Unix & Internet

Uwe Schmidt
FH Wedel

WS 1996/97

Inhaltsverzeichnis

Literatur

[Bach 91] Bach, M.:
UNIX – Wie funktioniert das Betriebssystem?
Hanser, München, 1991
ISBN 3-446-201740-7

[Bach 87] Bach, F., Doman, P., Weng-Beckmann, U.:
UNIX, Handbuch zur Programmentwicklung
Hanser, Wien, 1987
ISBN 3-446-15103-6

[Bolsky 89] Bolsky M.I., Korn D.G.:
The KornShell, Command and Programming Language
Prentice Hall, Englewood Cliffs, New Jersey, 1989
ISBN 0-13-516972-0

[Bourne 83] Bourne, S.R.:
The UNIX System
Addison–Wesley, Wokingham, England, 1983
ISBN 0-201-13791-7

[Coffin 89] Coffin, S.:
UNIX-Befehlsbibliothek
McGraw-Hill, Hamburg, 1989
ISBN 3-89028-152-4

[Dannegger 91] Dannegger, C., Geugelin-Danneger, P.:
Parallele Prozesse unter Unix: Simulation und Anwendung bekannter Synchronisationsmethoden in C unter UNIX
Hanser, München, 1991
ISBN 3-446-15911-8

[Foxley 87] Foxley, E.:
UNIX für Superuser
Addison–Wesley, Bonn, 1987
ISBN 3-92511824-1

[Gulbins 88] Gulbins, J.:
UNIX, Version 7, bis System V.3
3.Auflage, Springer–Verlag, Berlin, 1988
ISBN 3-540-19248-4

[Kernighan 86] Kernighan, B.W., Pike, R.:
Der UNIX–Werkzeugkasten, Programmieren mit UNIX
Hanser, Wien, 1986
ISBN 3-446-14273-8

[Maurer 96] Maurer, R.:
HTML und CGI Programmierung
Mit einer Einführung in Tcl
Buch und CD
dpunkt, Heidelberg, 1996, 68,–DM
ISBN 3-920993-28-4

[O'Reilly 89] O'Reilly, T.:
Das UNIX V Nutshell-Buch
Addison–Wesley, Bonn, 1989
ISBN 3-89319-234-4

[Schreiner 87] Schreiner, A.T.:
Professor Schreiners UNIX Sprechstunde, Technik, Tips und Tricks von awk bis yacc
Hanser, Wien, 1987
ISBN 3-446-14894-9

[Stevens 90] Stevens, R.:
Unix Network Programming

Prentice Hall Software Series
ISBN 0-13-949876-1

[Hetze 94] Hetze, S.,Müller, M. u.a.:
LinuX Anwenderhandbuch und Leitfaden für die Systemverwaltung
4. erweiterte Auflage
lunetIX Softfair, 1994
ISBN 3-929764-00-8

[Beck 94] Beck, M. u.a.:
Linux–Kernel–Programmierung: Algorithmen und Strukturen der Version 1.0
Addison–Wesley, Bonn, 1994
ISBN 3-89319-712-5

[WI-Seminar 94] *WI–Seminar SS94: Linux*
FH Wedel, 1994

[SuSE 96] S.u.S.E GmbH:
S.u.S.E. Linux, Unix für PCs, Quellcode & Anwendungen
3 CD-ROMs, Kernel 2.0.18, September 1996

1 Einfache Kommandos und Filesystem

einfache Kommandos

```
$ pwd
//node_25e48/sys5.3/usr/uwe/unix

$ ls
hello

$ who
uwe                  display     Sep 11 13:16

$ date
Tue Sep 11 13:37:34     1990
```

Kommandos mit Parametern

```
$ echo hello world
hello world

$ Echo hello world
Echo: not found

$ ls -l
total 1
-rw-rw-rw-   1 uwe            12 Sep 11 13:25 hello
```

root directory

```
$ ls /
bin      nohup.out.bak
bscom    pas_config
bsd4.2   pas_install
.......
```

online–Manual

```
$ man echo

ECHO(1)                   BSD                      ECHO(1)

NAME
     echo - echo arguments

SYNOPSIS
     echo [ -n ] [ arg ... ]

DESCRIPTION
     echo writes its arguments separated by blanks, and
     terminated by a newline, on the standard output.
     If you specify the -n option, echo doesn´t add a
     newline to the output.

     echo is useful for producing diagnostics in shell
     programs and for writing constant data on pipes.

     To send diagnostics to the standard error file in
     a /bin/sh shell script, specify the following:

          echo ... 1>&2

SEE ALSO
     csh(1), sh(1)
```

online MANual für das online MANual

```
$ man man

MAN(1) ......
```


conCATenate

```
$ cat hello
hello world

$ cat //node_25e48/sys5.3/usr/uwe/unix/hello
hello world

$ cat bye
good bye

$ cat hello bye
hello world
good bye
```

MoVe, CoPy, ReMove

```
$ mv hello hallo
$ ls
bye    hallo

$ cp hallo hello
$ ls
bye    hallo  hello

$ rm hello
$ ls
bye    hallo
```

LiSt files

```
$ ls
bye    hallo

$ ls -l
total 2
-rw-rw-rw-   1 uwe              9 Sep 11 13:45 bye
-rw-rw-rw-   1 uwe             12 Sep 11 13:25 hallo
```

Word Count

```
$ wc tmp
       2       2      12 tmp
```

```
$ wc hello bye
       1       2      12 hello
       1       2       9 bye
       2       4      21 total
```

Get Regular ExPressions

```
$ grep ll tmp
hallo
```

```
$ grep o tmp
hallo
world
```

```
$ grep ll hello tmp
hello:hello world
tmp:hallo
```

CoMPare 2 files

```
$ cmp tmp tmp
```

```
$ cmp tmp hello
tmp hello differ: char 2, line 1
```

DIFFerences of 2 files

```
$ diff tmp hello
1,2c1
< hallo
< world
---
> hello world
```


SORT lines

```
$ sort tmp
hallo
world
```

```
$ sort bye hello
good bye
hello world
```

```
$ sort tmp tmp
hallo
hallo
world
world
```

```
$ sort -r tmp tmp
world
world
hallo
hallo
```

TAIL of file

```
$ tail -1 tmp
world
```

```
$ tail +2 tmp
world
```

`ls`	alle Dateien listen (dir)
`ls` datei ...	ausgewählte Dateien listen
`ls -l`	listen der Dateiattribute
`man` befehl	on line manual
`vi` datei	Standard Editor
`cp` datei1 datei2	kopieren
`mv` datei1 datei2	umbenennen
`rm` datei ...	löschen
`cat` datei ...	ausgeben
`wc` datei ...	Zeilen, Worte und Zeichen zählen
`grep` muster datei ...	Zeilen ausgeben, in denen ein Muster enthalten ist
`sort` datei ...	Dateien zeilenweise sortieren
`tail` datei	letzten 10 Zeilen einer Datei ausgeben
`cmp` datei1 datei2	Dateien vergleichen
`diff` datei1 datei2	Unterschiede ausgeben

Dateioperationen

Change working Directory

```
$ cd /usr; pwd
//node_25e48/usr

$ cd; pwd
//node_25e48/sys5.3/usr/uwe/unix
```

MaKe DIRectory

```
$ mkdir d1; cd d1; pwd
//node_25e48/sys5.3/usr/uwe/unix/d1
$ ls
```

Eltern– und Arbeits– directory

```
$ ls ..
bye     d1    hallo

$ cp ../hallo hallo; ls
hallo

$ ls .
hallo
$ ls ./././.
hallo
```

ReMove DIRectory

```
$ cd ..
$ rmdir d1
rmdir: d1: Directory not empty

$ rm d1/hallo; rmdir d1
```

wildcards

```
$  ls
ch   ch1.1   ch1.3   ch2.2  tmp
ch.o ch1.2   ch2.1   change

$ ls ch*
ch   ch1.1   ch1.3   ch2.2
ch.o ch1.2   ch2.1   change

$ wc ch*
       1       2       8 ch
       1       2      10 ch.o
       1       2      11 ch1.1
       1       2      11 ch1.2
       1       2      11 ch1.3
       1       2      11 ch2.1
       1       2      11 ch2.2
       1       2      12 change
       8      16      85 total

$ rm tmp*
$ ls ch*.*
ch.o  ch1.1  ch1.2  ch1.3  ch2.1  ch2.2

$ ls ch[12].[1-3]
ch1.1 ch1.2  ch1.3  ch2.1  ch2.2

$ ls ch?.*
ch1.1 ch1.2  ch1.3  ch2.1  ch2.2
```


output redirection

```
$ ls > filelist
$ cat filelist
bye
filelist
hallo

$ cat hallo > hello
$ cat hello
hello world

$ cat hello bye > tmp
$ cat tmp
hello world
good bye

$ echo hallo > tmp
$ echo world >> tmp
$ cat tmp
hallo
world
```

input redirection

```
$ ls > tmp
$ sort -r < tmp
tmp
hello
hallo
filelist
bye

$ who > tmp
$ grep uwe < tmp
uwe                    display     Sep 11 13:16

$ sort tmp
uwe                    display     Sep 11 13:16
$ sort < tmp
uwe                    display     Sep 11 13:16
```


pipes

```
$ ls | sort -r
change
ch2.2
ch2.1
ch1.3
ch1.2
ch1.1
ch.o
ch

$ ls | wc -l
       8
$ ls | grep ch2 | wc -l
       2
```

