Unix Aufbau und Shellprogrammierung mit LINUX

Dipl.-Ing. Christian Prager

Inhalt

Grundlagen der csh Programmierung

Metazeichen

- Für
  - Ein-/Ausgabeumleitung
  - Dateinamenergänzung
  - Prozeßsteuerung
  - Variablensubstitution
  - Historymechanismus
  - Kommentare
- Eigenschaften
  - von der Shell verarbeitet
  - für ausgeführte Befehle unsichtbar

Metazeichen

Meta	Bedeutung
>	stdout in Datei umleiten (stdout = Standard-Output -> Bildschirm)
>>	hängt stdout an Datei
<	stdin von Datei lesen (stdin = Standard-Input -> Tastatur)
<<	here document, liest von stdin bis angegebenes Wort auftritt
>&	stderr und stdout in Datei umleiten (stderr = Standard-Error -> Bildschirm)
>>&	stderr und stdout an Datei anhängen
|	stdout in stdin von Befehl leiten (Pipe)
|&	stdout und stderr in stdin von Befehl leiten (Pipe)
*	kein oder mehrere Zeichen
?	ein beliebiges Zeichen
[ ]	ein Zeichen aus der angegebenen Menge
{p1,p2,...}	deckt jeden String ab, der durch Pattern p1, p2 oder ... angegeben ist
1>&2	Standard-Ausgabe in Fehlerkanal leiten

Metazeichen

Meta	Bedeutung
c1;c2	Trennzeichen zwischen Befehl c1 und c2, sequentielle Ausführung
c&	Befehl c wird im Hintergrund ausgeführt
`c`	ersetze Ausdruck durch den Output von Befehl c
(c)	führe c in einer Sub-shell aus
c1&&c2	führe c2 nur aus, wenn c1 erfolgreich ausgeführt wurde
c1||c2	führe c2 nur aus, wenn c1 nicht erfolgreich ausgeführt wurde
!	history Substitution - Einleiten
^	history Substitution - Ersetzung
:	leitet Modifikator ein (von Umgebungsvariablen)
$0..$9	Kommandozeilenparameter eines csh-Skripts
$var	Wert der Variablen var
${var}text	Wert der Variablen var mit angehängtem text

Metazeichen

Meta	Bedeutung
\c	Sonderbedeutung von Metazeichen c aufheben
".."	Sonderbedeutung aller Metaz. außer $, ! und ` ` wird aufgehoben
'..'	Sonderbedeutung aller Metazeichen außer ! wird aufgehoben
${10},...	Kommandozeilenparameter 10, 11,... eines csh-Skripts
~	~username wird durch dessen $HOME Pfad ersetzt
#	Kommentar, nur in Shell-Skripts
#!/bin/csh	leitet in 1. Zeile ein csh-Skript ein
#!/bin/tcsh	leitet in 1. Zeile ein tcsh-Skript ein
#!/bin/sh	leitet in 1. Zeile ein Bourne-Shell-Skript ein

Kommandos und exit-Werte

- **jeder Ausgeführte Befehl liefert in Variable** $status **seinen exit-Wert**
  - **0** **Ausführung erfolgreich**
  - **≠ 0** **Ausführung nicht erfolgreich, ist Wert <128 wurde Programm normal beendet, bei Wert >128 abnormale (kill, ...)**

```
% cd /abc
 /abc: No such file or directory.
% echo $status
 1
% echo $status
 0
% cd
% echo $status
 0
% cd csh
% echo $status
 0
```

Pipelines und Kommandolisten

Listen

- **gebildet aus Pipelines**
- **Pipelines durch Metazeichen getrennt**

- **Metazeichen:**
  - **;** sequentielle Ausführung der Befehle
  - **&** zuvor angegebene Pipeline wird im Hintergrund ausgeführt
  - **&&** nachfolgende Pipeline wird nur bei Erfolg der vorangegangenen ausgeführt
  - **||** nachfolgende Pipeline wird nur bei Fehler ausgeführt

Kommandolisten

- **Output einer Kommandoliste wie Output von Einzelkommando behandeln**

```
% ls /bin/e*
/bin/e2fsck   /bin/efdisk   /bin/env   /bin/expand   /bin/expr
% ls /bin/f*
/bin/faillog   /bin/fdisk   /bin/free   /bin/false /bin/fold
/bin/fsck
% ls /bin/e* ; ls /bin/f*
/bin/e2fsck   /bin/efdisk   /bin/env    /bin/expand   /bin/expr
/bin/faillog   /bin/fdisk   /bin/free /bin/false    /bin/fold
/bin/fsck
% ls /bin/e* ; ls /bin/f* | wc -w
/bin/e2fsck   /bin/efdisk   /bin/env   /bin/expand   /bin/expr
6
% (ls /bin/e* ; ls /bin/f*) | wc -l
11
```

Kommentare

- durch # eingeleitet
- Erweiterung
  - erste Zeile von Skript erlaubt Shell oder anderes ausführende Programm zu definieren:

```
#!/bin/csh
#!/bin/tcsh
#!/bin/sh
#!/bin/perl
```

- Bei Kommandoeingabe ist # als Kommentar nicht erlaubt!

Kommandosubstitution

- `cmd` wird durch Output von cmd ersetzt

```
% ls -C
text   typ
% wc `ls -C`
37       92      659          text
 1        2       12          typ
38       94      671          total

% echo `pwd`-Directory
/home/u2/csh-Directory
```

Shell Skripts - Aufruf

- **mit vi oder anderem Editor erstellen**
- 2 Aufrufvarianten

- **Variante 1:**

  % csh Skriptname

- **Variante 2:**

  % chmod u+x Skriptname

  % Skriptname

- **Wird bei Variante 2 Skript nicht gefunden: "**./Skriptname**" oder falls "." im Pfad muss mit "**rehash**" Pfad-Hash-Tabelle in csh neu aufgebaut werden.**

Shell - Parameter

- **csh kennt 2 Parameterarten**
  - **Positionsparameter**
  - **Shell-Variablen**
- **Positionsparameter**
  - **geben Kommandozeilenargumente eines csh-Skripts an**
  - **mit $1..$9, ${10}, ${11}, ... bezeichnet**
  - **$0 ist Name des Skripts**
  - **$argv[1],$argv[2],... können anstatt von $1,$2,... verwendet werden**
- **$#**argv **liefert immer die Anzahl der angegebene Positionsparameter.**

