

7.2. Die C-Shell

```
=====
```

Metazeichen

```
-----
| - Pipe
* - kein, ein oder mehr Zeichen
? - ein beliebiges Zeichen
[...] - eines der in den Klammern angegebenen Zeichen ([!...] nicht),
Index von Feldern
; - Trennzeichen für Kommandos
& - Kommando in Hintergrund, E/A-Verkettung
'kommando' - Ersetzung durch Standardausgabe
( ) - subshell benutzen ({kommando;} nicht), Initialisierung von Felder
$ - Leitet eine Shellvariable ein
\ - Maskierung von Metazeichen
'...' - Shellinterpretation innerhalb der Apostrophs wird
abgeschaltet
"... " - Shellinterpretation innerhalb der Doppelapostrophen
# - wird ausgeschaltet ausser für '$', '' und '\ '
= - Beginn eines Kommentars
- Wertzuweisung
&& - bedingte Ausführung von Kommandos
|| - bedingte Ausführung von Kommandos
> - E/A-Umlenkung
< - E/A-Umlenkung
neu:
~ - Tilde - Homedirectory
! ^ - History Substitution
: - für History Substitution
```

j-p bell

Seite 1

7.2.Cshell

Aufbau eines Kommandos - 1.Teil

```
-----
```

```
<Kommando> ::= <einfaches Kommando> | ...
```

```
<einfaches Kommando> ::= <Kommandoname> { <Argument> }
```

Folge von Wörtern, die durch Leerzeichen (Tabulatoren) voneinander getrennt sind. Das erste Wort gibt den Programmnamen an. Alle weiteren Worte sind die Argumente des Programms.

kommandoname argument1 argument2 argument3
Kommandoname wird intern auch als argument0 bezeichnet.

Beispiele:

```
ls
ls -lisa
ls -lisa Text
```

```
<Liste von Kommandos> ::= <Kommando> { <NL> <Kommando> } |
<Kommando> { ";" <Kommando> } |
<Kommando> { "|" <Kommando> } |
<Kommando> { "&&" <Kommando> } |
<Kommando> { "||" <Kommando> }
```

"|" - Pipe, die Standardausgabe des vorangegangenen Kommandos wird auf die Standardeingabe des nachfolgenden Kommandos geleitet.

```
ls | wc
ls -lisa | wc
```

j-p bell

Seite 2

<NL> - Kommandos werden nacheinander ausgeführt
(einzelne Kommandos stehen in mehreren Zeilen)

```
echo a
echo b
```

; - Kommandos werden nacheinander ausgeführt

```
echo -n a; echo -n b; echo c
```

&& - das nachfolgende Kommando wird ausgeführt, wenn
das vorangegangene Kommando den Returnwert 0
(true) liefert.

```
true && echo TRUE && echo true
false && echo FALSE && echo false
```

|| - das nachfolgende Kommando wird ausgeführt, wenn
das vorangegangene Kommando einen Returnwert
ungleich 0 (false) liefert.

```
true || echo TRUE || echo true
false || echo FALSE || echo false
```

Returnwert (Rückkehrkode): Jedes Programm liefert einen Returnwert.
0 wird als True interpretiert und alles andere als False.

<Kommando> ::= <einfaches Kommando> |
"(" <Liste von Kommandos> ";" ")"

"(" und ")" - Zusammenfassung von Kommandos, die in einer
Subshell abgearbeitet werden.
Nicht in Scripten !!!!

```
%1 ( cd Texte ; ls ; ) ; pwd
/home/bell/Tools
%2 pwd
/home/bell/Tools
%3
```

C-Shellvariable

```
-----
<C-Shellvariable> ::= <Nicht-Ziffer> { <Nicht-Ziffer> | <Ziffer> }
<Nicht-Ziffer> ::= "a"|"b"|...|"z"|"A"|"B" | ... |"Z"|"_"
<Ziffer>      ::= "0"..."9"
```

Definition von C-Shellvariable:

```
set {<Bezeichner>["="<wert>] }           lokale Variable
setenv <Bezeichner> <wert>                Umgebungsvariable
```

Felder von C-Shellvariablen:

```
set <Feldbezeichner>="( " { <Wert> } " )"
```

Zugriff auf eine C-Shellvariable/Felder/Feldelement:

```
$<Bezeichner>   oder ${<Bezeichner>}   - C-Shellvariable
${<Feldbezeichner>}["<Index>"]         - Feldelement
${#<Feldbezeichner>}                  - Anzahl der Feldelemente
${<Feldbezeichner>} oder ${<Feldbezeichner>}[*] - Alle Elemente
eines Feldes durch Leerzeichen getrennt.
Der Index eines Felder läuft von 1 bis ..
```

z.B.

```
 $#argv - Anzahl der Parameter
 $argv[1] - 1.Parameter
 $argv[12] - 12.Parameter
```

Achtung!!! Der Zugriff auf eine nichtdefinierte C-Shellvariabel liefert nicht die leere Zeichenkette!!!
 Paarige C-Shellvariable: z.B. path (Variable) und PATH (Umgebungsvariable) haben immer den gleichen Wert!!!