Kommando–Sequenzen

```
$ date; who
Tue Sep 11 14:20:06     1990
uwe                    display     Sep 11 13:16
```

i/o redirection

```
$ cat ch* > tmp
$ sort -r < tmp > tmp1
$ diff tmp tmp1
1,7d0
< file ch
< ........
< file ch2.2
8a2,8
> file ch2.2
> ........
> file ch
```

SET: listet alle shell variablen

```
$ set
DISPLAY=:0
HOME=//node_25e48/sys5.3/usr/uwe/unix
IFS=

ISP=m68k
LOGNAME=uwe
NODEID=25E48
NODETYPE=DN3500
ORGANIZATION=none
PATH=:/usr/ucb:/bin:/usr/bin:/usr/apollo/bin
PROJECT=none
PS1=$
PS2=>
SHELL=/bin/sh
SYSTYPE=bsd4.3
TERM=apollo_15_color
TZ=   -2
TZNAME=   ,
USER=uwe
```


eigener prompt string

```
$ PS1="noch ein Kommando ? "
noch ein Kommando ? PS1=´$ ´
$
```

Redefinition der PATH Variablen

```
$ ls -l progs/*
-rwxrwxrwx   1 uwe          2900 Sep 11 14:54 progs/c1

$ c1
c1: not found

$ echo $PATH
:/usr/ucb:/bin:/usr/bin:/usr/apollo/bin

$ PATH=$PATH:$HOME/progs
$ echo $PATH
:/usr/ucb:/bin:/usr/bin:/usr/apollo/bin://node_25e48/sys5.3
/usr/uwe/unix/progs

$ c1
Mein 1. C-Programm
```

Abkürzungen mit shell–Variablen

```
$ d=$HOME/progs
$ echo $d
//node_25e48/sys5.3/usr/uwe/unix/progs

$ $d/c1
Mein 1. C-Programm
```

2 Filesystem

Inhalt

```
$ echo hello > muell
$ echo world >> muell

$ ls -l muell
-rw-rw-rw-   1 uwe            12 Sep 11 15:04 muell

$ cat muell
hello
world

$ od -c muell
0000000    h   e   l   l   o  \n   w   o   r   l   d  \n
0000014

$ od -cb muell
0000000    h   e   l   l   o  \n   w   o   r   l   d  \n
         150 145 154 154 157 012 167 157 162 154 144 012
0000014

$ file . muell
.:     directory
muell: unstruct ascii text
```

Hirarchie

```
$ mkdir rezepte; cd rezepte; pwd
//node_25e48/sys5.3/usr/uwe/unix/rezepte

$ mkdir kuchen kekse torten
$ echo "man nehme 5 Aepfel,..." > kuchen/apfel
$ echo "man nehme 20 Mandeln,..." > kekse/mandel
$ echo "man nehme 3 kg Kaese,..." > kuchen/kaese

$ cd; pwd
//node_25e48/sys5.3/usr/uwe/unix

$ find rezepte -name apfel -print
rezepte/kuchen/apfel
```

Zugriffsrechte von directories

```
$ ls -ld .
drwxrwxrwx   1 uwe          1024 Sep 11 15:06 .
```

schreiben in eine directory

```
$ who > .
.: cannot create
```


CHange MODe: Zugriffsrechte ändern

```
$ echo ´ls -l´ > l
$ cat l
ls -l

$ ls -l l
-rw-rw-rw-   1 uwe             6 Sep 11 15:15 l

$ chmod +x l
$ l
total 2
-rwxrwxrwx   1 uwe             6 Sep 11 15:15 l
drwxrwxrwx   1 uwe          1024 Sep 11 15:07 rezepte
```

inodes

```
$ who > muell
$ ls -l muell
-rw-rw-rw-   1 uwe            29 Sep 11 15:16 muell

$ date > muell
$ ls -l muell
-rw-rw-rw-   1 uwe            29 Sep 11 15:17 muell

$ ls -lu muell
-rw-rw-rw-   1 uwe            29 Sep 11 15:16 muell

$ ls -lc muell
-rw-rw-rw-   1 uwe            29 Sep 11 15:17 muell

$ chmod -w muell
$ ls -l muell
-r--r--r--   1 uwe            29 Sep 11 15:17 muell

$ ls -lu muell
-r--r--r--   1 uwe            29 Sep 11 15:16 muell

$ ls -lc muell
-r--r--r--   1 uwe            29 Sep 11 15:21 muell
```

links

```
$ date > muell
$ ln muell oskar

$ cat muell oskar
Tue Sep 11 15:47:47     1990
Tue Sep 11 15:47:47     1990

$ ls -l [mo]*
-rw-rw-rw-   2 uwe            29 Sep 11 15:47 muell
-rw-rw-rw-   2 uwe            29 Sep 11 15:47 oskar

$ diff muell oskar

$ cp muell mehrmuell

$ date > muell
$ diff muell mehrmuell
1c1
< Tue Sep 11 15:48:45     1990
---
> Tue Sep 11 15:47:47     1990
$ rm muell
$ ls -l o*
-rw-rw-rw-   1 uwe            29 Sep 11 15:48 oskar
```


`/`	Wurzel des Dateisystems
`/bin`	ausführbare Programme für das BS
`/dev`	Gerätedateien
`/dev/dsk`	Gerätedateien für Platten
`/etc`	Systemtabellen, –programme
`/etc/passwd`	Passworddatei
`/lib`	wesentliche Bibliotheken
`/tmp`	temoräre Dateien werden bei Systemstart gelöscht
`/unix`	ausführbarer BS–Kern
`/usr`	Benutzerkatalog
`/usr/bin`	ausfühtbare Systemprogramme (cc, make,. . .)
`/usr/include`	header–Dateien für cc
`/usr/include/sys`	System–header–Dateien für cc
`/usr/lib`	Bibliotheken für cc,. . .
`/usr/man`	online–manual
`/usr2`	weiterer Benutzerkatalog
`/usr/you`	Benutzerkatalog
`/usr/you/bin`	eigene Programme

Interessante Kataloge und Dateien

3 *shell*–Benutzung

pipes

```
$ who | wc
       1       5      46
```

Sequenzen

```
$ date;who
Tue Sep 11 17:19:47     1990
uwe                  display     Sep 11 13:16
```

Sequenzen und pipes

```
$ date; who | wc
Tue Sep 11 17:20:02     1990
       1       5      46

$ (date; who) | wc
       2      10      75
```

tee

```
$  (date; who) | tee save | wc
       2      10      75
$ cat save
Tue Sep 11 17:21:49     1990
uwe                  display     Sep 11 13:16
$ wc < save
       2      10      75
```

batch jobs

```
$ wc > tmp &
918
$ (date; sleep 70; date) > tmp &
919
$ (sleep 300; echo Tee ist fertig) &
922
```

quoting

```
$ echo ***
hello muell rezepte

$ echo ´***´
***

$ echo \*\*\*
***

$ echo \´´***´\´
´***´

$ echo "´***´"
´***´

$ echo ´hello
> world´
hello
world
```

eigene Kommandos

```
$ echo ´who | wc -l´ > nu
$ cat nu
who | wc -l

$ sh < nu
       1
$ sh nu
       1

$ chmod +x nu
$ nu
       1
```


eigene Kommando directory

```
$ pwd
//node_25e48/sys5.3/usr/uwe/unix
$mkdir bin
$ mv nu bin
```

Kommandos mit Parametern

```
$ echo ´chmod +x $1´ > cx

$ sh cx cx

$ ls -l cx
-rwxrwxrwx   1 uwe            12 Sep 11 17:27 cx

$ mv cx bin
$ Tee ist fertig
```

Kommando hello

```
echo ´echo hello world´ > hello

$ cx hallo
chmod: hallo: No such file or directory

$ cx hello
$ hello
hello world
```

Kommandos mit mehreren Parametern

```
$ echo ´: #lc : Anzahl Zeilen zaehlen
> wc -l $*´ > lc

$ cx lc

$ lc hello lc
       1 hello
       2 lc
       3 total
$ mv lc bin
```

Telephonliste

```
$ echo ´grep $* telefon´ > 118

$ cat telefon
FH Wedel    043103-8048-0
Uwe                    -45
Nele        8134
Dr.Markus   441777
Auskunft    118

$ cx 118

$ 118 Nele
Nele        8134

$ 118 FH Wedel
Wedel: No such file or directory
telefon:FH Wedel    043103-8048-0
```

2.Versuch mit quoting

```
$  echo ´grep "$*" telefon´ > 118

$  118 FH Wedel
FH Wedel    043103-8048-0

$ 118 nele
```