Positionsparameter

% cat para

```
#!/bin/csh
echo Erstes Argument ist $1
echo Erstes Argument ist $2
echo Der Skriptname ist $0
```

% chmod u+x para

% ./para susi edi

```
Erstes Argument ist susi
Erstes Argument ist edi
Der Skriptname ist para
```

% cat para

```
#!/bin/csh
echo Erstes Argument ist $argv[1]
echo Erstes Argument ist $argv[2]
echo Anzahl der Parameter $#argv
```

% para susi edi

```
Erstes Argument ist susi
Erstes Argument ist edi
Anzahl der Parameter 2
```

Shell-Variablen

- können beliebige Namen haben (gleiche Regeln wie für Dateinamen)
- Werte der Variablen werden als Zeichenketten gespeichert
- csh erkennt selbst, wann Variableninhalt als numerischer Wert interpretiert werden soll
- Groß-/Kleinschreibung wird unterschieden

- Setzen von Variablen
  - lokale Variable (nur in aktueller csh bekannt, nicht vererbt)
    set variable [= wert] { variable [= wert] }
  - Umgebungsvariable (werden an Subshells vererbt)
    setenv variable [wert]
  - numerische Ausdrücke
    @ variable = ausdruck { variable = ausdruck }
    (Die Variable muß vorher gesetzt werden)

- Bei Zuweisungen verlieren Metazeichen ihre Bedeutung nicht!

Shell-Variablen

- **Überprüfung ob Variable definiert ist**

  $?Variablenname

  0 wenn Variable nicht definiert

  1 sonst

- **Beispiel**

```
% echo $?hugo
0
% echo $hugo
hugo: Undefined variable.
% set hugo=wifi
% echo $?training
1
```

Shell-Variablen - Vektoren

- csh erlaubt Definition von 1-dimensionalen Arrays
- Arraylänge muß bei Definition festgelegt werden
- Definition

  set arrayname = (wert1 wert2 wert3 ...)
- Zugriff auf i.tes Element

  $arrayname[i]
- Zugriff auf mehrere Elemente

  $arrayname[m - n]

  $arrayname[-m]

  $arrayname[m-]
- Wertzuweisung

  set arrayname[i] = wert
- Anzahl der Elemente im Array

  $#arrayname

Shell-Variablen - Vektoren

```
% echo $?zahl
    0
% set zahl=(eins zwei drei vier)
% echo $?zahl
    1
% echo $zahl[3]
    drei
% set zahl[3]=sieben
% echo $zahl[2-4]
    zwei sieben vier
% echo $zahl[5]
    zahl: Subscript out of range.
% echo $#zahl
    4
```

Anzahl von

zahl

eins	zwei	drei	vier

Shell-Variablen - Vektoren

- **Zugriff auf alle Array-Elemente mit**

  $arrayname

  $arrayname[*]

% echo $zahl[*]

```
eins zwei sieben vier
```

% echo $zahl

```
eins zwei sieben vier
```

% set zahlx="eins zwei drei vier"

% echo $#zahlx

```
1
```

% echo $zahlx

```
eins zwei drei vier
```

% echo $zahlx[1]

```
eins zwei drei vier
```

Variablen und Vektoren löschen

- **lokale Variablen**
  unset variable
- **Umgebungsvariablen**
  unsetenv variable

```
% unset zahl zahla
% echo $?zahl
    0
% echo $zahl
    zahl: Undefined variable.
```

Expandierung bei Zuweisungen

- **bei Variablenzuweisungen in csh findet eine Dateinamenexpandierung statt**

```
% ls /bin/c*
/bin/cat  /bin/chgrp  /bin/chsh  /bin/compress  ...
% set anfang = /bin/c*
% echo $anfang
/bin/cat  /bin/chage  /bin/chfn  /bin/chgrp  /bin/chmod  /bin/chown...
% echo $#anfang
14
% set anfang = '/bin/c*'
% echo $#anfang
1
% echo $anfang
/bin/cat  /bin/chage  /bin/chfn  /bin/chgrp  /bin/chmod  /bin/chown...
% echo "$anfang"
/bin/c*
% echo ´$anfang´
$anfang
```

Vordefinierte Variablen

- **Ausgabe mit**
  set **(lokale Variable)**
  setenv , printenv **(Umgebungsvariable)**

% set

```
argv     ()
cwd      /home/u2/csh
edit     vi
history  50
home     /
mail     /usr/spool/mail/root
notify
path     (/bin /usr/bin /usr/local/bin /usr/bin/X11 /etc /usr/openwin/bin .)
prompt   %
shell    /bin/tcsh
term     xterm
tty      ttyp1
```

Vordefinierte Variablen

- **automatische Variablen**
  werden von csh bei jedem Befehl gesetzt
  Abfrage wie bei anderen Variablen (durch $var)

$	Prozeßnummer der laufende csh
*	alle Positionsparameter als ein String
argv	Positionsparameter ($argv[0] nicht definiert!)
cwd	aktuelles Unterverzeichnis
status	Exit-Status des letzten Befehls

- **Unterschied zwieschen $1 und $argv[1]: falls $1 nicht existiert (kein Kommandozeilenparameter) wird ein leerer String zurückgeliefert, $argv[1] liefert einen Fehler!**

Vordefinierte Variablen

- **Überschreiben von Datei verhindern: set noclobber**

```
% set noclobber
set noclobber
% cat >text1
edi
mimi
CTRL-D                          ( = End Off File)
% cat >text1
text1: File exists.
```

Vordefinierte Variablen

% cat >>text1
```
cat
Convex
CTRL-D
```
% cat text1
```
cat text1
Hallo
Convex
```
% cat >! text1
```
cat
Computer
CTRL-D
```
% cat text1
```
cat text1
Computer
```
% rm text1
% cat >> text1
```
text1: No such file or directory.
```

Eingabe von stdin - csh Skripts

- **Durch "Variable" $< mit**
  set variable = $<
- **Beispiel:**

% vi eingabe

```
#!/bin/csh
echo -n "Dein Name:"
set name = $<
echo
echo "Deine Eingabe ist --- $name ---"
echo "Du hast $#name Woerter eingegeben"
echo "Das 2. Wort ist: $name[2]"
```

% chmod +x eingabe

% ./eingabe

```
Dein Name: Susi Wong
Deine Eingabe ist --- Susi Wong ---
Du hast 1 Woerter eingegeben
Subscript out of range
(getrennte Wörter: siehe Modifikatoren)
```

Modifikatoren

- erlauben die Auswahl von Teilen von Variableninhalten
- werden hinter Ausdrücken mit ":" angehängt ($var:Modifiaktor)

Modifikator	Bedeutung
:h	liefert den Pfadnamen des entsprechende Worts (head)
:t	liefert den Dateinamen des entsprechenden Worts (tail)
:r	schneidet vom Wort eine vorhandene Endung .xxx ab (remove)
:e	liefert vorhandene Endunge .xxx eines Worts (extension)
:q	setzt gesamtes Wort unter Quotes (keine weiteren Substitutionen oder Expandierunge werden ausgeführt)
:x	wie :q nur wird Wort in Einzelworte, die durch Blanks, Tabs oder Zeilenvorschübe getrennt sind, zerteilt

- Jeder Modifikator wirkt nur einmal pro Ausdruck. Soll er auf den ganzen Ausdruck (oder ein Array) wirken, muß ein "g" vorangestellt werden (z.B. :gh).

