[$filename]="${sizes[$filename]}"\n"$size # append
 ((count[$filename]++))
fi
```

85

## Lösung 4: Datenabgleich (7)

Leerzeilen ggf. an vorhandene Einträge anhängen:

```
for ((i=0; i<$machines; i++)); do # noch mal die ganze Schleife
 datafile=... # im Überblick
 while read; do # ...
 # filename, hash, date, size setzen # ...
 # Dictionaries aktualisieren # ...
 done < $datafile # ...

ggf. Leerzeilen anhängen
if [$i -gt 0]; then
 for filename in "${!hashes[@]}"; do
 if [${count[$filename]} -lt ${i+1}]; then
 hashes[$filename]="${hashes[$filename]}"\n"
 dates[$filename]="${dates[$filename]}"\n"
 sizes[$filename]="${sizes[$filename]}"\n"
 ((count[$filename]++))
 fi
 done
fi
done # Ende äußere Schleife
```

86

Für eine Beispieldatei test3.txt, die auf drei Rechnern liegt, kann sich jetzt ergeben:

```
hashes[test3.txt] = d41d8cd98f00b204e9800998ecf8427e
 e3577c206a2f8eb4ef57f8b7ece4c44d
 e3577c206a2f8eb4ef57f8b7ece4c44d

sizes[test3.txt] = 4673
 3211
 3211

dates[test3.txt] = 2020-10-28-01.32
 2020-10-28-01.32
 2020-10-28-01.32
```

Da die Hashes nicht alle gleich sind, soll eine Ausgabe erfolgen; das geht z. B. (relativ) bequem mit pr:

```
{ echo -e ${sizes[$filename]}
 echo -e ${dates[$filename]}
 echo -e ${hashes[$filename]}
} | pr -3 -t -s "␣" | xargs printf "%s␣%s␣%s\n"
```

erzeugt Ausgabe der Form

```
4673 2020-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
3211 2020-10-28-01.32 5e6d4955ac1c39e3f9b5bb115e00c4ca
3211 2020-10-28-01.32 5e6d4955ac1c39e3f9b5bb115e00c4ca
```

(und das ist schon nah am Ziel)

- pr -3 verteilt Eingabe auf drei Ausgabespalten
- Optionen -t (keine Header/Footer), -s "␣" (Leerzeichen als Spaltentrenner)
- xargs macht aus Standardeingabe Argumente für den folgenden Befehl

87

88

### Einschub: pr und xargs

```
$ seq 1 12 | pr -3 -t $ seq 1 12 | pr -3 -t -s" "
1 5 9 1 5 9
2 6 10 2 6 10
3 7 11 3 7 11
4 8 12 4 8 12

$ echo 1 2 3 | xargs printf "%s %s %s\n"
1 2 3
$ # gleiche Wirkung wie
$ printf "%s %s %s\n" 1 2 3
1 2 3

$ find -type f $ find -type f | xargs -i{} echo cp {} {}.bak
./test.sh cp ./test.sh ./test.sh.bak
./main.sh cp ./main.sh ./main.sh.bak
./test2.sh cp ./test2.sh ./test2.sh.bak
./machines.conf cp ./machines.conf ./machines.conf.bak
```

89

### Lösung 4: Datenabgleich (10)

Letzte Arbeiten: nur Dateien mit Änderungen ausgeben, Ausgabe schöner formatieren

```
Abweichungen suchen
treffer=0
for filename in "${!hashes[@]}; do
 # Anzahl der verschiedenen Hash-Einträge (bei Gleichheit: 1)
 num=$(echo -e ${hashes[$filename]} | sed 's/^\$/MISSING/' | \
 sort -u | wc -l)
 if [$num \> 1]; then
 ((treffer++))
 printf "[%2d]␣%s:\n" $treffer $filename
 {
 for ((i=0; i<$machines; i++)); do echo $treffer; done
 for ((i=0; i<$machines; i++)); do echo "pc$i:"; done
 echo -e ${sizes[$filename]} | sed 's/^\$/<missing>/'
 echo -e ${dates[$filename]} | sed 's/^\$/-/-'
 echo -e ${hashes[$filename]} | sed 's/^\$/-/-'
 } | pr -5 -t -s"␣" | xargs printf "[%2d]␣%s␣%10s␣%-17s␣%s\n"
 fi
done
```

90