3.Versuch: Groß– und Kleinschreibung

```
$   echo ´grep -y "$*" telefon´ > 118

$ 118 nele
Nele        8134
```


back–Tüdel

```
$ echo "Beim letzten Ton des Zeitzeichens
> ist es genau `date`"
Beim letzten Ton des Zeitzeichens
ist es genau Tue Sep 11 17:36:27     1990

$  echo ´Beim letzten Ton des Zeitzeichens
> ist es genau `date`´
Beim letzten Ton des Zeitzeichens
ist es genau `date`

$ echo `echo \`date\``
Tue Sep 11 17:37:17 1990

$ echo `echo date`
date
```

shell–Variablen

```
$ dir=`pwd`

$ echo $dir
//node_25e48/sys5.3/usr/uwe/unix

$ cd rezepte
$ pwd
//node_25e48/sys5.3/usr/uwe/unix/rezepte

$ cd $dir
$ pwd
//node_25e48/sys5.3/usr/uwe/unix
```

exportierte shell–Variablen

```
$ sh
$ echo $dir

$ CTRL/d *** EOF ***

$ export dir

$ sh
$ echo $dir
//node_25e48/sys5.3/usr/uwe/unix
$ CTRL/d *** EOF ***

$ echo $dir
//node_25e48/sys5.3/usr/uwe/unix
```


$prog > file$	Standard–Ausgabe für *prog* umlenken in *file*
$prog >> file$	Standard–Ausgabe für *prog* umlenken und an *file* anhängen
$prog < file$	Standard–Eingabe für *prog* aus *file*
$p_1 \vert p_2$	*pipe*: Standard–Ausgabe von p_1 als Standard–Eingabe für p_2
$prog << str$	Hier–Dokument: Standard–Eingabe für *prog* folgt von hier an bis *str* aleine auf einer Zeile steht
$*$	steht für 0 oder beliebig viele Zeichen in Dateinamen
?	steht für ein beliebiges Zeichen in Dateinamen
[*ccc*]	steht für ein Zeichen aus der in [und] aufgezählten Zeichen
$p_1; p_2$	Komandosequenz
&	Kommando ohne Warten (im Hintergrund) ausführen
$1 ... $9	Parameter 1 bis 9
$0	Programmname
$*	alle Parameter (außer $0)
c	Zeichen wird uninterpretiert übernommen
'...'	Zeichenfolge wird uninterpretiert übernommen
"..."	in Zeichenfolge Variablen ersetzen und ‘...‘ auswerten
‘...‘	Zeichenfolge als Kommando interpretieren und durch deren Ausgabe ersetzen
#	Kommentar bis zum Zeilenende
$var = wert$	Zuweisung an eine Variable
$*var*	Auslesen einer Variablen

shell–**Metazeichen**

stderr

```
$ time wc telefon
       6      11     106 telefon
        0.1 real         0.0 user         0.0 sys

$ time wc telefon > /dev/null
        0.1 real         0.0 user         0.0 sys

$ time wc telefon > /dev/null 2> time.out
$ cat time.out
        0.1 real         0.0 user         0.0 sys

$ time wc telefon > time.out 2>&1
$ cat time.out
       6      11     106 telefon
        0.1 real         0.0 user         0.0 sys
```

inline documents

```
$ cat 118
grep -y "$*" <<´End´
FH Wedel    043103-8048-0
Uwe                   -45
Nele        8134
Dr.Markus   441777
Auskunft    118
End

$ 118 nele
Nele        8134
```


for loops

```
$ for i in *
> do
> echo $i
> done
118
bin
hello
rezepte
telefon
tmp

$ for i in t* ; do chmod +x $i; done
$ ls -l t*
-rwxrwxrwx   1 uwe           106 Sep 11 17:32 telefon
-rwxrwxrwx   1 uwe            58 Sep 11 17:24 tmp

$ for i in t* ; do cat $i; done | wc
       8      21     191
```

bundle

```
$ cat bundle
#!/bin/sh
#bundle: Dateien zwecks Versand zusammenpacken
echo ´#!/bin/sh´
echo ´# zum Auspacken, sh auf diese Datei anwenden´
for i in $*
do
   echo "echo $i 1>&2"
   echo "cat >$i <<´Ende von $i´"
   cat $i
   echo "Ende von $i"
done
```

Aufruf von bundle

```
$ bundle cx lc > scripts.sh

$ cat scripts.sh
#!/bin/sh
# zum Auspacken, sh auf diese Datei anwenden
echo cx 1>&2
cat >cx <<´Ende von cx´
chmod +x $1
Ende von cx
echo lc 1>&2
cat >lc <<´Ende von lc´
#!/bin/sh #lc : Anzahl Zeilen zaehlen
wc -l $*
Ende von lc
```


auspacken

```
$ mkdir script.save; mv scripts.sh script.save;
$ cd sc*.save

sh sc*
cx
lc
```

4 Filter

grep

```
$  ls -l
total 3
drwxrwxrwx   1 uwe        1024 Sep 13 13:21 bin
drwxrwxrwx   1 uwe        1024 Sep 11 15:07 rezepte
-r--r--r--   1 uwe           1 Sep 13 13:23 tmp

$ ls -l | grep ´^d´
drwxrwxrwx   1 uwe        1024 Sep 13 13:21 bin
drwxrwxrwx   1 uwe        1024 Sep 11 15:07 rezepte

$ ls -l | grep ´^.......rw´
drwxrwxrwx   1 uwe        1024 Sep 13 13:21 bin
drwxrwxrwx   1 uwe        1024 Sep 11 15:07 rezepte

$ ls -l /usr/dict/words
-rwxr-xr-x+  2 root     201039 Feb 27  1988 /usr/dict/words
```

egrep

```
$ cat vowels
^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*$

$ egrep -f vowels /usr/dict/words
facetious

$ cat monoton
^a?b?c?......?z?$

$ egrep -f monoton /usr/dict/words
a
abc .............

$ egrep -f monoton /usr/dict/words | grep ´......´
almost
biopsy
```

SORT files

```
$ ls -l
total 4
drwxrwxrwx   1 uwe          1024 Sep 13 13:21 bin
-rw-rw-rw-   1 uwe            55 Sep 13 13:32 monoton
-rw-rw-rw-   1 uwe            62 Sep 13 13:30 vowels

$ ls -l | sort +4
total 4
drwxrwxrwx   1 uwe          1024 Sep 13 13:21 bin
-rw-rw-rw-   1 uwe            62 Sep 13 13:30 vowels
-rw-rw-rw-   1 uwe            55 Sep 13 13:32 monoton

$ ls | sort -r
vowels
monoton
bin
```


TRanslate

```
$ echo Hallo | tr a-z A-Z
HALLO

$ echo Hallo | tr A-Z a-z
hallo

$ echo "hello world, good bye!" | tr -sc A-Za-z ´\012´
hello
world
good
bye
```

uniq

```
$ cat tmp
hello
hello
world

$ uniq tmp
hello
world

$ uniq -u tmp
world

$ uniq -d tmp
hello
```

Prototyp: Rechtschreibhilfe

```
$ cat duden
#!/bin/sh
# Rechtschreibhilfe
# extrahiert aus der Eingabe alle Woerter und ueberprueft,
# ob diese in dem Woerterbuch in der Datei DUDEN stehen.
# Gross- und Kleinschreibung wird nicht unterschieden

#alle Woerter klein ohne Duplikate in tmp1
cat $* | tr A-Z a-z | tr -sc A-Za-z ´\012´ \
| sort -u > tmp1

#alle bekannten Woerter in tmp2
sort tmp1 DUDEN | uniq -d > tmp2

#alle unbekannten Woerter ausgeben
sort tmp1 tmp2  | uniq -u

#aufraeumen
rm tmp1 tmp2
```


ein Text

```
$ cat text

Ein Retzel

Was hat so fil Nadeln wie
im nekasten die schdeknadeln ?

was hat so fil estische wie manchmal
in der schdube.

weist dus nicht, wo kerzen dran sind
nah jetz weist du es

$ duden text
dus
estische
fil
jetz
nekasten
retzel
schdeknadeln
schdube
```

!!! nicht alle Fehler gefunden !!!