Modifikatoren

- **Beispiel:**

```
% set a = `which vi`
% echo $a
/usr/bin/vi
% echo $a:t
vi
% echo $a:h
/usr/bin
% set v = /usr/ucb/vi*
% echo $v
/usr/bin/vi /usr/bin/view
% echo $v:h
/usr/bin /usr/bin/view
% echo $v:gh
/usr/bin /usr/bin
% set a="*"
% echo $a
eingabe param text typ
```

Modifikatoren

```
#!/bin/csh
echo -n “Dein Name”
set name=$<
echo “Dein Name ist --- $name ---”
set namx = ($name:x)
echo “Du hast $#namx  Woerter eingegeben.”
echo “Das 2. Wort ist: $namx[2]”
```

Quoting

- **wird verwendet um Sonderbedeutung der Meta-Zeichen aufzuheben**
- 3 Arten
  - **Voranstellen von **
  - **Klammerung mit '..'**
  - **Klammerung mit ".."**
- **Voranstellen von **
  - **\ vor Metazeichen hebt dessen Bedeutung auf (\\ entspricht daher \)**
  - **\ von Nicht-Metazeichen hat keine Bedeutung**
  - **\ innerhalb einer Kommandosubstitution `..` wirkt nur auf $ ` und **
  - **\ am Zeilenende zeigt eine Fortsetzungszeile an**
- **Klammerung mit '..'**
  - **hebt die Bedeutung aller Metazeichen außer \ und ! Auf**
- **Klammerung mit ".."**
  - **alle Metazeichen außer $ " ` \ ! verlieren ihre Bedeutung**

Here-Dokument

- erlaubt die Eingabe in einen Befehl oder csh-Skript durch ein Steuerwort aus dem Eingabestrom zu beenden
- für das WORT wird keinerlei Substitution durchgeführt
- Eingabe in der Form

```
% Befehl ... <<WORT
        % cat <<ENDE
        ? Das ist ein Auto.
        ? $home ist mein Login Verzeichnis.
        ? Das Auto ist rot.
        ? ENDE
        Das ist ein Auto.
        / ist mein Login Verzeichnis.
        Das Auto ist rot.
```

- Wird WORT gequoted (“ENDE“), haben die Metazeichen im Eingabestrom keine Sonderbedeutung. Sonst erfolgt für die Metazeichen im Eingabestrom eine Parameter- und Kommandosubstitution.

Ausdrücke

- benötigt für
  - built-in-Befehle (if, while, exit, ...)
  - arithmetische Berechnungen mit @
- verwenden
  - Ganzzahlen
  - größtmögliche Darstellungsform (4 oder 8 Byte Integers)
  - gleiche Syntax wie Progammiersprache C
- logische Ausdrücke
  - TRUE entspricht einem Wert = 0
  - FALSE entspricht 0

Ausdrücke

Operator	Bedeutung
+ * - /	Addition, Multiplikation, Subtraktion, Division
~	Einerkomplement
%	Modulofunktion
< >	kleiner, größer
<= >=	kleiner gleich, größer gleich
== !=	gleich, ungleich
=~	Stringvergleich
!~	Stringvergleich, vergleicht auf Ungleichheit, sonst wie =~
<<	bitweiser links-Shift
>>	bitweiser rechts-Shift
&	bitweises UND
|	bitweises ODER
^	bitweises EXOR

Ausdrücke

Operatoren für Überprüfungen im File-System

Ausdruck	liefert TRUE wenn
-d datei	datei ein Directory ist
-e datei	datei existiert
-f datei	datei eine normale Datei ist
-l datei	ein link ist
-o datei	dem Benutzer gehört
-r datei	die datei gelesen werden darf
-w datei	die datei beschrieben werden darf
-x datei	die datei ausgeführt werden darf
-z datei	die datei leer ist

Für den angegebenen Dateinamen wird Kommando- und Dateinamenexpansion durchgeführt.

Ausdrücke

- Ausdrücke können mit nachfolgenden Operatoren zu neuen Ausdrücken verknüpft werden:

Operator	Bedeutung
!ausdr	logischer Negationsoperator
ausdr1 && ausdr2	logisches UND
ausdr1 || ausdr2	logisches ODER
{ kommandoliste }	liefert exit-Status der Kommandoliste
(ausdr)	Klammerung, ermöglicht andere Auswertereihenfolge

- Sämtliche Operatoren müssen durch Leerzeichen von den einzelnen Ausdrücken getrennt werden (ausgenommen &,|,<,>,(,))
- In manchen csh-Versionen werden Zahlen die mit 0 beginnen als Oktalzahlen interpretiert. Die Ausgaben erfolgen aber immer als Dezimalzahlen
- Ein leerer String entspricht der Zahl 0
- Metazeichen haben in Ausdrücken keine Sonderbedeutung (anders in sh)

Ausdrücke

- **Built-In-Kommando @**
  - zur Auswertung von arithmetischen und logischen Ausdrücken
  - zur Variablenzuweisung von Ausrücken
  - erlaubt Verwendung von C-Postfix-Operatoren ++ und --

Kommando	Wirkung
@	Ausgabe aller definierten csh-Variablen
@ variable = ausdruck	Wertzuweisung an Variable
@ array[index] = ausdruck	Wertzuweisung an Arrayelement
@ variable++	erhöhe Variable um 1
@ variable--	vermindere Variable um 1

- **Anstatt = können auch die C-Operatoren +=, -=, *=, /=, %=, ^= verwendet werden.**
- **Bei Verwendung von <,>,&,| müssen die Teilausdrücke mit ( ) geklammert werden.**