7.2.Cshell

Löschen von C-Shellvariablen:

```
unset <C-Shellvariable>
unsetenv <C-Shell-Umgebungsvariable>
```

Beispiele:

```
%1 echo $XX
XX: Undefined variable.
%2 set XX=asdf
%3 echo $XX
asdf
%4 set XX=asdf asdf
%5 echo $XX
asdf
%6 set XX="asdf asdf"
%7 echo $XX
asdf asdf
%8
%9 set XX=Anfang
10% echo $XX
Anfang
11% echo $XXswort
$XXswort: Undefined variable.
12% echo ${XX}swort
Anfangswort
13% (/bin/echo $XX)
Anfang
14%
```

```
cs1, cs1a, cs1b
```

```

14% cat echo_xx
#!/bin/csh
echo $XX
15% ./echo_xx
XX: Undefined variable.
16% set | grep XX
XX   Anfang
17% env | grep XX
18% setenv XX ANFANG
19% env | grep XX
XX   ANFANG
20% ./echo_xx
ANFANG
21% unset XX
22% echo $XX
ANFANG
23%
23% unsetenv XX
23% echo $XX
XX: Undefined variable.
24%

```

Weitere Zugriffsmöglichkeiten auf Variable

Zulässige Indexausdrücke:

```

<Zahl n> - n-te Element eines Feldes $array[4]
<Zahl n>-<Zahl m> - n-te bis m-te Element eines Feldes $array[4-9]
-<Zahl n> - entspricht 1-n
             $array[-n] gleich $array[1-n]
<Zahl n>-- - entspricht n-<letzte Feldelement>
             $array[n-] gleich $array[n-#$array]
$#<Feld>   - Anzahl der Feldelement ( ${#<Feld>} )

```

```

${?<variable>} oder ${?<variable>} - liefert 1,
                                wenn Variable gesetzt ist sonst 0

```

Modifikatoren

werden der Variablen nachgestellt, verändern den Ausgabewert

```

:h - head eines Pfadnamen
:gh - head eines Pfadnamen für jedes Wort eines Feldes
:t - tail(basename) eines Pfadnamen
:gt - tail(basename) eines Pfadnamen für jedes Wort eines Feldes
:r - abschneiden der Extension eines Wortes
:gr - abschneiden der Extension für jedes Wort eines Feldes
:e - liefert die Extension eines Wortes
:ge - liefert die Extension eines Wortes für jedes Wort eines Feldes
:q - Quoting für das Wort - keine weitere Substitution

```

Beispiele

```

8% set x=/dir1/dir2/file
9% set xx=( /dd1/dd2/ff1 /ddd1/ddd2/fff1 )
10% echo $x
/dir1/dir2/file
11% echo $xx
/dd1/dd2/ff1 /ddd1/ddd2/fff1
12% echo $x:h
/dir1/dir2
13% echo $xx:h
/dd1/dd2 /ddd1/ddd2/fff1
14% echo $xx:gh
/dd1/dd2 /ddd1/ddd2
15% echo $x:t
file
16% echo $xx:t
ff1 /ddd1/ddd2/fff1
17% echo $xx:gt
ff1 fff1
18% set f=xxx.tar
19% echo $f:r
xxx
20% echo $f:e
tar
21%

```

Quoting - Maskieren von Metazeichen

Quotings:

```

\ - vorgestellter "\" - das nachfolgende Metazeichen wird
    als normales Zeichen interpretiert.
' ... ' - Text in einfachen Apostrophs - Alle im Text enthaltenen
    Zeichen werden als normale Zeichen
    interpretiert. Auch "\" verliert
    seine Bedeutung.
" ... " - Text in Doppelapostrophs - Alle Metazeichen aufer:
    "\" und "$" !!!!!!!
    werden als normale Zeichen interpretiert.

```

Beispiele:

```

%1 touch xx\*
%2 touch xxx
%3 ls
xx*      xxx
%4 mv xx* YY
mv: Beim Verschieben mehrerer Dateien muß das letzte Argument
    ein Verzeichnis sein
%5 ls "xx*"
xx*
%6 ls 'xx*'
xx*
%7

```

Automatische C-Shellvariable:

```

$?, $status -
Returnwert des letzten Kommandos
%8 true ; echo $?
0
%9 false ; echo $?
1
$$ - Prozesnummer der aktuellen shell
%1 echo $$
1234
%2 touch xxx$$
%3 ls -lisa xxx*
-rw-r--r-- 1 bell unixsoft 0 Okt 30 08:57 xxx1234
%4

$# - Zahl der Positionsparameter
%5 echo $#
0
%6
$* - entspricht "$1 $2 ..." (theoretisch)
%1 echo $*
%2

$!, $schild -
Prozesnummer des letzten Hintergrundprozesses

$pwd - Aktuelle Working-Directory
cs2