Stream EDitor: sed

```
$ who
uwe                     display     Sep 13 13:17

$ who | sed ´s/ .* / /´
uwe 13:17
```

umständlich: Groß ⇒Kleinbuchstaben

```
$ cat $* | sed ´s/A/a/g´ | sed ´s/B/b/g´ | ....
... | sed ´s/Z/z/g´
```

duden 2.Version

```
$ cat duden
...
#einzelne Buchstaben loeschen

cat $* \
| tr A-Z a-z \
| tr -sc A-Za-z ´\012´ \
| sed ´/^.$/d´ \
| sort -u > tmp1

...
```

testlauf

```
$ cat tmp
a bc d ef
$ duden tmp
bc
ef
```

5 shell–Programmierung

echo3 1.Version

```
$ cat echo3
#!/bin/sh
# echo3 - 1. Version

echo "$0 wurde mit $# Parameter(n) aufgerufen."
echo "Die ersten drei Parameter haben die Werte:"
echo "   $1"
echo "   $2"
echo "   $3"
```

ein Aufruf mit 3 Parametern

```
$ echo3 a b c
echo3 wurde mit 3 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
   a
   b
   c
```

ein Aufruf mit einem Parameter

```
$ echo3 "a b c"
echo3 wurde mit 1 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
   a b c


$
```

echo3 2.Version

```
$ cat echo3
#!/bin/sh
# echo3 - 2. Version

echo "$0 wurde mit $# Parameter(n) aufgerufen."
echo "Die ersten drei Parameter haben die Werte:"
echo "   $1" ;shift
echo "   $1" ;shift
echo "   $1"
```

ein Aufruf mit 3 Parametern

```
$ echo3 a b c
echo3 wurde mit 3 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
   a
   b
   c
```

ein Aufruf mit einem Parameter

```
$ echo3 "a b c"
echo3 wurde mit 1 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
   a b c

echo3: cannot shift
```


Programm `test` *zur Auswertung von Prädikaten*
`test -r` *< datei > : Gibt es eine lesbare Datei?*
noch nicht

```
$ ls muell
muell not found
$ test -r muell; echo $?
1
```

aber jetzt

```
$ echo > muell
$ test -r muell; echo $?
0
```

`test` *ohne Argumente*

```
$ test; echo $?
1
```

abkürzende Schreibweise für das Programm `test`
`[` *ist der Programmname,*
`]` *das letzte Argument*

```
$ cat tonne
tonne: No such file or directory

$ [ -r tonne ] && cat tonne

$ echo "oskar" > tonne
$ [ -r tonne ] && cat tonne
oskar
```

echo3 3.Version

```
$ cat echo3
#!/bin/sh
# echo3 - 3. Version

echo "$0 wurde mit $# Parameter(n) aufgerufen."
echo "Die ersten drei Parameter haben die Werte:"

if test $1
then
  echo "   $1"; shift
  if test $1
  then
    echo "   $1";shift
    if test $1
    then
      echo "   $1"
    fi
  fi
fi
```

Aufruf mit 3 Parametern

```
$ echo3 a b c
echo3 wurde mit 3 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
   a
   b
   c
```

Aufruf mit einem Parameter

```
$ echo3 "a b c"
echo3 wurde mit 1 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
test: too many arguments
```

echo3 4.Version
< arg_1 > `-gt` *< arg_2 > : arithmetischer Vergleich >*

```
$ cat echo3
#!/bin/sh
# echo3 - 4. Version

echo "$0 wurde mit $# Parameter(n) aufgerufen."
echo "Die ersten drei Parameter haben die Werte:"

[ $# -gt 0 ] && echo $1 && shift
[ $# -gt 0 ] && echo $1 && shift
[ $# -gt 0 ] && echo $1
```

ein Aufruf mit 3 Parametern

```
$ echo3 a b c
echo3 wurde mit 3 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
a
b
c
```

ein Aufruf mit einem Parameter

```
$ echo3 "a b c"
echo3 wurde mit 1 Parameter(n) aufgerufen.
Die ersten drei Parameter haben die Werte:
a b c
```


echo3 5.Version

```
$ cat echo3
#!/bin/sh
# echo3 - 5. Version

echo "$0 wurde mit $# Parameter(n) aufgerufen."
echo "Die Parameter haben die Werte:"

while [ $# -gt 0 ]
do
   echo "   $1" ;shift
done
```

ein Aufruf mit 4 Parametern

```
$ echo3 a b c d
echo3 wurde mit 4 Parameter(n) aufgerufen.
Die Parameter haben die Werte:
   a
   b
   c
   d
```

ein Aufruf mit einem Parameter

```
$ echo3 "a b c d"
echo3 wurde mit 1 Parameter(n) aufgerufen.
Die Parameter haben die Werte:
   a b c d
```

Parameter–Interpretation
ein Skript `options`

```
$ cat options
#!/bin/sh
# Beispiel zur Interpretation von Optionen
# mit einer Schleife und einer case Anweisung

while [ $# -gt 0 ]
do
 case $1 in
  -c | -o) echo "Option -c oder -o"
    ;;
  -*) echo "unbekannte Option $1"
    ;;
  *.c | *.h ) echo "C Quelle $1"
    break;
    ;;
  *) echo "unbekannter Parameter $1"
    exit 1
 esac
 shift
done
```

ein Aufruf mit 3 Optionen und einem legalen Parameter

```
$ options -c -o -- c.c
Option -c oder -o
Option -c oder -o
unbekannte Option --
C Quelle c.c
```

der return code

```
$ echo $?
0
```


ein Aufruf mit einem fehlerhaften Parameter

```
$ options oskar
unbekannter Parameter oskar

$ echo $?
1
```

debugging von scripts
die `-x` *Option für die* `sh`

```
$ sh -x options -c c.c
+ : Beispiel zur Interpretation von Optionen
+ [ 2 -gt 0 ]
+ echo Option -c oder -o
Option -c oder -o
+ shift
+ [ 1 -gt 0 ]
+ echo C Quelle c.c
C Quelle c.c
+ break
```

bundle 2.Version

```
$ cat bundle
#!/bin/sh
# Bourne shell
# bundle: Dateien zwecks Versand zusammenpacken
# Ausgabe auf stdout
# erweiterte Version fuer directories

# trace Ausgabe
echo "$0 : fuer dateien $*" 1>&2

echo ´#zum Auspacken, sh auf diese Datei anwenden´
for i in $*
do
  if [ -f $i ] && [ -r $i ] && [ -s $i ]
  then
                      # trace Ausgabe im erzeugten Skript
    echo "echo $i 1>&2"
    echo "cat >$i <<´Ende von $i´"
    cat $i
    echo "Ende von $i"
  else
  if [ -d $i ]
  then
    $0 $i/*                                #rekursiver Aufruf
  else
    echo "# datei $i konnte nicht kopiert werden"
                                             # Fehler Ausgabe
    echo "datei $i kann nicht kopiert werden"     1>&2
  fi
  fi
done
```


alle Skripts im eigenen Verzeichnis `bin` *bündeln*

```
$ cd
$ bundle bin >tmp
bundle : fuer dateien bin
bundle : fuer dateien bin/118 bin/bundle.1 ....
datei bin/empty kann nicht kopiert werden

$ ls -l bin/empty
-rw-rw-rw-   1 uwe             0 Sep 13 12:46 bin/empty
```

das skript ist nicht sicher

```
$ echo "Ende von emil" > emil
$ bundle emil > tmp
$ sh tmp
emil
tmp: Ende: not found
```

bundle 3.Version: sicher

```
$ cat bundle
#!/bin/sh
# Bourne shell

echo ´#!/bin/sh´
echo ´# zum Auspacken /bin/sh auf diese Datei anwenden´
for i in $*
do
  if [ -f $i ] && [ -r $i ] && [ -s $i ]
  then
    echo "echo $i 1>&2"
    echo "sed ´s/^+//´ >$i <<´Ende von $i´"
    sed ´s/^/+/´ $i
    echo "Ende von $i"
  else
  if [ -d $i ]
  then
    echo "[ -d $i ] || mkdir $i"
    $0 $i/*
  else
    echo "# datei $i konnte nicht kopiert werden"
    echo "datei $i kann nicht kopiert werden" 1>&2
  fi
  fi
done
```


jetzt ok

```
$ echo "Ende von emil" > emil
$ bundle emil > tmp
$ sh tmp
emil
$ cat tmp
#!/bin/sh #zum Auspacken /bin/sh auf diese Datei anwenden
echo emil 1>&2
sed ´s/^+//´ >emil <<´Ende von emil´
+Ende von emil
Ende von emil
```

ein Skript `edit-cc`
zur Erleichterung des Edit–Compile Zyklus
die Beschreibung

```
#!/bin/sh Bourne shell

# this script takes logfiles of compile or make jobs as
# input, extracts the cc error messages
# then errors are inserted at the apropriate positions
# within temporary copies of the source files
# and the vi editor is called with a search pattern
# for the error messages and all files containing errors
# editing is done in a tmp directory
# after editing the updated sources can be written back
# to the original sources

# set -x  # trace flag for debugging the skript

if [ $# -eq 0 ]
   then
   echo "usage: $0 <logfile>..."
   exit 1
fi
```

die Muster zur Fehlermarkierung
und für die Syntax der Fehlermeldungen
Syntax der Fehlermeldungen: ``< *datei* >`` , line < *zeile* >: < *text* >

```
errorpattern=´-----error-----´

pattern1=´^"[^"]*", line ´
pattern2="$pattern1"´[0-9][0-9]*:´
```


die Hilfsdateien, Namen erzeugen, anlegen und löschen

```
tmpdir=/tmp/$$
tmpfile=$tmpdir

trap "rm -f -r $tmpdir"´*´"; exit" 0 1 2 9 15

mkdir $tmpdir
```

die Fehlermeldungen herausfiltern

```
cat $* | grep "$pattern2" > $tmpfile.1
```

die Dateinamen der fehlerhaften Dateien herausfiltern

```
files=`cat $tmpfile.1 | sed -e ´s/^"//´ -e ´s/".*$//´ \
                      | sort -u`
tmpfiles=´´
```

die Fehler an die richtigen Stellen einfügen
a) Zeilennummern erzeugen
b) Fehlermeldungen einsortieren
c) Zeilennummern löschen

```
for i in $files
  do
    pr -n:1 -t $i > $tmpfile.2
    grep ´^"´$i´"´  $tmpfile.1 |                            \
    sed -e ´s/´"$pattern1"´//´                              \
        -e ´s/:/:´"$errorpattern"´/´                        \
                                          >> $tmpfile.2
    sort +n $tmpfile.2 | cut -f2- -d:  > $tmpdir/$i
    tmpfiles="$tmpfiles $tmpdir/$i"
  done
```

der Editoraufruf

```
vi ´+/´"$errorpattern" $tmpfiles
```

die Rückfrage zum Überschreiben der alten Quellen

```
for i in $files
  do
    echo -n "overwrite $i with $tmpdir/$i (y/n) "
    read ok
    if [ "$ok" = ´y´ ]
    then
      cat $tmpdir/$i | grep -v ´^´"$errorpattern" > $i
    else
      echo "file $i not changed" 1>&2
    fi
  done
```


zwei C Quelldateien
`c.c`

```
#include "c.h"

main(argc,argv)
{ int n,mn;
  n=mn;
  return emil;
}
```