```

student@linux-us1d:~/prakt/problem4> ./main.sh
pc0 = machine=local:/tmp/dir1
pc1 = machine=student@localhost:/tmp/dir2
pc2 = machine=local:/tmp/dir3
pc3 = machine=local:/tmp/dir4

[1] Daten/test3.txt:
[1] pc0: 0 2016-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
[1] pc1: <missing> - -
[1] pc2: <missing> - -
[1] pc3: <missing> - -
[2] Daten/test2.txt:
[2] pc0: 0 2016-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
[2] pc1: 7 2016-10-28-01.32 5e6d4955ac1c39e3f9b5bb115e00c4ca
[2] pc2: 7 2016-10-28-01.32 5e6d4955ac1c39e3f9b5bb115e00c4ca
[2] pc3: 7 2016-10-28-01.32 5e6d4955ac1c39e3f9b5bb115e00c4ca
[3] Daten/test4.txt:
[3] pc0: <missing> - -
[3] pc1: 0 2016-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
[3] pc2: 6 2016-10-28-01.32 e3577c206a2f8eb4ef57f8b7ece4c44d
[3] pc3: 6 2016-10-28-01.32 e3577c206a2f8eb4ef57f8b7ece4c44d
[4] Bilder/2.jpg:
[4] pc0: 0 2016-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
[4] pc1: 0 2016-10-28-01.32 d41d8cd98f00b204e9800998ecf8427e
[4] pc2: <missing> - -
[4] pc3: <missing> - -
student@linux-us1d:~/prakt/problem4>

```

91

```

1 #!/bin/bash
2 CONFIG_FILE=machines.conf
3 MACHINE_PATTERN=machine=([^\@]+@[^\:]+\|local):.+
4 # example 1: machine=username1@pcname1:basedir1
5 # example 2: machine=local:basedir1
6
7 function get_data {
8 old_IFS="$IFS"; IFS=":"
9 index=0
10 for arg in $1; do
11 case $index in
12 0) ;; # "machine"
13 1) user=$arg;; # "username1"
14 2) host=$arg;; # "pcname1"
15 3) dir=$arg;; # "basedir1"
16 esac
17 let index++
18 done
19 IFS="$old_IFS"
20 if [$index -eq 3]; then # local
21 dir=$host; unset user host
22 fi
23
24 if test -n "$user"; then
25 # Remote!
26 tmpfile=.tmpfile.txt
27 scp -q $0 ${user}@${host}:/tmp/
28 ssh ${user}@${host} chmod a+x /tmp/$0
29 echo machine=local:${dir} | ssh ${user}@${host} "cat > /tmp/machines.conf"
30 ssh ${user}@${host} "cd /tmp/; /tmp/$0 -r" # Parameter -r !
31 remote_files=(ssh ${user}@${host} "cd /tmp/; ls -tr1 *.data *.sums | tail -2")

```

92

## Lösung 4: Komplette Lösung (2)

```

32 for file in $remote_files; do
33 scp -q ${user}@${host}:/tmp/${file} tmpfile
34 mv tmpfile "${user}@${host}:${file}"
35 done
36 return
37 fi
38
39 filename_prefix="${dir//\://}"
40 current_dir="$PWD"
41 cd "$dir"
42 find . -type f | cut -c3- > ${current_dir}/${filename_prefix}.files
43 cd "$current_dir"; rm -f ${filename_prefix}.tmpdata
44 while read file; do
45 sum=$(md5sum "$dir/$file" | cut -f1 -d" ")
46 date=$(stat -c '%y' "$dir/$file" | cut -c1-16 | tr ' ' '-')
47 size=$(stat -c '%s' "$dir/$file")
48 echo "$file!$sum!$date!$size" >> ${filename_prefix}.tmpdata
49 done < ${filename_prefix}.files
50
51 sort ${filename_prefix}.tmpdata > ${filename_prefix}.data
52 cut -f1,2 -d! < ${filename_prefix}.data > ${filename_prefix}.sums
53 rm ${filename_prefix}.*{files,tmpdata}
54 }
55
56
57 # Hauptprogramm
58 if [! $BASH_VERSION \> 4]; then
59 echo "Brauche Bash >= 4.0"
60 exit
61 fi
62

```

93

## Lösung 4: Komplette Lösung (3)