Ausdrücke

- **Beispiele:**

```
% grep hans *
% echo $status
1
% @ var = ( { grep hans * } )
% echo $var
0
% grep Du *
eingabe:echo "Du hast $#name Woerter eingegeben"
% echo $status
0
% @ var = ( { grep Du * } ) >& /dev/null
% echo $var
1
% @ a = 5
% @ a++
```

Ausdrücke

```
% echo $a
6
% @ b=$a*20
6*20: No match.
% @ b=$a * 20
% echo $b
120
% @ x[2] = 10
x: Undefined variable.
% set x = (a b c d e f g h i j k l m)
% @ x[2] = 10
% echo $x
a 10 c d e f g h i j k l m
% @ x[2] +=25          oder          x[2] = x[2] + 25
% echo $x[2]
35
% @ x[3] += 10
@: Expression Syntax.
```

Ausdrücke

```
% cat pow2
    echo -n "Geben Sie eine Zahl ein: "
    set zahl = $<
    @ pow = $zahl * $zahl
    echo "Das Quadrat der Zahl ist: $pow"
% chmod +x pow2
% ./pow2
Geben Sie eine Zahl ein: 9
Das Quadrat der Zahl ist: 81
% @ a = ( -f . )
% echo $a
0
% @ a = ( -f pow2 )
% echo $a
1
% @ a = ( -d /bin )
% echo $a
1
```

Kommandoklammerung

- **Werden csh-Befehle in ( ) geklammert, werden diese Befehle in einer Sub-Shell ausgeführt. Dies kann zu unvorhersebaren Seiteneffekten führen.**

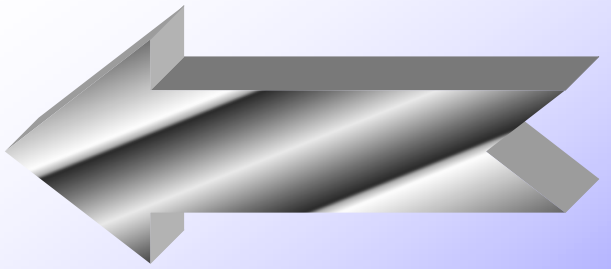
```
% pwd
/home/u2/csh
% (cd /bin; pwd); pwd
/bin
/home/u2/csh
% pwd
/home/u2/csh
```

- **Die Kommandoklammerung kann aber verwendet werden um Stderr und Stdout in verschiedene Dateien umzuleiten**

  Beispiel siehe nächste Seite !

Stdout und Stderr in getrennte Dateien

```
% cat lsax
    #!/bin/csh
    ls p*
    ls x*
% chmod +x lsax
% ./lsax
param  pow2
ls: No match.
% ./lsax > y
ls: No match.
% (lsax >stdout) >& stderr
% cat stdout
param
pow2
% cat stderr
ls: No match.
```



Erweiterte Funktionen der csh Programmierung

Kommandos zur Ablaufsteuerung

- Eigenschaften
  - dienen zum Verändern der Ausführungsreihenfolge von csh-Befehlen
  - sind in csh eingebaut (built-in)
  - dürfen nicht in Pipes oder Kommandolisten enthalten sein (nicht mehr jede csh hat diese Einschränkung)
  - logischen Bedingungen müssen geklammert ( ) werden
  - alle Schlüsselwörter müssen als erste in der Zeile stehen
- Es existieren
  - IF
  - SWITCH
  - WHILE
  - REPEAT
  - FOREACH
  - GOTO

IF - Anweisung

- 3 Formen

1. if (ausdruck) kommando
2. if (ausdruck) then
   kommandos
   endif
3. if (ausdruck) then
   kommandos
   else
   kommandos
   endif

Der if-Teil muß immer in einer Zeile angegeben werden (ausg. \ als Fortsetzungszeichen am Zeilenende)

IF - Anweisung

- **Beispiel:**

% cat wcdat

```
#!/bin/csh
set tmpdatei = "/tmp/wcdat.$$"
if ($#argv == 0) set argv = (.)
(find $argv -print > $tmpdatei) >& /dev/null
echo -n "Der Directorybaum"
echo " $argv"
echo " enthaelt `cat $tmpdatei | wc -l` Dateien"
/bin/rm -f $tmpdatei
```

% ./wcdat /bin

```
Der Directorybaum /bin
enthaelt 90 Dateien
```

% ./wcdat /bin /etc

```
Der Directorybaum /bin /etc
enthaelt 339 Dateien
```

SWITCH - Anweisung

- **erlaubt abhängig vom Wert eines Ausdruckes verschiedene Fälle zu unterscheiden. Die Pattern können die Metazeichen [ ] * ? enthalten.**

```
switch (ausdruck)
  case pattern1:
            kommandos
            breaksw
  case pattern2
            kommandos
            breaksw
  ...
  default:
            kommandos
             breaksw
  endsw
```

- **Die Angabe von** default **ist optional. Wird nach einem** case **kein** breaksw **angegeben, wird, wie in C, mit der Ausführung des nächsten** case **fortgesetzt. Beispiel: Implementierung eines Papierkorbes (vgl. WIN 95)**

SWITCH - Anweisung

```
#!/bin/csh
set trash = $home/.trash
if (! -d $trash) then
  echo "Trashcan $trash not found - making one."
  mkdir $trash
endif
if ($#argv == 0) set argv = ( -l )
switch ($1)
  case -[lL]*:
    echo "Trashcan contains the files:"
    ls -l $trash
    exit 0
  case -[dD]*:
    echo "Trashcan is cleaned up..."
    rm -rf $trash/*
    exit 0
  default:
    mv $argv $trash
    breaksw
endsw
```

SWITCH - Anweisung

Beispiel für Aufruf:

```
% ./trash
Trashcan //.trash not found - making one.
Trashcan contains the files:
total 0
% ./trash stderr
% ./trash stdout
% ./trash [-l]
Trashcan contains the files:
total 2
-rw-r-----   1 root     root       14 Nov 14 20:08 stderr
-rw-r-----   1 root     root       11 Nov 14 20:08 stdout
% ./trash -d
Trashcan is cleaned up...
% ./trash
Trashcan contains the files:
total 0
```