`c.h`

```
int i = i+1;
int k = kk;
```

Compileraufruf: `cc -c c.c`
das Fehlerlisting Datei:`errors`

```
"./c.h", line 1: illegal initialization
"./c.h", line 2: warning: undeclared initializer name kk
"./c.h", line 2: illegal initialization
"c.c", line 6: emil undefined
```

ein Aufruf: `edit-cc errors`
mit trace Ausgabe `set -x`

```
+ : Bourne shell
+ [ 1 -eq 0 ]

  errorpattern=-----error-----
  pattern1=^"[^"]*", line
  pattern2=^"[^"]*", line [0-9][0-9]*:

  tmpdir=/tmp/129
  tmpfile=/tmp/129
+ trap rm -f -r /tmp/129*; exit 0 1 2 9 15
+ mkdir /tmp/129

+ cat errors
+ grep ^"[^"]*", line [0-9][0-9]*:

+ cat /tmp/129.1
+ sed -e s/^"// -e s/".*$//
+ sort -u
  files=./c.h c.c

  tmpfiles=
+ pr -n:1 -t ./c.h
+ grep ^"./c.h" /tmp/129.1
+ sed -e s/^"[^"]*", line // -e s/:/:-----error-----/
+ sort +n /tmp/129.2
+ cut -f2- -d:
  tmpfiles= /tmp/129/./c.h
```



```
+ pr -n:1 -t c.c
+ grep ^"c.c" /tmp/129.1
+ sed -e s/^"[^"]*", line // -e s/:/:-----error-----/
+ sort +n /tmp/129.2
+ cut -f2- -d:
  tmpfiles= /tmp/129/./c.h /tmp/129/c.c

+ vi +/-----error----- /tmp/129/./c.h /tmp/129/c.c
```

die tmpfiles beim Editieren
`ls -l /tmp`

```
total 3
drwxr-xr-x   2 uwe      other         64 Nov 12 08:30 129
-rw-r--r--   1 uwe      other        167 Nov 12 08:30 129.1
-rw-r--r--   1 uwe      other        119 Nov 12 08:30 129.2
```

`cat /tmp/129/c.c`

```
#include "c.h"

main(argc,argv)
{ int n,mn;
  n=mn;
  return emil;
-----error----- emil undefined
}
```

`cat /tmp/129/c.h`

```
int i = i+1;
-----error----- illegal initialization
int k = kk;
-----error----- illegal initialization
-----error----- warning: undeclared initializer name kk
```

die Fortsetzung des trace

```
+ echo -n overwrite ./c.h with /tmp/129/./c.h (y/n)
+ read ok
+ [ n = y ]
+ echo file ./c.h not changed
  file ./c.h not changed

+ echo -n overwrite c.c with /tmp/129/c.c (y/n)
+ read ok
+ [ n = y ]
+ echo file c.c not changed
  file c.c not changed
+ rm -f -r /tmp/129 /tmp/129.1 /tmp/129.2
+ exit
```

6 Tcl

Tcl Tool Command Language

↪ eine *shell*, die in ein Anwendungsprogramm eingebunden werden kann, z.B. in ein C- oder C++–Programm

↪ Ein-/Ausgabe

- ↪ parsen der Eingabe
- ↪ Plausibilitätsüberprüfung
- ↪ Fehlererkennung und –behandlung

mit vorgefertigten Tcl–Kommandos

↪ eigentliche *Programmierarbeit* nur zur Implementierung der reinen Funktionalität

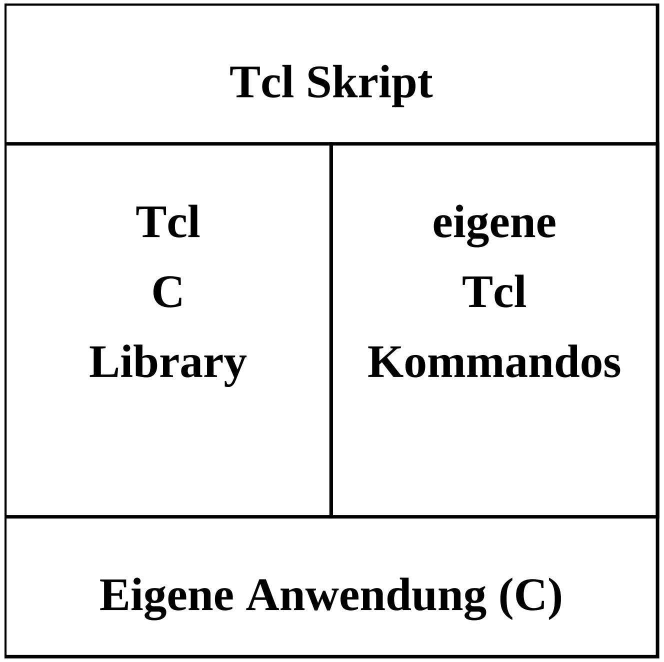
Tcl

Syntax	`<command> <arg1> ... <argn>`
↪	Tcl–Programm: eine Folge von Kommandos
strings	alles wird als *string* dargestellt ↪ Programm ↪ Daten ↪ Parameter ↪ Resultate

Beispiele	`set x 42`
	`puts stdout Hallo`
	`proc p x {puts stdout $x}`

Argumente	Zeichenreihen, die durch Leerzeichen und Tabs getrennt sind
{...}	ein Argument, das Sonderzeichen (Leerzeichen, Tab, Zeilenumbruch, $, [,], ...) enthalten kann Sonderzeichen werden nicht interpretiert
"..."	ein Argument, das Sonderzeichen enthalten kann diese werden interpretiert
[...]	ein Argument, das als Tcl–Kommando interpretiert wird und durch sein Resultat ersetzt wird
$name	Auslesen einer Variablen

Architektur



tclsh	eine vorgefertigte Anwendung mit allen eingebauten Tcl–Kommandos
Tcl	gut für textbasierte, kommmandozeilenorientierte Anwendungen
↪	Tcl–Kommandos als Kommunikationsprotokoll zwischen Anwendungen

7 HTML

8 CGI

9 Prozesse und Signale

Prozess–Nummer

```
 $ echo $$
176
```

Prozess–Status

```
$ ps
  PID TTY     STAT  TIME COMMAND
  180 pad01   R     0:00 ps
```

Status BSD4.3

```
$ ps -l
      F   UID   PID  PPID CP PRI NI ADDR ...
1008001    18   188   176  0   3  0    a

...  SZ  RSS STAT TTY      TIME COMMAND
    640   80 R    pad01    0:00 ps -l
```

Status SYS5

```
$ ver sys5.3
$ ps -l
 F S   UID   PID  PPID  C PRI NI ADDR  ...
 1 R    18   189   176  0   3 20   10
 1 S    18   176     1  0   3 20    9

...    SZ TTY       TIME COMD
      672 pad01     0:00 ps
      672 pad01     0:00 $(shell)
```

dynamisches Kreieren

```
$ sh
$ ps
  PID TTY     STAT  TIME COMMAND
  192 pad01   S     0:00 sh
  193 pad01   R     0:00 ps
$ sleep 300 &
199
$ ps
  PID TTY     STAT  TIME COMMAND
  192 pad01   S     0:00 sh
  199 pad01   S     0:00 sleep 300
  200 pad01   R     0:00 ps
```

Prozesse beenden

```
$ kill 199
$
199 Terminated

$ sleep 300 &
202
$ kill -9 202
$
202 Killed
```


Prozesse überlagern

```
$ echo $$
192

$ sh
$ echo $$
211

$ exec date
Thu Sep 27 21:11:56 UTC 1990

$ echo $$
192
```

temporäre Dateien

```
$ date > /tmp/date-$$
$ ls -l /tmp/date-$$
-rw-rw-rw-   1 uwe          29 Sep 27 21:12 /tmp/date-192

$ cat /tmp/date-$$
Thu Sep 27 21:12:32 UTC 1990

$ rm /tmp/date-$$
```