```

63 REMOTE_EXEC=$1 # Hier steht bei Remote-Ausführung "-r" drin
64
65 i=0
66 while read; do
67 if [[$REPLY =~ $MACHINE_PATTERN]]; then
68 machine[i]=$REPLY;
69 let i++
70 fi
71 done < $CONFIG_FILE
72 machines=$i
73
74 # Daten der verschiedenen Ordner sammeln
75 for ((i=0; i<$machines; i++)); do
76 get_data ${machine[i]}
77 done
78
79 if ["$REMOTE_EXEC" = "-r"]; then
80 exit # Auswertung nur auf lokaler Maschine
81 fi
82
83 # Daten auswerten
84 for ((i=0; i<$machines; i++)); do
85 echo "pc$i=$_{machine[i]}"
86 done
87 echo
88
89 declare -A hashes dates sizes count # Hash-Datenstrukturen
90 for ((i=0; i<$machines; i++)); do
91 # Eine Datei auslesen
92 datafile=$(echo ${machine[i]} |
93 sed -e s/^machine=// -e s/^local:// -e s/!.data/ | \

```

94

```

94 tr "/" ":")
95
96 while read; do
97 line=$REPLY
98 filename=${line%%!*}; tmp_rest=${line#!}
99 hash=${tmp_rest%%!*}; tmp_rest=${tmp_rest#!}
100 date=${tmp_rest%%!*}; size=${tmp_rest#!}
101
102 if ["${hashes[$filename]}" = ""]; then
103 # Datei taucht zum 1. Mal auf
104 extra=""; num=1
105 for ((j=0; j<$i; j++)); do extra=$extra"\n"; ((num++)); done
106 hashes[$filename]=$extra$hash
107 dates[$filename]=$extra$date
108 sizes[$filename]=$extra$size
109 count[$filename]=$num
110 else
111 # Datei war schon da
112 hashes[$filename]="${hashes[$filename]}"$hash # append
113 dates[$filename]="${dates[$filename]}"$date # append
114 sizes[$filename]="${sizes[$filename]}"$size # append
115 ((count[$filename]++))
116 fi
117 done < $datafile
118
119 # ggf. Leerzeilen anhängen
120 if [$i -gt 0]; then
121 for filename in "${!hashes[@]}; do
122 if [${count[$filename]} -lt $[i+1]]; then
123 hashes[$filename]="${hashes[$filename]}"$"\n"
124 dates[$filename]="${dates[$filename]}"$"\n"

```

95

```

125 sizes[$filename]="${sizes[$filename]}"$"\n"
126 ((count[$filename]++))
127 fi
128 done
129 fi
130 done
131
132 # Abweichungen suchen
133 treffer=0
134 for filename in "${!hashes[@]}; do
135 # Anzahl der verschiedenen Hash-Einträge (bei Gleichheit: 1)
136 num=$(echo -e "${hashes[$filename]} | sed 's/^$/MISSING/' | sort -u | wc -l)
137 if [$num \> 1]; then
138 ((treffer++))
139 printf "[%2d]_s:\n" $treffer $filename
140 {
141 for ((i=0; i<$machines; i++)); do echo $treffer; done
142 for ((i=0; i<$machines; i++)); do echo "pci:"; done
143 echo -e "${sizes[$filename]} | sed 's/^$/<missing>/'
144 echo -e "${dates[$filename]} | sed 's/^$/-/'"
145 echo -e "${hashes[$filename]} | sed 's/^$/-/'"
146 } | pr -5 -t -s"_" | xargs printf "[%2d]_s%10s_u%-17s_u%\n"
147 fi
148 done
149
150 rm *.data *.sums # Aufräumen

```

96

## Problem/Lösung 4: Neues Wissen

- Pattern-Vergleich mit `[ [ $v1 =~ $v2 ] ]` (war schon mal da)
- **let** `i++` statt `((i=i+1))` oder `i=${i+1}`
- C-Style-**for**-Schleife: `for ((i=0; i<5; i++)); do ...`
- String zerlegen, `$IFS` anpassen
- `hexdump`
- Fallunterscheidung mit `case $x in ... ;; ... esac`
- Einfaches Suchen/Ersetzen in Variablen mit `${var//alt/neu}`
- `find path -type f`
- `md5sum`, `stat -c`, `diff`, `pr`, `xargs`, `ssh`, `scp`
- Dictionaries, **declare** `-A`, `key + value` (nur in bash 4.x)
- Variablen abschneiden mit `${var%muster}`, `${var%%muster}`, `${var#muster}`, `${var##muster}`

97

## Problem 5: Das CSV-Problem

## Problem 5: CSV-Dateien verarbeiten (1)

- CSV-Dateien (Comma-Separated Values) sind ein beliebtes Exportformat für Daten mit Tabellenstruktur
- klassisch: Komma als Feldtrenner (daher der Name)
- alternativ: beliebiger Trenner
- Beispiel:

