7.4. Die Turbe-C-Shell - tcsh

Paul DuBois: "benutze die tcsh und nicht die csh"

Manual: tcsh - C shell with file name completion and command line editing

Vorteile der tcsh gegenüber der csh:

- Kommandozeileneditor
- Im Dialog eine Komplettierung für Programm- und Variablennamen
- Filenamenkompettierung
- Spelling correction - für Leute mit vielen Tipfehlern
- rm * - Kontrolle
- bessere Standardisierung
- Quelltext verfügbar

Metazeichen

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

```
Aufbau eines Kommandos - 1.Teil
-------------------------------
wie C-Shell:
   <Kommando> ::= <einfaches Kommando> | ...

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

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

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



```
Turbo-C-Shellvariable
---------------------
wie C-Shell + Erweiterungen:

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

   Definition von Turbo-C-Shellvariable:
         set ["-r"]{<Bezeichner>["="<wert>] }                        !!!!
         setenv <Bezeichner>  <wert>

   Felder von Turbo-C-Shellvariablen:
         set ["-r"] <Feldbezeichner>"=(" { <Wert> } ")"              !!!!
              -r   - readonly-Variable, auch mit unset ist diese      !!!!
                     Variable nicht zu löschen.                       !!!!

  Zugriff auf eine Turbo-C-Shellvariable/Felder/Feldelement:

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

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

Löschen von Turbo-C-Shellvariablen:

```
unset <Turbo-C-Shell-Variable>
unset <Turbo-C-Shell-Umgebungsvariable>
```

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 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
```

Quoting - Maskieren von Metazeichen

Quotings:

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

```
Automatische Turbo-C-Shellvariable:

      $?, $status - Returnwert des letzten Kommandos

      $$          - Prozeßnummer der aktuellen Shell

      $#          - Zahl der Positionsparameter

      $*          - entspricht "$1 $2 ..."

      $!, $child  - Prozeßnummer des letzten Hintergrundprozesses

      $cwd        - Aktuelle Working-Directory

einige Standard-Turbo-C-Shell-Variable:
  etwas mehr als bei der C-Shell

   autologout - Zeit in Min. bis zum automatischen Logout nach der
                letzten Eingabe                                    !!!!

   cdpath     - Suchpfad für rel. Pfadangaben für das Turbo-C-Shell-
                Kommando cd, keine Voreinstellung

   correct    - Spelling correction, cmd - nur für Kommando, all - alles
                set correct=cmd  oder set correct=all
                                                                   !!!!
   echo       - Wenn gesetzt, wird jedes Kommando expandiert ausgegeben.
```



```
   fignore    - Liste von Suffixen von Filenamen, die bei der Filenamen-
                komplettierung ignoriert werden.                   !!!!
                     set fignore=( .o .bak .tmp )

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

   history    - Länge des History-Speichers (für Dialogeingabe)

   home       - Homedirectory

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

   mail       - Mailfolder

   noclobber  - Wenn gesetzt, dann ist das Überschreiben von existieren-
                den 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, beacht PATH
```

```
prompt     - primärer Prompt,                                    !!!!
prompt2    - sekundärer Prompt,   prompt3 - tertiärer Prompt
             %w - Monatsname               %W - Monatsnummer
             %d - Wochentag                %D - Tagesnummer (im Monat)
             %y - Jahresnummer zweistel.   %Y - Jahresnummer vierstel.
             %p - Uhrzeit (engl. mit Sek.)  %P - Uhrzeit (hh:mm:xx)
             %t - Uhrzeit (engl.ohne Sek.)  %T - Uhrzeit (hh:mm)
             !  - Kommandonummer in der Shell

rmstar     - Wenn gesetzt, werden rm-Kommandos mit "*"-Parameter  !!!!
             nachgefragt.

savedirs   - Wenn gesetzt, wird bei login-Shells beim logout      !!!!
             der Directorystack in ~/.cshrdirs gerettet und
             beim login wieder geladen.

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

shell      - Name der aktuellen Shell

tcsh       - Enthält die Versionsnummer der Turbo-C-Shell          !!!!

term       - Terminaltype, wichtig für vi, beachte TERM

time       - Zeitmessung für jedes Kommando

verbose    - echo der History-Substitutionen eines Kommandos

version    -  Enthält die Verionsangaben und Optionen der Turbo-C-Shell
```



```
Expandieren von Dateinamen
--------------------------
 in Scripten:
    *         - beliebige Zeichenfolge ( auch leer)

    ?         - ein beliebiges Zeichen (nicht leer)

    [...]     - ein beliebiges Zeichen aus der Menge ...

    [^...]    - kein Zeichen aus der Menge ...                     !!!!