*Prozessabbruch mit CTRL/C und CTRL/ *

```
$ cat loop.c
main(){while (1);}

$ cc -o loop loop.c

$ loop
CTRL/\ Quit
$
```

Signal–Behandlung in shell skrips

```
$ cat loop.sh
#!/bin/sh Bourne shell
while [ 1 = 1 ]
do
  date >> /tmp/date-$$
  sleep 10
done
rm /tmp/date--$$

$ loop.sh &
268

$ cat /tmp/date*
Thu Sep 27 21:23:05 UTC 1990
Thu Sep 27 21:23:16 UTC 1990

$ ls /tmp/date*
/tmp/date-268

$ kill 268

$ ls /tmp/date*
/tmp/date-268
```


trap–Anweisung

```
$ rm /tmp/date*
$ cat loop.sh
#!/bin/sh Bourne shell
trap "rm /tmp/date-$$;exit" 0 1 2 3 15
while [ 1 = 1 ]
do
  date >> /tmp/date-$$
  sleep 10
done

$ loop.sh &
385

$ ls /tmp/date*
/tmp/date-385

$ kill -15 385
$ ls /tmp/date*
/tmp/date* not found
```

10 X–Windows

X–Windows

Eigenschaften

Fenstersystem für

- ↪ Grafik–Bildschirm
- ↪ Tastatur
- ↪ 3–Tasten Maus

Netzwerk in einem Netzwerk transparent einsetzbar

- ↪ mehrere Rechner von einem Bildschirm aus bedienbar
- ↪ Anwendungen können an beliebiger Stelle im Netz laufen

unabhängig von Herstellern und Rechnerarten

portabel

Architektur	
client / server	Konzept
	bei X client / server *vertauscht*
service	grafische Ein- und Ausgabe
client	ein Anwendungsprogramm
↪	ein *server* kann viele *clients* aus dem Netz bedienen
server	ein *UNIX*–Prozeß
↪	der einzige Prozeß, der auf die Grafik–Hardware und Software zugreift
↪	die *clients* sind systemunabhängig und portabel

Architektur	im Detail
X–lib	ist verantwortlich für
↪	reine Grafikaufgaben
↪	Zeichnen von Rechtecken, Linien, Punkten, ...
↪	Textausgabe, Fontauswahl
↪	Farbverwaltung: sw / 256 Farben / 16k Farben / ...
XUI-toolkits	liefern
widget–sets	labels, buttons, menues, scrollbars, Textfenster, popup menues, ...
↪	komplexe grafische Objekte
↪	oft mit einem hierarchischen Aufbau
↪	Verteilung der Eingaben gemäß der (Unter–)Fensteranordnung *widget Hierarchie*
event loop	Schleife, die die Eingaben (Ereignisse, *events*) vom X–server verarbeitet
dispatcher	Ereignis–Verteiler: Zuordnung von *events* zu *widgets*

Aufgabenteilung

X	legt nicht die Benutzungsschnittstelle fest
gut	flexibel: jeder kann nach seinem eigenen Geschmack arbeiten
schlecht	uneinheitlich: kein durchgängiges *look and feel*
XUI-toolkits	legen das Aussehen der grafischen Elemente durch das implementierte *widget set* fest
↪	legt aber nicht die Verwaltung der verschiedenen Anwendungen auf dem Bildschirm fest
window–manager	eine X–Anwendung mit erweiterten Rechten
↪	bestimmt die Verwaltung der verschiedenen Anwendungen
↪	unterschiedliche WMs: ↪ Motif: mwm ↪ OpenLook: olwm, olvwm ↪ Linux: fvwm
Aufgabenteilung	↪ Grafik–Funktionen ↪ grafische Objekte ↪ Fensterverwaltung

Ereignisgesteuerte Programmierung:

- ↪ Was ist das?
- ↪ Was ist anders als bei “gewöhnlicher” Programmierung?
- ↪ Was ist gleich?

Einordnung	in Programm–Klassen
(a) batch	`cc -c prog.c`
↪	keine Interaktion mit dem Benutzer während des Programmlauft
↪	fast kein Aufwand für die Entwicklung der Ein- und Ausgabe
↪	soll es auch heute noch geben

Programm–Klassen

(b) shell in einer Interpretationsschleife werden Kommandos sequentiell abgearbeitet

↪ Interpretationsschleife wie in einer CPU

- ↪ Instruktion holen
- ↪ Instruktion dekodieren
- ↪ Instruktion in Abhängigkeit des Typs interpretieren

↪ nur ein Punkt, an dem eine Eingabe gemacht wird

↪ nur eine Ereignisart tritt ein: *Eingabe einer Zeile*

↪ während der Abarbeitung eines Kommandos verhält sich das Programm wie ein *batch*–Programm

Programm–Schema

```
finished := false;
while not finished
do

    cmd := read_line;

    cmd_type, args := parse_command(cmd);

    case cmd_type of
        t1   : interpret_t1_command(args);
        ...
        tn   : interpret_tn_command(args);
        ...
        quit : finished := true
    end case
end while
```

(c) ereignisgesteuert

in einer Interpretationsschleife werden Ereignisse, *events*, sequentiell abgearbeitet

↪ Programmfluß wie bei dem *shell*–Programm

↪ die Eingabeinformation ist viel feiner gestückelt:

- ↪ ein Tastendruck
- ↪ ein Mausklick
- ↪ eine Mausbewegung

die Verteilung der Eingaben, der *events*, ist sehr, sehr viel aufwendiger

↪ während der Abarbeitung eines *events* verhält sich das Programm, der *event handler*, wie ein *batch*–Programm

die Konstruktion der *event loop* bildet die Hauptaufgabe bei der Entwicklung, die eigentliche Anwendung ist (manchmal) nur noch nebensächlich

Produktivität ???

(c) ereignisgesteuert die *event loop*

```
finished := false;
while not finished
do
      event := get_event;

      event_type,
      event_pos,
      event_val := analyse_event(event);

      widget := select_widget(event_pos);
      case widget of
            w1 :
                  case event_type of
                        t1 : interpret_w1_t1_command(event_val);
                        ...
                        tn : interpret_w1_tn_command(event_val);
                  end case
            ...
            wn :
                  case event_type of
                        t1 : ...
                        tn : ...
                  end case
      end case
end while
```

UI–toolkits

↪ enthalten eine allgemein einsetzbare *event loop* einschließlich *dispatcher* zur Verteilung und Interpretation der *events*

↪ keine feste `case` Anweisung, sondern eine Datenstruktur für die grafischen Objekte und ihre *event handler*

Kommunikation Anwendung ⇔ *toolkit*

.1 Anwendung fordert *widget* bei dem *toolkit* an

↪ oft in Programm–Initialisierungsphase

.2 Anwendung muß Reaktion auf *events* bestimmen

↪ *event handler* Routinen aus Anwendung dem *toolkit* bekannt machen

↪ *call back Routine*

Ereignisgesteuerte Programmierung
Erstellung von vielen kleinen Prozeduren, den *event handlern*, die auf gemeinsamen globalen Daten, den *widgets* arbeiten

↪ jede Routine ändert den Zustand des Programms

↪ ganz gewöhnliche imperative Programmierung

das Hauptprogramm befindet sich im *UI–toolkit*

Beispiel

Initialisierung Erzeugen der *widgets* und setzen der *call backs*

```
button := create_button("pressme");
bind_event(button, press, hello)
...
procedure hello
      begin
            writeln("hello world")
      end
```

event loop

start_event_loop

X–Anwendungen

xterm	Terminal Emulation
xeyes	*X is watching you*
ico	Benchmark: Zeichnen
xclock	Uhr
xload	Maschinenauslastung
xbiff	Ist Post da?
xman	man pages anzeigen
xemacs	Entwicklungsumgebung
xmodemap	Tastaturbelegung
xset	Tastatur und Mauskonfiguration
xev	event tracer
xwd	window dump

11 Tk

Tk	Toolkit
↪	eine Tcl–Anwendung neue Kommandos zur einfachen X–Programmierung
↪	wieder nur eine *library*
↪	wieder um eigene Routinen erweiterbar
widgets	eine Kollektion von *widgets* an Motif angepaßt
widget–Klassen	
label	einzeilige Texte, *bitmaps*, Bilder
message	mehrzeilige Texte
button	normal, *radio, check, menue*, Knöpfe
listbox	Listen
text	Textfenster
canvas	Zeichenfenster
scrollbar	für *entry, listbox, text, canvas*
scale	Skalen
frame	Rahmen zum Gruppieren von *widgets*
toplevel	neues X–Fenster
↪	*widget*–Objekte bilden einen Hierarchie–Baum Wurzel: .

wish	*window shell* eine shell mit allen eingebauten Tk–Kommandos
Kommandos	für *widgets* (Auswahl)
pack	ordnet ein *widget* in einem Fenster an macht das *widget* sichtbar
configure	veränderte *widget*–Attribute
bind	bindet ein Tcl–Kommando an ein X–*event* definiert *call back*–Funktion
destroy	zerstört *widgets*
focus	
grab	setzt *input focus* Zuordnung: Eingabe zu *widget*
wm	*window manager*–Kommunikation