WHILE - Anweisung

while (ausdruck)

kommandos

end

führt kommandos **aus, solange** ausdruck **TRUE liefert**
while **und** end **müssen am Zeilenanfang stehen**
Beispiel:
% cat fibo

```
#!/bin/csh
# Beispiel Fibonacci-Reihe
if ($#argv == 0) then
  echo "Use: fibo <number>"
  exit 1
endif
set x=1 y=1
while ($x <= $1)
  echo $x
  @ z = $x + $y
  set x=$y y=$z
end
```

REPEAT - Anweisung

repeat n kommando

- erlaubt die n-malige Ausführung eines Befehls
- Ein-/Ausgabeumleitungen werden nur einmal vorgenommen (auch bei n=0 !)
- gesamte Anweisung muß in einer Zeile stehen (\ als Fortsetzung!)

```
% repeat 5 date
Sun Nov 14 20:44:05 GMT 1993
Sun Nov 14 20:44:05 GMT 1993
Sun Nov 14 20:44:05 GMT 1993
Sun Nov 14 20:44:05 GMT 1993
Sun Nov 14 20:44:05 GMT 1993
```

FOREACH - Anweisung

foreach variable (wortliste)
    kommandos
end

- **die angegebene Variable wird mit den einzelnen Worten von wortliste belegt**
- **für jede Belegung werden die Kommandos ausgeführt**

Prompting

```
% foreach i (*)
foreach? wc $i
foreach? end
       7       28      162 eingabe
      13       32      153 fibo
      12       27      152 filetest
       4        5       24 lsax
%
```

FOREACH - Anweisung

```
% cat tree
#!/bin/csh
if ($#argv == 0) exit
if ($?indent == 0) setenv indent "   "
set prev = "$indent"
cd $1
echo "$indent $cwd"
setenv indent "$indent   "
foreach f ( `ls` )
  if ( -d $f && ! -l $f) csh -f /home/u2/csh/tree $f
end
setenv indent "$prev"
% tree /etc
   /etc
     /etc/default
     /etc/national
       /etc/national/fonts
     /etc/skel
%
```

GOTO - Anweisung

goto label **erlaubt einen unbedingten Sprung auf ein Label der Form label:**

Beispiel:

% cat zzaehl

```
#!/bin/csh
start:
  set zeile = $<
  if ( $zeile:q == "" ) goto ende
  @ lines++
  echo "($lines) $zeile"
  goto start
ende:
echo "Der Eingabetext enthaelt $lines Zeilen"
```

bash – shell Kontrollstrukturen

(Anmerkung: ksh und bash – shell sind fast identisch!)

IF – Anweisung:

```
if (ausdruck)
then
        kommandos
fi
```

ODER

```
if (Ausdruck)
then
        kommandos
else
        kommandos
fi
```

bash – shell Kontrollstrukturen

WHILE – Anweisung:

while (ausdruck) ← **solange**
do
    kommandos
done

UNTIL – Anweisung:

until (ausdruck) ← **bis**
do
    kommandos
done

FOR – Anweisung:

for variable **in** wortliste
do
    kommandos
done

bash – shell Kontrollstrukturen

CASE – Anweisung:

```
case ausdruck in
muster1)
        kommandos
        ;;
muster2)
        kommandos
        ;;
...
esac
```

TEST – Anweisung:

```
test ausdruck
```

z.B: if test –f dateiname
then .....

bash – shell Kontrollstrukturen

READ – Anweisung:

read –optionen variable1 variable2 **...**

Zeilenweises einlesen

PRINT – Anweisung:

print –optionen text

shell-Ausgabe ähnlich wie echo-Befehl

Beispiele dazu sie Musterverzeichnis-Directory !!

ALIASES

- **erlauben Kurznamen für Kommandoaufrufe und Kommandoteile**
  - **Setzen mit**
    alias aliasname Kommando(teile)
  - **Anzeigen mit**
    alias
  - **Löschen mit**
    unalias aliasname
    unalias *
  - Eintragen in .login - Datei

```
% alias dir 'ls -l'
% alias type cat
% alias h history
% alias
dir     ls -l
h       history
type    cat
```

ALIASES

Beispiele:

% dir

```
total 25
-rwxr-xr-x   1 root     root          162 Nov 14 19:05 eingabe
...
```

% unalias type

% type eingabe

```
type: Command not found.
```

% alias liste dir

% alias dir ls -CF

% set echo

% liste *

```
ls -CF eingabe fibo filetest lsax ok param pow2 textcsh trash tree typ
wcdat zzaehl
eingabe*   filetest*  ok*  pow2*  trash*  typ*
```

% unset echo

% unalias *

% alias

%

Zeitmessung von Prozessen

- time **erlaubt es die Zeit, die ein Kommando zur Ausführung benötigt sowie andere Laufzeitparameter zu ermitteln**
- time **ohne Argumente gibt die bisher in der laufenden csh verbrauchten Ressourcen an**
- time kommando **gibt den Ressourcenverbrauch von Kommando an**

% time

```
0.220u 0.760s 4:59.32 0.3% 0+0k 0+0io 39pf+0w
```

% time find /etc -name hosts

```
/etc/hosts
0.000u 0.110s 0:00.13 84.6% 0+0k 0+0io 9pf+0w
```

Datei-Utilities

- **Kommando zum Suchen von Dateien**

  which <befehl>

  - **which sucht nach dem angegebenen Befehl unter Verwendung des Suchpfades PATH.**
  - **wird der Befehl gefunden, wird sein kompletter Pfadname ausgegeben, sonst eine Fehlermeldung.**
  - **which funktioniert nur bei Verwendung der csh oder tcsh!**
  - **sh: where, whence**

- **Beispiel**

  % which ls

  ```
  /usr/bin/ls
  ```

Kommandos zum Anzeigen von Dateien

- erlauben Anzeige in verschiedenen Formaten
  - ASCII
  - hexadezimal
  - dezimal
- erlauben Steuerung der Anzeige
  - seitenweise
  - Zeilenweise

head [-count] **file**

- head gibt die ersten 10 oder count Zeilen einer Datei aus.

tail

tail [[[+|-]n] [lbc]] [-r] [-f] file

- **tail gibt Ende einer Datei aus**
- **bei positivem n wird der Inhalt der Datei ab der n.ten Zeile ausgegeben**
- **bei negativem n wird n-Zeilen vor dem Dateiende mit der Ausgabe begonnen**
- **n in Zeilen angegeben, andere Einheiten: l (line),b (block), c (character) kein n: -10 default**
- **-r Option erlaubt die Ausgabe der Zeilen der Datei in umgekehrter Reihenfolge (letzte Zeile zuerst).**
- **-f Option: tail wartet ob sich die Datei vergrößert; zeigt gegebenenfalls automatisch die neuen Zeilen an. Abbruch mit CTRL-C.**
- Beispiele

head -5 /etc/motd
tail +5 /etc/motd
tail -50c /etc/motd

more

more [-n] [+zeile] [+/suchbegriff] file ...