```

j-p bell

Seite 11

7.2.Cshell

einige standard-C-Shell-Variablen:

```

cdpath - Suchpfad für rel. Pfadangaben für das
C-Shell-Kommando cd, keine Voreinstellung
%1 ls
Einleitung Regexpr Shell Texte
%2 set cdpath=(/home/bell /home/bell/Tools)
%3 cd Vorlesung
%4 pwd
/home/bell/Vorlesung
%5 cd Cshell
%6 pwd
/home/bell/Tools/Cshell
%7

echo - Wenn gesetzt, wird jedes Kommando expandiert ausgegeben.

histchars - definition der History-metazeichen ! und ^
(für Dialogeingabe)

history - Länge des History-Speichers (für Dialogeingabe)

home - Homedirectory
%1 echo $home
/home/bell
%2

ignoreeof - bei interaktiver C-Shell wird nur exit und logout als EOF
gewertet, nicht ^D

```

j-p bell

Seite 12

```

mail      - Mailfolder
           %3 echo $mail
           /vol/maillslv1/bell
           %4

noclobber - Wenn gesetzt, dann ist das Überschreiben von existierenden
Dateien bzw. das Anfügen an nicht existierenden
Dateien durch Ausgabeumlenkung nicht möglich.

noglob    - Wenn gesetzt, wird die Dateinamen-Expandierung unter-
drückt.

notify    - Wenn gesetzt, wird das Ende eines Hintregrundprozesse
sofort gemeldet.

path      - Pfad für ausführbare Kommandos
           %1 echo $path
           /usr/local/bin /bin /usr/bin /usr/X11R6/bin .

prompt    - primärer Prompt
           ! - Kommandonummer,
           % - normaler Nutzer, # - root

prompt2   - sekundärer Prompt
prompt3   - tertiärer Prompt

savehist  - Anzahl der Kommandos, die in .history gerettet werden
sollen.

shell     - Name der aktuellen Shell

```

```

term      - Terminaltype, wichtig für vi
           $ echo $term
           xterm
           $

time      - Wenn gesetzt, dann Zeitmessung für jedes Kommando
           1% set time
           2% ls
           cs1 cs1a cs1b cs2 cs2~ echo_xx
           0.000u 0.000s 0:00.00 0.0%    0+0k 0+0io 173pf+0w
           3%

verbose   - Wenn gesetzt, dann echo der History-Substitutionen eines
Kommandos

```

Expandieren von Dateinamen

```

-----
*   - beliebige Zeichenfolge ( auch leer)
?   - ein beliebiges Zeichen (nicht leer)
[...] - ein beliebiges Zeichen aus der Menge ...
{wort1,wort2,wort3,...} -
jedes Wort wird eingesetzt
~   - Pfadname es Homedirectories
~username - Pfadname des Homedirectories des Nutzers <username>
folgende Zeichen werden nur erkannt, wenn sie explizit im Muster
angegeben wurden:
. (Punkt am Anfang eines Dateinamen), /., /
ls *
ls *.tar
ls -lisad .*
ls [Ss]*
ls s3?
ls -d .?
ls ~bell
ls -lisa {cs1,cs1a,cs1b,cs}

```

j-p bell

Seite 15

7.2.Cshell

Ein- und Ausgabe

```

-----
Standardeingabe:      Kanal 0
Standardausgabe:     Kanal 1
Standardfehlerausgabe: Kanal 2

> file               - Umlenkung Standardausgabe in das File file
                    %1 echo start von asdf > protokollfile

>! file              - wie > file aber ohne Berücksichtigung von noclobber
                    (wenn noclobber=1, ist das Überschreiben von Files
                    normalerweise verboten)

>& file               - Umlenkung von Standardausgabe und Standardfehlerausgabe
                    in das File file

>&! file              - wie >& file aber ohne Berücksichtigung von noclobber

< file                - Umlenkung Standardeingabe von file
                    %1 cat < eingabefile

>> file               - Umlenkung Standardausgabe mit Anfügen in das File file
                    %1 echo Anfang des Scripts >> Protokollfile

>>! file              - wie >> file aber ohne Berücksichtigung von noclobber

>>& file               - Umlenkung Standardausgabe und Standardfehlerausgabe mit
                    Anfügen in das File file
                    %1 echo Anfang des Scripts >>& Protokollfile

```

j-p bell

Seite 16


```
>>! file - wie >>& file aber ohne Berücksichtigung von noclobber
<<ENDE - Lesen aus Shellscript bis ENDE

%1 SUMME='bc <<EOF
1+3
EOF\
%2 echo $SUMME
4
%3

|& - Umlenkung von Standardausgabe und Standardfehlerausgabe in
eine Pipe

`Kommando` - Umlenkung der Standardausgabe in eine Zeichenkette
echo "Start des Scripts: `date`" >> protokoll

$< - Einlesen eines Variablenwertes von Standardeingabe.

%1 set a=$<
asdf
%2 echo $a
asdf
%3
```

j-p bell

Seite 17

Ausdrücke

csch berechnet Ausdrücke wie in C. True ist ungleich 0 und False ist gleich 0. Für ganze Zahlen wird die größtmögliche Darstellungsform gewählt. Es tritt kein Überlauf auf.