12 Client–Server und Prozess–Kommunikation

einfache Client Server Prozesse
simuliert mit named pipes
der server process: `server`

```
#!/bin/sh                             # @(#)server 2.4 93/01/03
# a simple Bourne shell server
# call $0 <message pipe> <answer pipe> &
# every line in the <message pipe> is interpreted
# as a command, the command output is send back
# through the <answer pipe>, trace is written to c-s.out

usage() {
  echo "Usage: $0 <message pipe> <answer pipe>" 1>&2
  exit 1
}

[ $# -eq 2   ] || usage
Input=$1;          Output=$2
[ -p $Input  ] || usage
[ -p $Output ] || usage

Message=""
Tmpfile=/tmp/tmp-$$.1
Trace=c-s.out

trace() {
   echo `date | cut -d´ ´ -f4` $$ $0" "\"$*\" >> $Trace; }
```

der eigentliche server

```
receiveMessage() {
  read Message < $Input
  trace receiveMessage $Message
}

processMessage() {
 $Message > $Tmpfile
}

sendAnswer() {
 trace sendAnswer `cat $Tmpfile`
 echo `cat $Tmpfile` >> $Output
}

trap "trace terminated; rm -f $Tmpfile; exit" 0 1 2 15

trace started

while true
do
   receiveMessage
   processMessage
   sendAnswer
done
```

ein Aufruf

```
$ /etc/mknod inputpipe p
$ /etc/mknod outputpipe p
$ server inputpipe outputpipe &
$ echo date >> inputpipe
$ cat outputpipe
Sun Jan 03 13:57:57 MET 1993
```


ein einfacher client: `client`

```
#!/bin/sh                                # @(#)client 2.5 93/01/03
# a simple cient, that executes a command a specified
# number of times by sending this command to a server
# a message is a single line command

usage() {
  echo "Usage: $0 <command> <times> <delay>  \
                <server input> <server output>" 1>&2
  exit 1
}

[ $# -eq 5 ]          || usage
Command=$1;              Times=$2;             Delay=$3;
ServerInput=$4;          ServerOutput=$5
[ -p $ServerInput  ] || usage
[ -p $ServerOutput ] || usage

Message=""
Trace=c-s.out

trace() {
   echo `date | cut -d´ ´ -f4` $$ $0" "\"$*\" >> $Trace; }
```

der eigentliche client

```
generateMessage() {
     Message=$Command
}

sendMessage() {
                                   trace sendMessage $Message
     echo $Message >> $ServerInput
}

receiveAnswer() {
     read Answer < $ServerOutput
                                  trace receiveAnswer $Answer
}

processAnswer() {
     echo $Answer
}

                                         trace client started
while [ $Times -ne 0 ]
do
     generateMessage
     sendMessage
     receiveAnswer
     processAnswer
     sleep $Delay ; Times=`expr $Times - 1`
done
                                      trace client terminated
```


das Starten eines servers: `startserver`

```
#!/bin/sh                         # @(#)startserver 2.1 93/01/03
# this program creates two pipes for communication and
# starts a server process# with these pipes for message
# passing, the names of the pipes are written to stdout

usage() {
  echo "Usage: $0 <server>" 1>&2
  exit 1
}
[ $# -eq 1 ] || usage ; Server=$1

Tmpfile=/tmp/$$
Input=$Tmpfile.in;              Output=$Tmpfile.out

Trace=c-s.out
trace() {
   echo `date | cut -d´ ´ -f4` $$ $0" "\"$*\" >> $Trace; }

/etc/mknod $Input p
/etc/mknod $Output p
                       trace pipes created: $Input $Output
$Server $Input $Output &
                             trace server started: $Server
echo $Input $Output
```

das Stoppen eines servers: `stopserver`

```
#!/bin/sh                         # @(#)stopserver 2.1 93/01/03
# this program stops a server process and
# removes the communication pipes

usage() {
  echo "Usage: $0 <server input> <server output>" 1>&2
  exit 1
}
[ $# -eq 2 ] || usage
ServerInput=$1;              ServerOutput=$2

Trace=c-s.out
trace() {
   echo `date | cut -d´ ´ -f4` $$ $0" "\"$*\" >> $Trace; }

                                  trace send exit command
echo exit >> $ServerInput

            trace remove pipes: $ServerInput $ServerOutput
rm $ServerInput $ServerOutput
```


ein Initialisierungsprogramm: `startappl`

```
#!/bin/sh                          # @(#)startappl 2.2 93/01/03
usage(){
  echo "Usage: $0 <client> <server> <client params>" 1>& 2
  exit 1
}
[ $# -eq 3 ] || usage

Client=$1;              Server=$2;                 Params=$3
Tmp=/tmp/$$.pipes

Trace=c-s.out;
trace() {
   echo `date | cut -d´ ´ -f4` $$ $0" "\"$*\" >> $Trace; }

                            trace call startserver $Server
startserver $Server > $Tmp
Pipes=`cat $Tmp`; rm $Tmp
                 trace start client: $Client $Params $Pipes
$Client $Params $Pipes
                              trace call stopserver $Pipes
stopserver $Pipes
```

ein Testlauf
`startappl client server ´date 2 1´ &`

```
time     pid  program      action

13:57:55 1082 startappl   "call startserver server"
13:57:56 1086 startserver "pipes created: /tmp/1086.in..."
13:57:56 1086 startserver "server started: server"
13:57:56 1091 server      "started"
13:57:56 1082 startappl   "start client: client date 2..."
13:57:57 1101 client      "client started"
13:57:57 1101 client      "sendMessage date"
13:57:57 1091 server      "receiveMessage date"
13:57:57 1091 server      "sendAnswer Sun Jan 03 13:57..."
13:57:57 1101 client      "receiveAnswer Sun Jan 03 13..."
13:57:59 1101 client      "sendMessage date"
13:57:59 1091 server      "receiveMessage date"
13:57:59 1091 server      "sendAnswer Sun Jan 03 13:57..."
13:57:59 1101 client      "receiveAnswer Sun Jan 03 13..."
13:58:01 1101 client      "client terminated"
13:58:01 1082 startappl   "call stopserver /tmp/1086.i..."
13:58:01 1136 stopserver  "send exit command"
13:58:01 1136 stopserver  "remove pipes: /tmp/1086.inp..."
13:58:01 1091 server      "receiveMessage exit"
13:58:02 1091 server      "terminated"
```


Erweiterung: `server`
ein server für mehrere cients (multithreaded)

```
#!/bin/sh                              # @(#)server 3.2 93/01/03
# a simple multithreaded Bourne shell server
usage() {
   echo "Usage: $0 <message pipe>" 1>&2
   exit 1
}
[ $# -eq 1  ] || usage; Input=$1; [ -p $Input  ] || usage
...
receiveMessage() {
   read Line < $Input
   Output=`echo $Line | cut -d: -f1`
   Message=`echo $Line | cut -d: -f2`
}
processMessage() {
   $Message > $Tmpfile
}
sendAnswer() {
   echo `cat $Tmpfile` >> $Output
}
...
while true
do
   receiveMessage
   processMessage
   sendAnswer
done
```

Starten: `startserver`

```
#!/bin/sh                          # @(#)startserver 3.2 93/01/03
...
[ $# -eq 1 ] || usage;
Server=$1;                                   Input=/tmp/$Server.in

if [ -p $Input ]
then
                         trace server $Server already running
else
   /etc/mknod $Input p
                                    trace pipe created: $Input
   $Server $Input &
                                 trace server started: $Server
fi
```

stoppen: `stopserver`

```
if [ -p $Input ]
then                                       trace send exit command
   echo exit >> $Input
                                    trace remove pipes: $Input
   rm $Input
else
                                     trace $Server not running
fi
```


ein Aufruf

```
startserver server
client date 2 20 &
 client who 2 10 &
 stopserver server
```

```
16:58:01 2192 startserver "pipe created: /tmp/server.in"
16:58:01 2192 startserver "server started: server"
16:58:01 2196 server      "started"
16:58:22 2203 startserver "server server already running"
16:58:22 2202 client      "client started"
16:58:23 2202 client      "sendMessage /tmp/2202.out:date"
16:58:23 2196 server      "receiveMessage /tmp/2202.out:..."
16:58:23 2196 server      "sendAnswer to /tmp/2202.out S..."
16:58:23 2202 client      "receiveAnswer Sun Jan 03 16:5..."
16:58:38 2227 startserver "server server already running"
16:58:38 2226 client      "client started"
16:58:38 2226 client      "sendMessage /tmp/2226.out:who"
16:58:38 2196 server      "receiveMessage /tmp/2226.out:who"
16:58:39 2196 server      "sendAnswer to /tmp/2226.out u..."
16:58:39 2226 client      "receiveAnswer uwe vt01 Jan 3 ..."
16:58:44 2202 client      "sendMessage /tmp/2202.out:date"
...
16:58:49 2226 client      "sendMessage /tmp/2226.out:who"
...
16:58:50 2226 client      "receiveAnswer uwe vt01 Jan 3 ..."
16:59:00 2226 client      "remove pipe /tmp/2226.out"
16:59:00 2226 client      "client terminated"
16:59:05 2202 client      "remove pipe /tmp/2202.out"
16:59:05 2202 client      "client terminated"
16:59:18 2301 stopserver  "send exit command"
16:59:18 2301 stopserver  "remove pipes: /tmp/server.in"
16:59:18 2196 server      "receiveMessage exit"
16:59:19 2196 server      "terminated"
```