- **more zeigt Datei bildschirmweise an**
- **-n erlaubt die Angabe der Ausgabeseitenlänge in Zeilen**
- **+zeile erlaubt Angabe, ab welcher Zeile der Dateiinhalt angezeigt wird**
- **+/-suchbegriff veranlaßt more nach "suchbegriff" in der Datei zu suchen und 2 Zeilen vor dem ersten Auftreten des Begriffs mit der Anzeige zu beginnen**
- **während der Anzeige erscheint am unteren linken Bildschirmrand eine Anzeige, wieviel Prozent der Datei bereits angezeigt wurden.**
- **während der Anzeige können Befehle eingeben werden, die die Ausgabe der Datei steuern**

more

- **Die wichtigsten Befehle in more**

i<space>	zeigt i weitere Zeilen, oder wenn i fehlt eine neue Seite an
Return	zeigt eine weitere Zeile an
=	zeige aktuell Zeilennummer an
v	starte Editor (vi). Kann durch Umgebungsvariable EDITOR gesteuert werden. Nach Verlassen des Editors kehrt man in more zurück.
/<text>	suche nach <text> in der Datei und zeige Stelle an
h	HELP
q	beende more

- **In vielen UNIX-Versionen wurde Funktionalität von more stark erweitert.**
- **Hauptnachteil: kann nicht rückwärts Blättern.**
- **Neuer Befehl (GNU): less**

less

less [-M] [+commands] file ...

- **less zeigt Datei bildschirmweise an**
- **less erlaubt die Angabe von Kommandos in der Kommandozeile**
- **Umgebungsvariable LESS erlaubt Definition von Optionen, die bei jedem less-Aufruf verwendet werden sollen**
- **-M Option Anzeige von Dateinamen, Zeilennummer der Zeile am oberen Bildschirmrand**
- **und die Prozentzahl des bisher angezeigten Dateiinhaltes anzeigt am linken unteren Bildschirmrand**
- **Befehlsumfang gegenüber more stark erweitert**
- **bekannte Befehle aus more weiterhin verwendbar**

less

zusätzliche Befehle (nur auszugsweise!):

b	blättere eine Seite rückwärts
r	Bildschirminhalt neu ausgeben
R	Bildschirminhalt neu ausgeben und zuvor alle Puffer löschen (praktisch, wenn sich Datei ändert)
nG ng	gehe zu Zeile n
G	gehe zum Dateiende
g	gehe zum Dateianfang
n% np	gehe zur Position wo n% der Datei liegen
= ^G	zeige Informationen zur Datei
:n	zeige nächste Datei
:p	zeige vorhergehende Datei

pr

pr [options] [file] ...

- **pr formatiert eine angegebene Datei seitenweise**
- **Seiten können Seitennummer, Überschrift, Datum, Zeit usw. enthalten**
- **standardmäßig Ausgabe von Seitennummer, aktuellem Datum und Zeit**
- **sowie Dateinamen**
- **Seitenlänge standardmäßig 66 Zeilen**

wichtige Optionen:

- **+n** beginne mit der n.ten Seite
- **-n** erzeuge einen n-Spaltigen Ausdruck
- **-h** nimmt das nächste Argument als Seitenüberschrift
- **-wn** stellt die Zeilenlänge auf n Zeichen, default ist 72
- **-f** verwende Seitenvorschubsteuerzeichen (ASCII 12 (=FF)) anstelle von Leerzeilen für den Seitenumbruch
- **-ln** stelle die Seitenlänge auf n Zeilen

pr und Drucken

- **wichtige Optionen (forts.):**
  - **-t** unterdrückt die 5-zeiligen Fuß- und Kopfzeilen
  - **-sc** definiere c als Spaltentrennzeichen (default ist ein Leerzeichen) fehlt c wird ein Tabulator angenommen
  - **-m** gibt alle spezifizierten Dateien gleichzeitig, in getrennten Spalten aus

- **Beispiele:**

```
pr text | more
pr -h "Das ist eine Ueberschrift" text | more
pr -2 text | more
```

- **Drucken allgemein:**

lp dateiname **oder**
lpr dateiname

sort

sort [-bdfinrcmu] [-tx] [+pos1 [-pos2]] [-o outfile] [file] ...

- **sotiert eine oder mehrere Dateien zeilenweise und gibt die sortierten Zeilen auf stdout aus**
- **-o outfile erlaubt Angabe einer Ausgabedatei (Ausgabedatei darf den gleichen Namen wie eine Eingabedatei haben)**
- **standardmäßig wird für die Sortierung die gesamte eingelesene Zeile verwendet**
- **die Sortierreihenfolge ist lexikographisch in der Reihenfolge des Zeichensatzes der Hardware (meist ASCII).**

- **sort erlaubt Angabe von**
  - **Sortierreihenfolge**
  - **Sortierschlüssel**

sort

- Optionen zum Ändern der Sortierreihenfolge
  - b ignoriere führende Leerzeichen und Tabulatoren
  - d Sortiere nur nach Buchstaben, Leerzeichen und Ziffern
  - f konvertiere Großbuchstaben zu Kleinbuchstaben vor dem Sortieren
  - i ignoriere alle Zeichen außerhalb des ASCII Zeichensatzes (040-0176 Oktal)
  - n sortiere nach numerischen Werten (diese können aus Ziffern gefolgt von einem Dezimalpunkt gefolgt von Ziffern bestehen)
  - r kehre die Sortierreihenfolge um
- Optionen zum Ändern des Sortierschlüssels
  - Angabe von +pos1 -pos2 beschränkt Sortierschlüssel auf Feld, das bei pos1 beginnt und vor pos2 endet
  - Felder standardmäßig durch Leerzeichen getrennt
  - -tx definiert x als Feld-Trennzeichen

sort

- **Format der Positionsangaben**
  - pos1 und pos2 haben Form m.n
  - m definiert Feldnummer innherhalb der Zeile
  - n definiert der Zeichen die im Feld übersprungen werden sollen
  - jede Positionsangabe kann von einer der Optionen bdfinr gefolgt werden
  - Sortieroptionen für einzelne Felder überschreiben Kommandozeilenoptionen
  - .n weggelassen: .0 eingesetzt
  - fehlt pos2, wird Zeilenende verwendet
- **Werden mehrere Sortierbegriffe angegeben, so werden nur dann weitere Sortierbegriffe beachtet, wenn alle vorhergehenden Sortierbegriffe einen sortierten Input erkannt haben.**

sort

- **zusätzliche Optionen:**
  - **c** prüfe ob Input schon sortiert. Wenn ja erzeuge keinen Output.
  - **m** vermische bereits sortierte Eingabedateien
  - **u** gib doppelte Zeilen nur einmal aus