```
1 # Name,Vorname,Matr.Nr.,Note
2 Maier,Maria,123456,1.0
3 Müller,Markus,234567,2.0
4 Ohflein,Thomas,323232,5.0
5 Ritter,Roman,432432,1.3
```

98

## Problem 5: CSV-Dateien verarbeiten (2)

- Mit `grep`, `sed`, `column -s`, `sort -t`, `cut -d` etc. können wir CSV-Dateien schon ganz gut verarbeiten – dank „field separators“ / „delimiters“
- Beispiel:  

```
$ grep -v -e ^# -e ^$ noten.dat | \
 cut -d, -f1,4 | sort -t, -k2 -n | \
 sed 's/\([^,]*\),\([^,]*\)/\2,\1/'
```

1.0,Maier  
1.3,Ritter  
2.0,Müller  
5.0,Ohflein
- Problem: Trenner kann in Feld vorkommen
- CSV-Lösung: Feld in Anführungszeichen setzen

99

## Problem 5: CSV-Dateien verarbeiten (3)

- CSV-Dateien mit Trenner im Feldinhalt

```
1 # Kurs,Name,Vorname,Matr.Nr.,Note
2 BS 1,Maier,Maria,123456,1.3
3 "BS_2, Vorlesung",Maier,Maria,123456,1.0
4 "BS_2, Praktikum",Maier,Maria,123456,2.3
5 BS 1,Müller,Markus,234567,2.0
```

- Einfache Trenner-Angabe funktioniert nicht mehr:

```
$ tail -n +2 noten2.dat | cut -d, -f2,4
Maier,123456
 Vorlesung",Maria
 Praktikum",Maria
Müller,234567
```

100

## Problem 5: CSV-Dateien verarbeiten (4)

Aufgabenstellung: Mit CSV-Dateien wie gewohnt arbeiten:

- Zeilen auswählen (wie `grep`),
- Spalten auswählen (wie `cut`),
- sortieren,
- Text ersetzen etc.

Aber mit `cut`, `grep` etc. kommen wir da nicht weiter.

101

- Was tun?

102

- Was tun?
- Trenner antworten auf: „Was ist zwischen den Feldern?“
- Alternative: „Wie sieht ein Feld aus?“
  - (1) etwas ohne Kommata: `[^, ]+`
  - (2) etwas in Anführungszeichen: `"[^"]+"`
  - zusammen: `( [^, ]+ ) | ( "[^"]+" )`

102

- Was tun?
- Trenner antworten auf: „Was ist zwischen den Feldern?“
- Alternative: „Wie sieht ein Feld aus?“
  - (1) etwas ohne Kommata: `[^, ]+`
  - (2) etwas in Anführungszeichen: `"[^"]+"`
  - zusammen: `( [^, ]+ ) | ( "[^"]+" )`

- Hier hilft **awk**:

```
$ tail -n +2 noten2.dat | \
 awk '{ print $1 }' FPAT='([^\,]+)|("[^"]+")'
```

```
BS 1
```

```
"BS 2, Vorlesung"
```

```
"BS 2, Praktikum"
```

```
BS 1
```

102

- **awk**, benannt nach
  - Alfred V. **Aho** (→ Koautor von „Compilers: Principles, Techniques, and Tools“),
  - Peter J. **Weinberger**,
  - Brian W. **Kernighan** (→ Mitentwickler von C; „K&R C“, Koautor von „The C Programming Language“)
- Skriptsprache, „*data-driven*“ (vs. prozedural)
- sehr mächtiges Tool, heute oft als GNU awk (gawk) im Einsatz

103

Was macht der kurze **awk**-Befehl?