Operatoren für Zahle bzw. Strings:

```
- Minuszeiche, Subtraktion
* Multiplikation
/ Division
% Modulo
+ Addition
<< Links-Shift
>> Rechts-Shift
< kleiner
> größer
<= kleiner-gleich
>= größer-gleich

== Strings: gleich
!= Strings: ungleich
=~ Strings: Pattermatching (Patter steht rechts)
!~ Strings: Pattermatching - ungleich

~ Bitweise Negation
& Bitweises UND
^ Bitweises XOR
| Bitweises Oder
```

j-p bell

Seite 18

Operatoren für Eigenschaften von Dateien:

```
-d Datei ist Directory
-e Datei existiert
-f Datei ist ein gewöhnliches File
-o Datei gehört dem aktuellen Nutzer
-r Datei lesbar
-w Datei schreibbar
-x Datei ausführbar
-z Datei ist leer
```

Operatoren für die Verknüpfung von einfachen Ausdrücken:

```
! logischer Negationsoperator
&& Logisches UND
|| Logisches ODER
( ) Klammerung eines Ausdrucks
```

Vorrangregeln für Operatoren:

```
( )      hohe Priorität
- Minuszeichen
! ~      ^
* / %    |
+ -      |
<< >>    |
<= >= < > |
== != =~ !~ |
& ^ |   |
&& ||   |
Bei gleicher Priorität von links nach rechts
niedrige Priorität
```

Zuweisungsoperator für Variable (built-in-Kommando) @

```
@ - Ausgabe aller momentan definierten C-Shellvariablen
@ <variable> "=" <Ausdruck> - Der Wert des Ausdrucks wird der
  Variablen zugewiesen.
@ <variable> "["<index>"]=" <Ausdruck> - Der Wert des Ausdrucks
  wird dem Feldelement zugewiesen.
```

cs3, cs3a

C-shell-Scripte

C-shell-Script: File mit gültigen C-Shell-Kommandos

Aufruf: csh <Shell-Script-Name>

cs4

```
%1 cat cs4
echo das ist ein C-Shellscript
%2 ls -l cs4
-rw-r--r-- 1 bell bell 29 Okt 30 17:55 cs4
%3 csh cs4
das ist ein C-Shellscript
%4 cs4
csh: cs4: command not found
%5 ./cs4
csh: ./cs4: Keine Berechtigung
%6 chmod +x cs4
%7 ./cs4
echo das ist ein C-Shellscript
%8
```

Am Anfang eines C-Shell-Scriptes sollte immer die benutzte Shell als Spezialkommentar (Major-Number: #!/bin/csh) eingetragen sein.

cs5

```
%1
%2 cat cs5
#!/bin/csh
echo das ist ein Shellscript
%3
```

j-p bell

Seite 21

7.2.Cshell

Können C-Shell-Scripte mit Parameter umgehen?

JA - erstmal die Parameter 1..9,10,11,...

```
$1 .. $9 $10 $11 ....
oder $argv[1] ... $argv[10] ...
#argv - Anzahl der Parameter ($#)
```

cs6

```
1% cat cs6
#!/bin/csh
# cs6
echo Anzahl der Parameter: $#
echo 1.Parameter: $1
echo 2.Parameter: $2
.....
echo 8.Parameter: $8
echo 9.Parameter: $9
2%
3% cs6 1 asdf 1=3 asdf=5 -asdf=8
Anzahl der Parameter: 5
1.Parameter: 1
2.Parameter: asdf
3.Parameter: 1=3
4.Parameter: asdf=5
5.Parameter: -asdf=8
6.Parameter:
7.Parameter:
8.Parameter:
9.Parameter:
4%
```

j-p bell

Seite 22

```

4%
5% #!/bin/csh
# cs6a
echo Anzahl der Parameter: $#argv
echo 1.Parameter: $argv[1]
echo 2.Parameter: $argv[2]
echo 3.Parameter: $argv[3]
echo 4.Parameter: $argv[4]
echo 5.Parameter: $argv[5]
echo 6.Parameter: $argv[6]
echo 7.Parameter: $argv[7]
echo 8.Parameter: $argv[8]
echo 9.Parameter: $argv[9]
echo 10.Parameter: $argv[10]
6% cs6a
Anzahl der Parameter: 0
argv: Subscript out of range.
%7 cs6a 1 2 3
1.Parameter: 1
2.Parameter: 2
3.Parameter: 3
argv: Subscript out of range.
8%

```

cs6a

Was passiert bei einer unbekanntem Zahl von Parametern?