- **Beispiele:**

```
sort /etc/motd
sort -r /etc/motd
sort -ur /etc/motd
sort -t: +2n /etc/passwd
```

cmp - compare

cmp [-l] [-x] [-s] file1 file2

- **cmp vergleicht 2 Dateien**
- **gibt gefundene Differenzen aus**
- **- Option liest file1 von stdin**
- **eine Differenz gefunden: kein Output**
- **Differenz gefunden: Ausgabe von Byteposition und Zeilennummer**
  - **-l** gibt alle Differenzen aus (Byteposition dezimal, differierende Bytes oktal)
  - **-x** gibt alle Differenzen aus (Byteposition dezimal, differierende Bytes hexadezimal)
  - **-s** gibt keine Differenzen aus (Exit-Status ist bei Unterschieden 1)

Beispiel:

cmp /etc/passwd /etc/group

```
/etc/passwd /etc/group differ: char 9, line 1
```

Spezialfunktionen

file file1 ...

- **file versucht angegebenen Dateien zu klassifizieren (Text, Binary, ...)**
- **Erkennung wird durch enthaltene Zeichen beeinflußt (8. Bit gesetzt, spezielle Header von a.out, ...)**

Beispiele:

% file /bin/w

```
/bin/w: UNIX executable
```

% file /.cshrc

```
/.cshrc: ascii text
```

Spezialfunktionen

strings [-o] [-number] file ...

- **strings sucht in den angegebenen Dateien nach ASCII-Strings**
- **erlaubt z.B. in Binärfiles nach lesbarem Text zu suchen**
- **-o Option gibt Position des Strings im file aus**
- **normalerweise nur Strings mit Mindestlänge von 4 Zeichen beachtet**
- **-number Option erlaubt Vorgabe von Stringmindestlänge**

Beispiel:

% strings /bin/ls

...

Spezialfunktionen

wc [-clw] [file...]

- **= word count**
- **wc zählt in angegebenen Dateien die Anzahl der Zeichen, Zeilen und Wörter**
- **Wörter sind durch " ", TAB und "\n" getrennte Zeichenketten**
- **-l Ausgabe der Zeilenzahl**
- **-c Ausgabe der Zeichenzahl**
- **-w Ausgabe der Wortzahl**
- **standardmäßig werden alle 3 Werte ausgegeben.**
- **werden mehrere Dateien angegeben, so werden auch Gesamtsummen ausgegeben**

Beispiel:

% wc /etc/motd

```
17       96      745 /etc/motd
```

Befehle zum Übertragen von Dateien

uuencode [file] remotename

- **uuencode wandelt Binärdatei in eine ASCII-Datei um**
- **Output kann z.B. mit mail übertragen werden**
- **fehlt file wird von stdin gelesen**
- **remotename definiert den Namen des Files nach der Dekodierung mit uudecode**
- **Ausgabe erfolgt auf stdout**

uudecode [file]

- **uudeocde dekodiert ein mit uuencode kodiertes file**
- **fehlt file wird von stdin gelesen**
- **dekodiertes File wird unter dem bei uuencode angegebenen remotenamen gespeichert.**

uuencode - uudecode

- **Beispiel:**

```
% cat a
a       b       c       d
% uuencode a remote > test
% cat test
begin 666 remote
/80EB"6,)9`H)"6$)8@H*
`
end

% uudecode test
% cat remote
a       b       c       d

% uuencode a remote | uudecode
```

split - cat

split [-n] file [name]

- **split zerteilt eine angegebene Datei in Dateien mit gleicher Zeilenzahl**
- **-n erlaubt Angabe der Dateilänge in Zeilen (default: 1000 Zeilen)**
- **bei Angabe von name werden Ausgabedateien mit nameaa, nameab, ... erzeugt**
- **fehlt name, werden die Dateien xaa, xab,... benannt**

cat [-n] [-v] file ...

- **cat erlaubt eine oder mehrere Dateien auszugeben oder zusammenzuhängen (concatenate)**
- **Ausgabe erfolgt auf stdout**
- **-n nummeriert Zeilen bei Ausgabe.**
- **-v gibt nicht-druckbaren-Zeichen in der Form ^X und Zeichen mit gesetztem 8.Bit in der Form M-x aus**

split - cat

- **Beispiel:**

```
% split -100 text
% cat x* >a
% cmp a text
```

- **Kombination von uuencode und split erlaubt lange Dateien via e-mail zu verschicken ohne an Limits von mail-Programmen zu stoßen:**

```
% uuencode datei remote | split -1000 datei
% foreach i (datei*)
? mail user@host <$i
? End
```

- **Empfänger muß die einzelnen mails in eine Datei speichern, mit Editor sämtliche Mail-Header löschen und die Datei mit uudecode dekodieren.**

Komprimieren und Expandieren von Files

- gzip [options] file1 ...
  - **Reduziert die Größe von Files (mittels Lempel-Ziv Algorithmus)**
  - **erzeugt Datei mit Suffix .gz**
- **Options:**

  -v **zeigt die prozentuelle Komprimierung an**

  -c **schreibt Output auf stdout**

  (Weitere Options siehe man-Page)

Beispiel:

```
% gzip -v test.rgb
test.rgb:    35% -- replaced with test.rgb.gz
% gzip -c test.rgb > testrgb.gz
```

Komprimieren und Expandieren von Files

- gunzip [options] file1 ...
  - **Mit gzip komprimierte Files werden “entpackt”**
  - **dies müssen Files , die mit .gz, -gz, .z, -z, _z, .Z enden**
- **Options:**
  - **-v** **zeigt die prozentuelle Komprimierung an**
  - **-c** **schreibt Output auf stdout**

    (Weitere Options siehe man-Page)
- **Beispiel:**

```
%gunzip -v  test.rgb.gz
 test.rgb.gz:    35% -- replaced with test.rgb
% gunzip -c testrgb.gz > test.rgb
```