```
awk '{ print $1 }' FPAT='([^\,]+)|("[^"]+")'
```

- liest von stdin, schreibt nach stdout
- zerlegt jede Eingabezeile in Felder und gibt Feld 1 (\$1) aus
- zur Zerlegung in Felder dient die Felddefinition über `( [^\,]+ ) | ( " [^"]+ " )` (via FPAT, „Field Pattern“)

Mehr zu **awk**: später, alternativ diverse Bücher, z. B.

„The GNU Awk User's Guide“,

<https://www.gnu.org/software/gawk/manual/gawk.html>

104

105

## Lösung 5: CSV-Dateien verarbeiten (4)

Zur Aufgabenstellung (nach Noten sortierte Liste mit Note, Vorname+Nachname, Veranstaltung):

```
$ awk '$5 ~ "^[0-9]+" { print $5 ":" $3 " " $2 ":" \
 $1 }' FPAT='([^\,]+)|("[^"]+")' noten2.dat | sort -n
```

```
1.0: Maria Maier: "BS 2, Vorlesung"
```

```
1.3: Maria Maier: BS 1
```

```
2.0: Markus Müller: BS 1
```

```
2.3: Maria Maier: "BS 2, Praktikum"
```

106

## Lösung 5: CSV-Dateien verarbeiten (5)

**awk**-Befehl zu unübersichtlich? → separates Skript erstellen:

```
$ cat extract-csv.awk
BEGIN {
 FPAT = "([^\,]+)|(\"[^\"]+\")"
}
$5 ~ "^[0-9]+" {
 print $5 ":" $3 " " $2 ":" $1
}
```

```
$ awk -f extract-csv.awk noten2.dat | sort -n
```

```
1.0: Maria Maier: "BS 2, Vorlesung"
```

```
1.3: Maria Maier: BS 1
```

```
2.0: Markus Müller: BS 1
```

```
2.3: Maria Maier: "BS 2, Praktikum"
```

107

Anwendung in Kombination mit column:

```
$ awk '$0 != "" { gsub(/^"/, "", $1); gsub(/"$/, "", $1); print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5 }' \
 FPAT='([^\,]+)|("[^"]+")' noten2.dat | \
 column -t -s $'\t'
```

| # | Kurs            | Name   | Vorname | Matr.Nr. | Note |
|---|-----------------|--------|---------|----------|------|
|   | BS 1            | Maier  | Maria   | 123456   | 1.3  |
|   | BS 2, Vorlesung | Maier  | Maria   | 123456   | 1.0  |
|   | BS 2, Praktikum | Maier  | Maria   | 123456   | 2.3  |
|   | BS 1            | Müller | Markus  | 234567   | 2.0  |

108

```
BEGIN {
 ... # Aktionen vor Verarbeitung der Eingabedaten
}

/muster/ { # Muster, mit dem Zeile verglichen wird
 ... # Aktionen, wenn dieses Muster passt
}

test { # Test / Bedingung, die erfüllt sein muss
 ... # Aktionen, falls ja
}

END {
 ... # Aktionen nach Verarbeitung der Eingabedaten
}
```

109

### Einschub: Beispiele für Muster/Tests

- dritte Zeile ausgeben: `awk 'NR==3 {print}' file`
- gerade Zeilen: `awk 'NR % 2 == 0 {print}' file`
- Eintrag in /etc/passwd mit User-ID 249:  
`awk '$3 == "249" {print}' FS=":" /etc/passwd`  
(FS = Field Separator)
- Einträge mit User-ID = Group-ID, nur Username und ID:  
`awk '$3 == $4 {print $1":"$3}' FS=":" /etc/passwd`
- Nach je vier Zeilen ===== ausgeben:  
`awk '{print} NR % 4 == 0 {print "=====}"' file`
- Warnung bei Leerzeilen:  
`awk '/^$/ {print "Warnung:␣Zeile␣" NR "␣leer}"' file`

110

### Problem/Lösung 5: Neues Wissen

- Grenzen der Feldverarbeitung über Trennzeichen
- Alternative: Definition Feldformat
- Erste Schritte in `awk`

111

## Problem 6: Verzeichnis überwachen

---

### Aufgabenstellung

- Skript soll einen Ordner überwachen – auf neue / geänderte / gelöschte Dateien
- Abhängig von Aktion und Dateityp: verschiedene Automatismen
  - Datei drucken
  - Datei komprimieren und archivieren
  - Information in Log-Datei schreiben
  - etc.
- Skript soll möglichst wenig Ressourcen (Rechenzeit) verbrauchen

112

## Problem 6: Verzeichnis überwachen (2/2)

## Lösung 6: Erster Ansatz

### Beispiel-Regeln

- Neue Datei \*.pdf → Datei drucken mit `lpr`
- Neue Datei \*.iso → ISO-Image auf DVD brennen mit `cdrecord` und anschließend nach `/home/user/ISOs/gebrannt/` verschieben
- Neue Datei \*.ps → PostScript-Datei mit `ps2pdf` ins PDF-Format konvertieren und Original löschen
- Unterverzeichnis in diesen Ordner verschoben → Zip-Archiv zu Ordner erstellen
- Datei gelöscht → Löschen in Protokolldatei vermerken

Regeln können noch für verschiedene Unterordner angepasst werden.

### Manuelle Überwachung

- Regelmäßig (z. B. alle fünf Sekunden)
 