Alle Parameter werden mittels "shift" um eine Position nach links verschoben.

cs7,cs7a

```

%1 cat cs7
#!/bin/sh
# cs7
echo Anzahl der Parameter: $#
echo 1.Parameter: $1
echo 2.Parameter: $2
echo 3.Parameter: $3
shift
echo 1.Parameter: $1
echo 2.Parameter: $2
echo 3.Parameter: $3
%2
%3 ./s7 1 2 3
Anzahl der Parameter: 3
1.Parameter: 1
2.Parameter: 2
3.Parameter: 3
1.Parameter: 2
2.Parameter: 3
3.Parameter:
%4

```

Kommandos - 2.Teil

```

<Kommando> ::= <einfaches Kommando> |
"(" <Liste von Kommandos> ";" ")" |
<if-Kommando> | <switch-Kommando> |
<while-Kommando> | <repeat-Kommando> |
<foreach-Kommando> | <goto-Kommando>

<if-Kommando> ::= "if" "(" <Ausdruck> ")" <einfaches Kommando> |
"if" "(" <Ausdruck> ")" then
  <Liste von Kommandos>
"endif"
"if" "(" <Ausdruck> ")" then
  <Liste von Kommandos>
"else"
  <Liste von Kommandos>
"endif"

```

Der Ausdruck nach "if" wird berechnet. Der Wert bestimmt die Verzweigungsbedingung. Ist der Wert gleich TRUE (ungleich 0), werden die Kommandos nach dem "then" abgearbeitet. Ist der Wert FALSE (ungleich Null), werden die Kommandos nach dem "else" abgearbeitet, falls diese vorhanden ist.
 Achtung: in der 1. Form darf die Kommandolist kein <NL> enthalten, das if-Kommando muß in einer Zeile stehen.

cs8, cs8a, cs8b

j-p bell

Seite 25

7.2.Cshell

```

<switch-Kommando> ::= "switch (<Wort>)"
  "case" <Muster> ":"
    <Liste von Kommandos>
  "breaksw"
{ "case" <Muster> ":"
  <Liste von Kommandos>
  "breaksw" }
["default:"
  <Liste von Kommandos> ]
"endsw"

```

Das Wort <Wort> wird der Reihe nach mit den Mustern vor den Kommandolisten verglichen. Wenn ein Muster "matchet" wird die zugehörige Kommandolist abgearbeitet und das case-Kommando beendet, wenn "breaksw" gefunden wird (Fortsetzung nach "endsw"). Fehlt ein "breaksw", werden die nachfolgenden Kommandos abgearbeitet bis ein "breaksw" oder ein "endsw" gefunden wird. Es gelten die gleichen Regeln wie bei der Dateierweiterung ("[..]", "**", "?", ~).

cs9

j-p bell

Seite 26

```
<while-Kommando>::= "while (" <Ausdruck>)"
    <Kommandoliste>
    "end"
```

Der Ausdruck nach dem "while" wird berechnet. Ist der Wert True (ungleich 0) wird die Kommandoliste abgearbeitet. Danach wird der Ausdruck nach dem "while" wieder berechnet. Dies geschieht solange, wie der Wert des Ausdrucks nach dem "while" gleich True ist. Ist der Wert False (gleich 0), wird das while-Kommando beendet (Fortsetzung nach dem "end"). Durch das Buildin-Kommando "break" kann das while-Kommando jederzeit beendet werden. Durch das Buildin-Kommando "continue" wird der nächste Schleifendurchlauf (Ausdrucksberechnung) gestartet.

cs10

```
<repeat-Kommando>::= "repeat" <Number> <einfaches Kommando>
```

Das einfache Kommando wird so oft abgearbeitet, wie durch <Number> spezifiziert wurde. Soll das Kommando auf eine neue Zeile gesetzt werden, so ist das Zeichen <NL> zu maskieren ("\n").

cs11

```
<foreach-Kommando>::= "foreach" <Laufvariable> "(" <wort> {<wort>} ")"
    <Kommandolist>
    "end"
```

Die Laufvariable nimmt nacheinander die Werte aus der Wortliste an und mit jedem Wort werden die Kommandos der Kommandoliste abgearbeitet.

Durch das Buildin-Kommando "break" kann das foreach-Kommando jederzeit beendet werden. Durch das Buildin-Kommando "continue" wird der nächste Schleifendurchlauf gestartet.