Komprimieren und Expandieren von Files

- gzcat **ist identisch mit** gunzip -c
- compress, uncompress,zcat

Diese haben dieselbe Funktion wie gzip, gunzip und gzcat mit kleinen Abweichungen (zb. Suffix wird .Z), schlechter Algorithmus ABER: Standard - UNIX - Kommandos

Archivieren von Files

- tar key [arg...] [file | -C directory]
  - **tar speichert und extrahiert mehrere Files in/aus einem Archiv**
  - **key steuert den tar-Befehl**
  - **es arbeitet default rekursiv auch auf Unterverzeichnisse**

- **key :**

c	**create Archiv**
x	**extract Archiv**
t	**anzeigen**
r	**Archive hinzufügen**
v	**Anzeigen der Files (verbose Mode)**
f	**nächstes Argument ist Archiv-Name**

(Weitere Options siehe man-Page)

- **Beispiel:**

tar cvf allem.tar ./m*	**Archivieren aller Files m***
tar xvf allem.tar	**Extrahieren aller Files**

Beispiel für Archivieren und Komprimieren

Pack and zip

```
% cd
% tar cvf alles.tar ./*
% gzip -c alles.tar > alles.tar.gz

% tar cvf - ./ | gzip -c >alles.tar.gz
```

Move direktories

```
% cd fromdir;  tar cBf - . | (cd todir && tar xBf -)
```

Move direktories to remote computer

```
% cd fromdir;  tar cvf - . | rsh remote-computer “(cd todir && tar xvf -
    )“
```

Find und grep

Find path arg filename arg

arg:

-name	**filename**
-print	**anzeigen**
-type	**b c d f Dateityp**

Bsp:

find /home –name hugo -print

grep suchbegriff dateiname

Bsp:

grep hugo /etc/passwd

Arbeiten im Netz

File Transfer

- **ftp [options] host**
  - erlaubt Filetransfer von und zu anderen Rechnern
- **Options:**
  - **-v** anzeigen aller Tätigkeiten
  - (Weitere Options siehe man-Page)
- **Beispiel:**

%ftp linux.training.wifi.ooe.at

```
Connected to linux.training.wifi.ooe.at
220 linux FTP sertver ........
Name(linux.chris): Userid
331 Password required for chris
Password: XXXXXXXX
ftp> Befehlseingabe
```

File Transfer

- **Befehle im ftp:**
  - **help**
  - **“Standard” Unix Befehle: dir, cd, mkdir, delete, ...**
  - **Für Filetransfer:**

binary	**Binary-Daten-Transfer**
prompt on | off	**Ein-/Ausschalten von Bestätigung bei multiple-file Kommandos**
get remotefile [localfile]	**hole File**
mget remotefiles	**hole mehrere Files**
put localfile [remotefile]	**schicke File**
mput localfile	**schicke mehrere Files**

  - **ftp beenden:**
    quit

(Weitere Befehle siehe man-Page)

Remote - Shell

- **Voraussetzungen:**
  - Eintrag in /etc/hosts.equiv - File als trusted host
  oder
  - in .rhosts im $HOME des remote-Host

- rsh host [-l username] command

  host **remote Rechner**
  command **alle Unix - Befehle, die remote - Rechner kennt**

  Beispiel:

  chp@linux % rsh spp ls -al
  chp@linux % rsh spp cat remotrefile >> localfile

X - Windows - Grundlagen

Was ist X - Windows ?

- **Client-Server-basierte graphische Anwendungsumgebung**
- **Window-Umgebung wird von eigenständiger Applikation (Window-Manager) verwaltet**
- **X-Window-Server läuft auf Grafikworkstations, PC (eXceed, LanWorkplace), X-Terminal und ist für die Anzeige verantwortlich**
- **X-Windows-Clients laufen auf bel. Rechnern und stellen Output über X-Server dar**
- **Zwischen Client und Server erfolgt Kommunikation über X-Windows-Protokoll (standardisiert, portabel)**
- VORAUSSETZUNG:
  Damit Client auf Resourcen des Servers zugreifen kann (z.B.: Fenster aufmachen) muß am Server für den Client eine Zugriffsberechtigung vergeben werden:

xhost Beispiele

- xhost + rechnername — Zugriff erlauben
  xhost - rechnername — Zugriff wegnehmen
  xhost + — Zugriff ALLEN erlauben
  xhost - — Zugriff ALLER wegnehmen
  xhost — Zugriff anzeigen

Beispiele:

%xhost linux.training.wifi.ooe.at
erlaube Clients von linux den lokalen Server zu verwenden

%xhost
zeige alle freigegebenen Clients an

%xhost +
erlaube allen Rechnern im Netz den Zugriff
(SICHERHEITSLOCH)

%xhost -linux
sperre linux

DISPLAY - Variable setzen

- **WICHTIG:**
  DISPLAY - Variable — **definiert X-Server, auf dem Client-Ausgabe erfolgen soll**

- **Setzen der Display-Variable mit**
  %setenv DISPLAY xservername:0[.0]

xservername	**IP-Adresse oder Hostname des X-Display-Servers**
:0	**Verwende 1. Display am angegebenen Server**
[.0]	**erlaubt die Spezifikation von weiteren Display-eigenschaften**

Terminalemulation für X Windows Systeme

- xterm [-option ...]
  - Terminal-Emulation für X Windows Systeme
- Options:

-help	Hilfe
-name	Fenstername
-132	132 Zeichen/Zeile
-e Befehl	Befehlsausführung
-sl Nummer	Scrolbar-Länge auf Nummer setzen
-iconic	als Icon starten
-bg Farbe	Hintergrundfarbe einstellen

(Weitere Options siehe man-Page)

Beispiel:

```
%xterm -name linux -132 -bg red -e rlogin linux.training.wifi.ooe.at
```

awk

- liest Eingabedatei zeilenweise ein
- prüft jede Zeile ob sie auf angegebene Suchmuster paßt
- Verarbeitet jede Zeile mit angegebenen Befehlen und Funktionen
- Suchmuster sind reguläre Ausdrücke und Vergleichsoperationen
- Befehle: if, while, for print, printf,next,exit
- Funktionen: C-Operatoren, length, exp, log, sqrt, int, substr, index, sprintf, split
- Details siehe Anhang !

- Beispiel:

```
awk ´{print $1>“text1”;print $2>“text2”}´ input.txt
```