```
cd $TMPDIR
mv files files.old; find $WATCHDIR -type f > files
diff files files.old > /dev/null || AKTION
```
- AKTION muss dann Unterschied (Ausgabe von `diff`) auswerten und angemessen reagieren

### Probleme

- Aufwändige, regelmäßige `find`-Aufrufe
- Dateien, die sich nur geändert haben, fallen hier nicht auf (→ könnte man mit `stat` für jede einzelne Datei lösen)

113

114

## Automatische Überwachung

- Das Programm `inotifywait` nutzt eine Funktion des Linux-Kernels zur Datei-/Verzeichnis-Überwachung
- Syntax, nützliche Optionen
  - Allgemein: `inotifywait [options] pfad`
  - `-m` → monitor: dauerhaft beobachten (sonst Abbruch nach dem ersten Event)
  - `-q` → quiet: keine Startnachricht
  - `-e ...` → event: Einschränkung der zu überwachenden Events
  - `-o dateiname` → Ausgabe umleiten
  - `--format ...` → Ausgabeformat ändern (`%w =`, `%f =`, `%e =`)
- Beispiel: Datei erzeugen, umbenennen, löschen, Listing (`ls`)

```
linux-us1d:~ # inotifywait -mq /tmp/
/tmp/ CREATE test.txt
/tmp/ OPEN test.txt
/tmp/ ATTRIB test.txt
/tmp/ CLOSE_WRITE,CLOSE test.txt
/tmp/ MOVED_FROM test.txt
/tmp/ MOVED_TO neuername.txt
/tmp/ OPEN,ISDIR
/tmp/ CLOSE_NOWRITE,CLOSE,ISDIR
/tmp/ DELETE neuername.txt
/tmp/ OPEN,ISDIR
/tmp/ CLOSE_NOWRITE,CLOSE,ISDIR
```

115

116

## Lösung 6: Vorbereitungen

- Ordner anlegen
 

```
linux-us1d:~ # mkdir /tmp/auto
```
- Skript mit FIFO-Datei
 

```
1 rm -f /tmp/watcher.fifo
2 mkfifo /tmp/watcher.fifo
3 inotifywait -mq -o /tmp/watcher.fifo --format \
4 "%w!%f!%e" -e close_write . 2> /dev/null &
5 while read -r msg; do
6 folder=${msg%!*}; rest=${msg#!*}
7 file=${rest%!*}; action=${rest#!*}
8 echo folder=\"$folder\", file=\"$file\", \
9 action=\"$action\"
10 done < /tmp/watcher.fifo
```

## Einschub: Was ist eine FIFO-Datei?

- FIFO (first in, first out): spezieller Dateityp
- erzeugen mit `mkfifo`

```
linux-us1d:/tmp # mkfifo test.fifo
linux-us1d:/tmp # ls -l test.fifo
prw-r--r-- 1 root root 0 5. Jan 22:31 test.fifo
```

→ p steht für *named pipe*
- Mehrere Prozesse können in Pipe schreiben und daraus lesen
- ähnlich zu dem, was in Shell-Pipes passiert: `prog1 | prog2` ist (grob) äquivalent zu
 

```
mkfifo pipe
prog1 > pipe &
prog2 < pipe
```

117

118



- Skript erzeugt FIFO `watcher.fifo`
- `inotifywait` als Hintergrundjob
- Ausgabe in die Pipe, mit Format  
`/tmp/|CLOSE_WRITE,CLOSE|test.txt`  
 statt  
`/tmp/ CLOSE_WRITE,CLOSE test.txt`  
 (Annahme: kein „|“-Zeichen in Dateinamen ...)
- Nur Events vom Typ `close_write` verarbeiten (für volle Liste: man `inotifywait`, Abschnitt EVENTS)
- Lesen aus der Pipe mit `while read; do ... done < pipe` (vgl. frühere Lösungen)
- Bash-String-Operationen `${var%...}` und `${var#...}` für vorne/hinten abschneiden

119

Skript erzeugt Ausgaben der Form