cs12

```
<goto-Kommando>::="goto" <Marke>
```

Das Kommando goto bewirkt einen Sprung zur Marke <Marke>. Eine Marke wird wie folgt definiert:

```
<Wort>": "
if ( $#argv == 0 ) goto ende
...
...
ende:
```

cs13

Interne C-Shell-Kommandos

```
-----  
<einfaches Kommando>::= ....| <interne C-Shell-Kommando>  
  
interne C-Shell-Kommando - Kommando innerhalb der C-Shell realisiert.  
  
Allgemeine C-Shell-Kommandos  
-----  
vielfach in C-Shell-Scripten benutzt  
  
#      Kommentar  
      # Das ist ein Kommentar bis Zeilenende  
  
break verlassen von Schleifenanweisungen (while, foreach).          cs14  
  
cd, chdir  
Definition des Working Directory (Current Directory)  
      Nur für die aktuelle C-Shell und nachfolgende Kommandos gültig.  
  
continue  
Beenden von Schleifen in Schleifenanweisung (while, foreach)          cs15  
  
echo {<argument>}  
Ausgabe der Argumente auf die Standardausgabe
```

j-p bell

Seite 29

```
eval {<argument>}  
Abarbeiten der Argumente in einer C-Shell  
1.Argument ist das Kommando.  
  
exec {<argumente>}  
Ausführen der Argumente als Kommando im aktuellen C-Shell-Prozess.  
1. Argumente ist das Kommando. Die C-Shell wird beendet.  
  
exit [<Rückkehrkode>]  
beenden der C-Shell mit einem Rückkehrkode  
  
glob [<Argumente>]  
Wie echo, aber ohne Ausgabe eines <NL>.  
  
limit [<Ressource> <Wert>]  
Setzen der Ressource auf den spezifizieren Wert.  
Wird keine Ressource angegeben, wird das aktuelle  
Limit für alle Ressourcen angegeben.  
  
login <user>  
Einloggen des Nutzers <user>. Nur bei login-C-Shell.  
  
logout  
Ausloggen, beenden einer login-C-Shell  
  
newgrp ["-"] <gid>  
Erzeugen einer neuen Shellinstanz mit der Gruppen-ID <gid>
```

j-p bell

Seite 30

```
nice <Priorität> [<Kommandos>]
Setzen der Priorität (19 >=Priorität>= -20) bei
der Ausführung eines Kommandos. nice ohne Kommando
setzt die Priorität der aktuellen C-Shell.

nohup [<Kommando>]
Ignorieren von HUP-Signalen für das Kommando.

onintr ["-" | <Marke>]
Behandlung von Signalen s.u.

pwd
Ausgabe des Workingdirectory

set
set <Variable>=
set <Variable>=<Wert>
set <Variable>=(<Wortliste>)
set <Variable>[<Index>]=<Wert>
Auflisten, Definition und Wertzuweisung für Variable
und Felder.

setenv <Variable> <Wert>
Definieren und Wertzuweisung für Umgebungsvariable.

shift [<Feld>]
Verschieben der Werte eines Feldes um eine Position nach
links. Ist kein Feld spezifiziert, werden die Parametern
um eins nach links verschoben.
```

j-p bell

Seite 31

```
source <Kommandodatei>
Lesen und Ausführen einer Kommandodatei in der
aktuellen C-Shell. ( Das .-Kommando der Shell)

time [<Kommando>]
Anzeigen der verbrauchte CPU-Zeit der aktuellen shell, bzw.
des Kommandos.

umask [<Mask>]
Setzen der Filecreationmask.
    Gesperrte Zugriffsrechte werden gesetzt.
    1 % umask 022
    2 % umask 077

unlimit [<Ressource>]
Setzen der bzw. aller Ressourcen auf den Maximalwert.

unset <Shellvariable>
Löschen von Variablen.
    Die Variable ist danach undefiniert.
    1 % set xxx=asdf
    2 % echo $xxx
    asdf
    3 % unset xxx
    4 % echo $xxx
    xxx: Undefined variable.
    5 %

unsetenv
Löschen einer Umgebungsvariablen.
```

j-p bell

Seite 32

@

Ausgabe der aktuellen C-Shell-Variablen

```
( set )
```

```
@<Variable>="<Ausdruck>
@<Feld>["<Index>"]="<Ausdruck>
  Berechnung von Ausdrücken und Wertzuweisung des Ergebnisses
  zu Variablen und Feldelementen
```

Kommandos für die Arbeit mit Jobs

```
-----
```

```
<job> - Job-Spezifikation
      "%"<Jobnummer>
      <Prozessnummer>
      "%" aktueller Job
      "%-" vorheriger aktueller Job
```

```
bg {<job>}
```

Spezifizierte, zuvor gestoppte Jobs im Hintergrund weiterarbeiten lassen.

```
fg {<job>}
```

Abarbeiten der Jobs im Vordergrund. Die Jobs liefen vorher im Hintergrund oder waren gestoppt.

```
jobs [-l]
```

Ausgabe einer Liste aller Jobs mit Statusangabe. bei -l werden die Prozessnummern mit angegeben.

j-p bell

Seite 33

7.2.Cshell

7.4.2017

kill

```
kill [-<Signalnr.>] {<job>}
```

Senden eines Signals an die spezifizierten Jobs. Wenn kein Signal angegeben wurde, wird das Signal TERM (15) gesendet. Für Jobs können auch Prozessnummern angegeben werden.

```
kill -STOP {<job>}
```