13 make

Mini–Projekt

```
$ cd project

$ cat main.c
#include "glob.h"
#include "m1.h"

main(){printf("f1(5) = %u\n",f1(5));}

$ cat glob.h
typedef unsigned long NO;

$ cat m1.h
extern NO f1(NO x);

$ cat m1.c
#include "glob.h"
#include "m2.h"
#include "m1.h"

NO f1(NO x){return x+f2(x+1);}

$ cat m2.h
extern NO f2(NO x);

$ cat m2.c
#include "glob.h"
#include "m2.h"

NO f2(NO x){return x*2;}
```

make prog

```
$ make
cc -c main.c
cc -c m1.c
cc -c m2.c
cc -o prog main.o m1.o m2.o
```

2. make prog

```
$ make
`prog´ is up to date.
```

main.c geändert: 3. make prog

```
$ touch main.c
$ make
cc -c main.c
cc -o prog main.o m1.o m2.o
```

m1.h geändert: 4. make

```
$ touch m1.h
$ make
cc -c main.c
cc -c m1.c
cc -o prog main.o m1.o m2.o
```


makefile 2.Version

```
$ cat make.2
CCC = $(CC) $(CFLAGS) -c $*.c
OBJ = main.o m1.o m2.o

prog   : $(OBJ)           ; cc -o $@ $(OBJ)
main.o : main.c           ; $(CCC)
m1.o   : m1.c m1.h m2.h ; $(CCC)
m2.o   : m2.c m2.h        ; $(CCC)

$(OBJ) : glob.h
```

makefile 3.Version: mehrere Ziele

```
$ cat make.3
# make.2 erweitert um Aufraeumen und Trace Version
TRC =
CCC = $(CC) $(CFLAGS) $(TRC) -c $*.c
OBJ = main.o m1.o m2.o

prog   : $(OBJ)          ; cc -o $@ $(OBJ)
main.o : main.c          ; $(CCC)
m1.o   : m1.c m1.h m2.h  ; $(CCC)
m2.o   : m2.c m2.h       ; $(CCC)

$(OBJ) : glob.h

CLEANUP: ; rm -f main.o m1.o m2.o

TRACE  : ; make TRC=´-DTRACE=1´
```

Aufräumen

```
$make CLEANUP
```

Trace Version erzeugen

```
$make TRACE
```


makefile 4.Version: selbstkommentierend

```
$ cat make.4
# make.3 erweitert um Kommentarausgabe
## Dies ist ein makefile fuer das Beispielprogramm prog
## ex gibt die folgenden Ziele
##
TRC =
CCC = $(CC) $(CFLAGS) $(TRC) -c $*.c
OBJ = main.o m1.o m2.o

COMMENT : ; grep ´^##´ makefile

##    prog    : Erzeugung des Programms prog
##
prog   : $(OBJ)          ; cc -o $@ $(OBJ)
main.o : main.c          ; $(CCC)
m1.o   : m1.c m1.h m2.h  ; $(CCC)
m2.o   : m2.c m2.h       ; $(CCC)

$(OBJ) : glob.h

##    CLEANUP : Loeschen aller Zwischendateien
##
CLEANUP: ; rm -f main.o m1.o m2.o

##    TRACE   : Erzeugung von prog mit TRACE flag
##
TRACE  : ; make TRC=´-DTRACE=1´ prog
```

Selbstkommentierender makefile:

```
$make
## Dies ist ein makefile fuer das Beispielprogramm prog
## ex gibt die folgenden Ziele
##
##    prog    : Erzeugung des Programms prog
##
##    CLEANUP : Loeschen aller Zwischendateien
##
##    TRACE   : Erzeugung von prog mit TRACE flag
##
```

14 SCCS

Archiv kreieren

```
$ admin -iecho3 s.echo3
No id keywords (cm7)
$ rm echo3
$ ls -l s.*
-r--r--r--   1 uwe           298 Sep 17 10:05 s.echo3
```

Projekt–Archiv

```
$ cd project; ls *.[ch]
glob.h  m1.c  m1.h  m2.c  m2.h  main.c

$ mkdir archiv
$ for i in *.[ch]
> do
> admin -i$i archiv/s.$i
> rm $i
> done
No id keywords (cm7)
..........
```

aus Archiv auslesen

```
$ get s.echo3
1.1
7 lines
No id keywords (cm7)
```

aus Archiv–dirctory auslesen

```
$ cd project
$ get archiv

archiv/s.glob.h:
1.1
1 lines
No id keywords (cm7)
........

$ ls -l *.[ch]
-r--r--r--   1 uwe            26 Sep 17 10:09 glob.h
-r--r--r--   1 uwe            82 Sep 17 10:09 m1.c
..........
```

id keywords

```
$ cat demo.c
static char ident[] = "%Z% %M% Version %I% (%D%)";
main() {}

$ admin -idemo.c s.demo.c
$ get s.demo.c
1.1
2 lines

$ cat demo.c
static char ident[]="@(#) demo.c Version 1.1 (90/09/17)";
main() {}

$ what demo.c demo
demo.c:
    demo.c Version 1.1 (90/09/17)
demo:
    demo.c Version 1.1 (90/09/17)
```


zum Editieren auslesen

```
$ get -e s.echo3
1.1
new delta 1.2
7 lines

$ cat echo3
#!/bin/sh
# echo3 - %Z% Version %I% (%D%)
.......
```

ins Archiv zurückschreiben

```
$ delta -y"shift verwendet,id keywords eingefuegt" s.echo3
1.2
3 inserted
3 deleted
4 unchanged
```

neues Release

```
$ get -e -r2 s.echo3
1.2
new delta 2.1
7 lines

$ delta -y"beliebig viele Argumente zugelassen" s.echo3
2.1
8 inserted
5 deleted
2 unchanged
```

branch kreieren

```
$ admin -fb s.echo3

$ get -e -r1.2 -b s.echo3
1.2
new delta 1.2.1.1
7 lines

$ delta -y"if Anweisungen eingefuegt" s.echo3
1.2.1.1
13 inserted
3 deleted
4 unchanged
```

SCCS Fehlermeldungen

```
$ help cm7

cm7:
"No id keywords"
No SCCS identification keywords were substituted for.
.........

This isn´t an error, only a warning.
```


PRint Sccs versions

```
$ prs s.echo3
s.echo3:

D 1.2.1.1 90/09/17 10:39:04 uwe 4 2 00013/00003/00004
MRs:
COMMENTS:
if Anweisungen eingefuegt

D 2.1 90/09/17 10:35:19 uwe 3 2 00008/00005/00002
MRs:
COMMENTS:
beliebig viele Argumente zugelassen

D 1.2 90/09/17 10:32:33 uwe 2 1 00003/00003/00004
MRs:
COMMENTS:
shift verwendet, id keywords eingefuegt

D 1.1 90/09/17 10:05:29 uwe 1 0 00007/00000/00000
MRs:
COMMENTS:
date and time created 90/09/17 10:05:29 by uwe
```

UNGET: Delta löschen

```
$ get -e s.echo3
2.1
new delta 2.2
10 lines

$ unget s.echo3
2.2
```

Versionen vergleichen

```
$ sccsdiff -r1.1 -r1.2 s.echo3
1c1
< : echo3 - 1. Version
---
> : echo3 - %Z% Version %I% (%D%)
4a5,6
> echo "   $1" ;shift
> echo "   $1" ;shift
6,7d7
< echo "   $2"
< echo "   $3"
```

SCCS und make

```
$ cd project; ls -l
total 1
drwxrwxrwx   1 uwe          1024 Sep 17 11:36 archiv
```

makefile aus Archiv lesen

```
$ get archiv/s.makefile
1.4
16 lines
```


make mit Regeln für lesen aus Archiv

```
$ make
get  archiv/s.m1.h
get  archiv/s.glob.h
get  archiv/s.main.c
/bin/cc -O -c main.c
get  archiv/s.m2.h
get  archiv/s.m1.c
/bin/cc -O -c m1.c
get  archiv/s.m2.c
/bin/cc -O -c m2.c
cc -o prog main.o m1.o m2.o
```

Testaufruf

```
$ prog
f1(5) = 17
```

id keywords in C–files

```
$ cat main.c
#include "glob.h"
#include "m1.h"
static char mainid[]="@(#) main.c Version 1.3 (90/09/17)";
main(){printf("f1(5) = %u\n",f1(5));}
```

WHAT modules in program

```
$ what prog
prog:
   main.c Version 1.3 (90/09/17)
   m1.c Version 1.3 (90/09/17)
   m2.c Version 1.3 (90/09/17)
```