```
folder=".", file="test-datei", action="CLOSE_WRITE,CLOSE"
folder=".", file="datei2.txt", action="CLOSE_WRITE,CLOSE"
folder=".", file="extra.pdf", action="CLOSE_WRITE,CLOSE"
```

Damit können wir arbeiten ...

Achtung: gemeldete Aktion ist immer `CLOSE_WRITE,CLOSE`. Es gibt auch: `CLOSE_NOWRITE,CLOSE`. Jedes Datei-Schließen erzeugt ein `CLOSE` (auch Schließen ohne Ändern).

Auch neu erzeugte Unterordner tauchen hier auf.

Nächste Folie: angepasstes Skript mit `-e close` (findet alle `close()`-Operationen auf Dateien und Verzeichnisse

120

## Lösung 6: Zweiter Ansatz

Befehle: `ls, mkdir z, touch d, cat d`

```
folder=".", file="", action="CLOSE_NOWRITE,CLOSE,ISDIR"
folder=".", file="z", action="CLOSE_NOWRITE,CLOSE,ISDIR"
folder=".", file="d", action="CLOSE_WRITE,CLOSE"
folder=".", file="d", action="CLOSE_NOWRITE,CLOSE"
```

(Erster Befehl bezieht sich nur auf das Verzeichnis selbst, nicht auf einen der Einträge, darum dort: `file=""`)

121

## Lösung 6: Weitere interessante Events

- Eine Append-Operation (in der Shell mit `>>`) verursacht (u.a.) einen `MODIFY`-Event: `echo Test >> d` →

```
folder=".", file="d", action="OPEN"
folder=".", file="d", action="MODIFY"
folder=".", file="d", action="CLOSE_WRITE,CLOSE"
```

- Umbenennen (in der Shell mit `mv`) verursacht zwei `MOVED_FROM`- und `MOVED_TO`-Events: `mv d e` →

```
folder=".", file="d", action="MOVED_FROM"
folder=".", file="e", action="MOVED_TO"
```

(Taucht nur eine der beiden Zeilen auf, war es eine Verschiebe-Aktion in diesen Ordner oder aus diesem Ordner heraus.)

122

Code hinter Zerlegung in folder, file, action einbauen:

- GUI-Hinweis auf neu erzeugte PDF-Datei:
 

```
[[$file =~ \.pdf$]] && [[$action = CREATE]] && \
 kdialog --msgbox \
 "Neue PDF-Datei $file im Ordner $folder"
```
- PostScript-Datei konvertieren und Original löschen:
 

```
[[$file =~ \.ps$]] && [[$action = MOVED_T0]] && \
 ps2pdf $file && rm $file
```
- ISO-Image brennen und verschieben:
 

```
[[$file =~ \.iso$]] && [[$action = MOVED_T0]] && \
 wodim dev=/dev/sr0 -eject $file && \
 mv $file ~/ISOs/gebrannt/
```

(Hinweis: wodim ist der Nachfolger von cdrecord.)

123

- Zip-Archiv zu Ordner erstellen:
 

```
[[$action = MOVED_T0,ISDIR]] && \
 (cd $folder; zip -r ${file}.zip $file)
```
- Löschen in Protokolldatei vermerken
 

```
[[$action = DELETE]] && \
 logger -t watcher "Datei $folder/$file geloescht"
```

Werden Objekte nicht in den überwachten Ordner verschoben (MOVED\_T0), sondern erzeugt (CREATE), sind Automatismen schwieriger umzusetzen:

ggf. Erstellung noch nicht abgeschlossen, wenn inotifywait reagiert

124

- inotifywait könnte auch in Schleife ohne -m (monitor) aufgerufen werden
- Warum nicht einfach Schleife der Form
 

```
while true; do
 msg=$(inotifywait -q --format '%w!%f!%e' .)
 ...
done
```

schreiben und auf die FIFO-Datei verzichten?

125

- inotifywait überwacht Ordner
- FIFOs (named pipes), mkfifo, „p“
- in Automatisierungen erwähnt: kdialog, ps2pdf, wodim (bzw. cdrecord), zip, logger

126