Stoppen eines Prozesses für Vordergrundprozesse: <CNTRL Z>

```
kill -CONT {<job>}
```

fortsetzen eines Prozesses

```
kill -l
```

gibt eine Liste der zulässigen Signale aus.

```
notify {<Job>}
```

Das Ende der spezifizierten Jobs wird sofort signalisiert. wenn die C-Shell-Variablen "notify" gesetzt ist wird das Ende jedes Jobs sofort gemeldet.

```
stop {<job>}
```

Anhalten des spezifizierten Jobs

```
suspend
```

Anhalten der aktuellen Shell. Der zuvor angehaltene C-Shell-Prozess wird fortgesetzt. Für den Wechsel zwischen zwei Shells.

```
wait
```

Warten auf das Ende eines Jobs

cs17,cs17a

j-p bell

Seite 34

Kommandos für die Dialogarbeit

```
alias [<Name>] [<Wortliste>]
Anzeigen und setzen von Aliasen
%1 alias ll ls -lisa --color
%2 alias
ll (ls -lisa --color)
%3
```

```
unalias <Name>
Löschen von Aliasen
```

```
hashstat
Ausgabe der Kommando-Hash-Tabelle
```

```
rehash
Kommando-Hash-Tabelle neu aufbauen
```

```
unhash
Abschalten der Kommando-Hash-Tabelle
```

```
history [-r] [<Nr>]
Kommando-History anzeigen.
-r - umgekehrte Reihenfolge
<Nr> - letzten <Nr> Kommandos
```

```
dirs [-l]
Ausgabe des Directory-Stacks
```

j-p bell

Seite 35

```
popd [<n>]
```

n Element aus dem Directoystack entfernen.

Wenn n nicht spezifiziert ist wird das oberste Element entfernt.

```
pushd
vertauscht die beiden oberen Elemente des Directory-
stacks.
```

```
pushd <Pfadname>
<Pfadname> wird oberstes Stackelement (aktuelles
Working-Directory)
```

```
pushd +<Number>
Rotation des ganzen Stacks, so daß das n-te Element
oben steht. Das oberste Element hat die Number 0.
```

Beispiel:

```
1 % pushd ~
~ ~/Tools/Cshell
2 % pushd Tools
~/Tools ~ ~/Tools/Cshell
3 % pushd Texte
~/Tools/Texte ~/Tools ~ ~/Tools/Cshell
4 % dirs
~/Tools/Texte ~/Tools ~ ~/Tools/Cshell
5 % pushd +1
~/Tools ~ ~/Tools/Cshell ~/Tools/Texte
6 %
```

j-p bell

Seite 36

Signalbehandlung

Die C-shell kann folgende Signale abfangen. :

```
intr - 2 <CTRL>C
hangup - 3 <CTRL>\
```

Die Signalbehandlung wird durch die Buildin-Funktion onintr gesteuert.

```
onintr
zurücksetzen der Signalbehandlung auf Standard
onintr -
Die Signale (intr, hangup, terminate) werden
ignoriert.
onintr <Marke>
Beim Auftreten der obigen Signale wird zur
Marke verzweigt.
```

Beispiel:

```
onintr ende
...
...
ende:
exit 1

trap1, trap2, trap4
```

j-p bell

Seite 37

Kommandozeileneditor

History-Substitutionen

```
!<Angabe>[:<Auswahl>][[:<Modifikatoren>]
```

```
<Angabe> Kommando
```

```
!! - letztes Kommando
!n - Kommando mit der Nummer <n>
!-n - n-te Kommando rückwärts
!<string> - Kommando das mit <string> anfängt
!?<string>? - Kommando das <string> enthält
```

```
<Auswahl> Wortauswahl
```

```
<Ziffer n>
^ - n-tes Wort (0.te Wort ist Kommando)
$ - erstes Argument nach Kommando
<Ziffer m>-<Ziffer n> - m-te bis n-te Wort
<n Ziffer> - 0-n Ziffer
<n Ziffer>- - n-te bis vorletztes Wort
* - 1.Argument bis letztes Argument
<Ziffer n>* - n-te Argument bis letztes Argument
```

j-p bell

Seite 38

```

<Modifikatoren> Leistung
h - Pfadname des Wortes
t - Basisname des Wortes
r - Wort ohne Extension
e - Extension des Wortes
s/<str1>/<str2>/ - Substitution, <str1> wird durch
                   <str2> ersetzt
& - Wiederholung der letzten Substitution
g - Globalisierung der Substitution auf
   alle ausgewählte Worte
p - Anzeigen des neuen Kommandos nicht ausführen
q - Quoting der Kommandozeile, keine weitere
   Expandierungen
x - Quoting wie q, aber Aufteilung in Worte

```

Beispiele:

```

46 % set history=30
47 % echo eins zwei drei vier
eins zwei drei vier
48 % !!
echo eins zwei drei vier
eins zwei drei vier
49 % !! fuenf
echo eins zwei drei vier fuenf
eins zwei drei vier fuenf
50 %