    ^<muster> - Files, die nicht dem Muster entsprechen            !!!!

    {wort1,wort2,wort3,...} -
             jedes Wort wird eingesetzt, kein Patternmattching

    ~         - Pfadname es Homedirectoies

    ~username - Pfadname des Homedirectories des Nutzers <username>

    folgende Zeichen werden nur erkannt, wenn sie explizit im Muster
    angegeben wurden:

        . (Punkt am Anfang eines Dateinamen)
        /.
        /
```

```
im Dialog zusätzlich:

   <TAB>    - vervollständigen des Filenamens                        !!!!
   <^D>     - Anzeigen der Files zum Vervollständigen                !!!!
   <^X*>    - Anzeigen der Files zum Vervollständigen wenn Pattern   !!!!
              im Filenamen vorkommen
   Korrektur von Tipfehlern                                          !!!!

      set correct=cmd          nur für Kommandos
          oder
      set correct=all          Kommandos und Filenamen


    Beispiel:

     32% mroe cs11

      CORRECT>more cs11 (y|n|e|a)? yes
      #!/bin/csh
      # cs11
      echo Anzahl der Parameter: $#argv
      if ( $#argv != 1 ) then
          echo use: filename
          exit
      endif
      repeat 10  cat $argv[1] >>! big
      33%
```



```
Ein- und Ausgabe
----------------
wie bei C-Shell

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

> file        - Umlenkung Standardausgabe in das File   file

>! 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

>> file       - Umlenkung Standarausgabe mit Anfügen in das File file

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

>>& file      - Umlenkung Standarausgabe und Standardfehlerausgabe mit
                 Anfügen in das File file

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

```
<<ENDE         - Lesen aus Shellscript bis ENDE

|&            - Umlenkung von Standardausgabe und Standardfehlerausgabe in
                eine Pipe

`Kommando`     - Umlenkung der Standardausgabe in eine Zeichenkette

$<             - Einlesen eines Variabelenwertes von Standardeingabe.
```



```
Ausdrücke
---------

Turbo-C-Shell 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 Zahlen bzw. Strings:
   -       Minuszeiche, Subtraktion
   *       Multiplikation
   /       Division
   %       Modulo
   +       Addition
   <<      Links-Shift
   >>      Rechts-Shift
   <       kleiner
   >       größer
   <=      kleiner-gleich
   >=      größer-gleich

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

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

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



```
Zuweisungsoperator für Variable (built-in-Kommando)  @

     @      -  Ausgabe aller momentan definierten Turbo-C-Shellvariablen

     @<variable>"="<Ausdruck>  - Der Wert des Ausdrucks wird der
                                 Variablen zugewiesen.

     @<variable>"["<index>"]="<Ausdruck>   - Der Wert des Ausdrucks
                                 wird dem Feldelement zugewiesen.
```

```
Turbo-C-Shell-Scripte
---------------------

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

   Aufruf:  tcsh <Shell-Script-Name>

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

    Können Turbo-C-Shell-Scripte mit Parameter umgehen?
      JA - erstmal die Parameter 1..9,10,11,...
           $1 .. $9 $10 $11 ....
           oder $argv[1] ... $argv[10] ...
           #argv - Anzahl der Parameter ($#)

    Was passiert bei einer unbekannten Zahl von Parametern?

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

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.
```

```
<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 Kommandoliste abgearbeitet und das case-Kommando beendet, wenn "breadksw" 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 ("[..]", "*", "?", ~).


```
<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.

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

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

```
    <foreach-Kommando>::= "for" <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.


    <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:
```



```
Interne Turbo-C-Shell-Kommandos
-------------------------------

    <einfaches Kommando>::=   ....| <interne Turbo-C-Shell-Kommando>

interne Turbo-C-Shell-Kommanod - Kommando innerhalb der Turbo-C-Shell realisi

Allgemeine Turbo-C-Shell-Kommandos
----------------------------------
vielfach Turbo-C-Shell-Scripten benutzt

#
     Kommentar
         # Das ist ein Kommentar bis Zeilenende


alloc Speicherbelegung anzeigen                                      !!!!!