```

j-p bell

Seite 39

```

50 % echo !!:2-4
echo zwei drei vier
zwei drei vier
51 % !49:0 !49:3-5
echo drei vier fuenf
drei vier fuenf
52 % !49:s/fuenf/sechs/
echo eins zwei drei vier sechs
eins zwei drei vier sechs
53 % echo !!:^
echo eins
eins
54 % echo /vol/delta-vol5/beta.tar
/vol/delta-vol5/beta.tar
55 % echo !54:^:h
echo /vol/delta-vol5
/vol/delta-vol5
56 % echo !54:^:t
echo beta.tar
beta.tar
57 % echo !54:^:r
echo /vol/delta-vol5/beta
/vol/delta-vol5/beta
58 % echo !54:^:e
echo tar
tar
59 %

```

j-p bell

Seite 40

Kommandozeile und Initialisierung von csh

Aufrufsyntax der C-Shell:

```
csh [-c <Kommando>] [-s <Argumente>][-efintvVxX] {<Kommando>}
-c <Kommando>      - Ausführen des Kommandos <Kommando>
-e                - csh verlassen, wenn ein Kommando fehlerhaft
                  beendet wird
-f $home/.cshrc   - $home/.cshrc nicht abarbeiten (fast)
-i               - csh interaktiv starten
-n              - Kommandos nur syntaktisch prüfen,
                  nicht abarbeiten (noexec)
-s {<Argument>} - interaktive Subshell starten, Argumente werden
                  der Subshell übergeben.
-t              - Nur ein Kommando ausführen, <NL> kann durch "\"
                  maskiert werden
-v              - c-Shell-Variablen verbose wird gesetzt. Nach jeder
                  History-Substitution wird die Kommandozeile ange-
                  zeigt.
-V              - wie -v, aber c-Shell-Variablen verbose wird vor
                  Abarbeitung von .cshrc gesetzt.
-x              - c-Shell-Variablen echo wird gesetzt. Jede
                  Kommandozeile wird vor der Ausführung "geecho".
-X              - wie -x, aber c-Shell-Variablen echo wird vor
                  Abarbeitung von .cshrc gesetzt.
```

j-p bell

Seite 41

Initialisierung

Wenn C-Shell als login-Shell läuft, werden folgende Dateien abgearbeitet:

1. /etc/cshrc oder /etc/csh.cshrc
Systemweite csh-Grundeinstellungen
2. /etc/login oder /etc/csh.login
Systemweite Grundeinstellungen für login-Shell
3. \$home/.cshrc
nutzerspezifische csh-Einstellungen
4. \$home/.login
nutzerspezifische csh-Einstellungen für login

Wenn C-Shell nicht als login-Shell läuft, werden nur die Dateien:

1. /etc/cshrc oder /etc/csh.cshrc
2. \$home/.cshrc

abgearbeitet.

Wurde die Option -f gesetzt wird keine Datei abgearbeitet

Wenn eine login-Shell beendet wird, wird die Datei

\$home/.logout

abgearbeitet.

j-p bell

Seite 42

```

Beispiel:
Linux:
# /etc/cshrc
#
# csh configuration for all shell invocations. Currently, a prompt.
echo "in /etc/csh.cshrc"
[ `id -u` = "0" ] && limit coredumpsize 1000000
if ($?prompt) then
  if ($?tcsh) then
    set prompt='[%n%m %c]$ '
  else
    set prompt=[`id -nu`@`hostname -s`]\]$\  

  endif
endif
set i = /etc/profile.d/inputrc.csh
if (-r $i) then
  source $i
endif

```

j-p bell

Seite 43

```

# /etc/csh.login
# System wide environment and startup programs for csh users
if (! -r /usr/bin) then
  unhash
endif
if ($?PATH) then
  setenv PATH "${PATH}:/usr/X11R6/bin"
else
  setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin"
endif
limit coredumpsize unlimited
[ `id -gn` = `id -un` -a `id -u` -gt 14 ]
if $status then
  umask 022
else
  umask 002
endif
setenv HOSTNAME `bin/hostname`
set history=1000
if (-d /etc/profile.d) then
  set nonomatch
  foreach i ( /etc/profile.d/*.csh )
    if (-r $i) then
      source $i
    endif
  end
  unset i nonomatch
endif

```

j-p bell

Seite 44

Für Profis: Abarbeiten einer Kommandozeile durch die C-Shell:

1. Entfernen aller \n-Zeichen
2. History-Substitution
3. Speichern der aktuellen Kommandozeile im History-Puffer
4. Alias-Substitution
5. Parametersubstitution und Auswertung der Variablenzuweisungen
6. Kommandosubstitution
7. Expandieren der Dateinamen
8. Auswertung der E/A-Umlenkung
9. Kommando lokalisieren und ausführen
(in der gleichen C-Shell: buildin-Kommando
fork - neuer Prozeß: sonstige Kommandos)