bindkey [-l|-d|-e|-v|-u]                                             !!!!!
bindkey [-a] [-b] [-k] [-r] [--] key
bindkey [-a] [-b] [-k] [-c|-s] [--] key command
      Steuert die Bedeutung von Tasktenkombination für die Zeileneingabe
         -v   Bedeutung wie bei vi
         -e   Bedeutung wie bei emacs
         -d   default
         -a   Ausgabe aller Bindungen(Tastenkombination und Bezeichnung)
         -l   Ausgabe aller Tastenbezeichnung und deren Bedeutungen

break
     verlassen von Schleifenanweisungen (while, foreach).
```

```
breaksw   Switch verlassen

buy   logout

cd, chdir
cd [-p] [-l] [-n|-v] [name]
     Definition des Working Directory (Current Directory)
       Nur für die aktuelle Turbo-C-Shell und nachfolgende Kommandos gültig.
cd -                                                                      !!!!
     vorherige Working Directory (zum Wechseln zwischen zwei Directories)

complete [command  [word/pattern/list[:select]/[[suffix]/] ...]] (+)    !!!!
     Erlaubt die Definition von Vervollständigungsregeln in Abhängigkeiten
     vom Kommando:  complete  cd 'p/1/d/'

continue
     Beenden von Schleifen in Schleifenanweisung (while, foreach)

echo {<argument>}
     Ausgabe der Argumente auf die Standardausgabe

echotc [-sv] arg                                                   !!!!!
     gibt die Eigenschaften des Terminals entsprechend der Termcap.
     arg - Terminaleigenschaft
     -s  - leere Zeichkette bei unbekannten Terminaleigenschaften
     -v  - verbose
       1 % echotc lines
       14
       2 % set lines=`echotc lines`
```



```
eval {<argument>}
     Abarbeiten der Argumente in einer Turbo-C-Shell
     1.Argument ist das Kommando.

exec {<argumente>}
     Ausführen der Argumente als Kommando im aktuellen Turbo-C-Shell-Prozeß.
     1. Argumente ist das Kommando. Die Turbo-C-Shell wird beendet.

exit [<Rückkehrkode>]
     beenden der Turbo-C-Shell mit einem Rückkehrkode

filetest -op file ...                                                   !!!!!
     Testet die spezifizierte Fileeigenschaft (-op) und gibt 0 oder 1 aus
        1 % filetest -rw asdf
        1
        2 %

glob [<Argumente>]
     Wie echo, aber ohne Ausgabe eines <NL>.

hup  <command>                                                          !!!!
     Das gestartete Kommando wird bei einem Signal SIGHUP beendet.

inlib  shared-library
     Hinzufügen von Shared-Libraries zur momentanen Umgebung            !!!!

limit [<Ressource> <Wert>]
     Setzen der Ressource auf den spezifizieren Wert.
     Wird keine Ressource angegeben, wird das aktuelle
     Limit für alle Ressourcen angegeben.
```

```
log  Ausgabe der watch-Variablen                                      !!!!

login <user>
     Einloggen des Nutzers <user>. Nur bei login-Turbo-C-Shell.

logout
     Ausloggen, beenden einer login-Turbo-C-Shell

ls-F <Optionen>  <Argumente>                                         !!!!
     ls -F aber schneller

newgrp ["-"] <gid>
     Erzeugen einer neuen Shellinstanz mit der Gruppen-ID <gid>

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

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

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

printenv                                                             !!!!
     Ausgabe der Umgebungsvariablen
```



```
pwd
     Ausgabe des Workingdirectory

sched
sched [+]hh:mm <commando>
sched -<nn>
     Starten des spezifizieren Kommandos zur angegebenen Zeit        !!!!

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

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

settc <cap> <value>                                                  !!!!
     setzen der Terminaleigenschaft

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

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

```
telltc   Ausgabe der Terminal-Eigenschaften                          !!!!

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

umask [<Mask>]
     Setzen der Filecreationmask.
       Gesperrte Zugriffsrechte werden gesetzt.

uncomplete <pattern>                                                 !!!!
     Rücksetzen aller Kompletierungsregeln, die dem Pattern entspricht.
     uncomplete *  setzt alles zurück.

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

unset <Shellvariable>
     Löschen von Variablen.  Die Variable ist danach undefiniert.

unsetenv
     Löschen einer Umgebungsvariablen.

ver  [systype [command]]                                             !!!!
     Anzeigen und Setzen des Systemtypes und Ausführen des Kommandos
     unter dem Systemtype  (nur bei Systemen mit mehreren Systemtypen)

where <command>                                                      !!!!
     liefert alle Instanzen eines Kommandos
```



```
which <command>                                                      !!!!
     liefert die benutzte Instanz eines Kommandos


%<Job>                                                               !!!!
     Kürzel für
        fg %<Job>

%<Job> &                                                             !!!!
     Kürzel für
        bg  %<Job>

@
     Ausgabe der aktuellen Turbo-C-Shell-Variablen
     ( set )

@<Variable>"="<Ausdruck>
@<Feld>"["<Index>"]="<Ausdruck>
     Berechnung von Ausdrücken und Wertzuweisung des Ergebnisses
     zu Variabelen 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 Fordergrund. 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.
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 Fordergrundprozesse:  <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 Turbo-C-Shell-Variable "notify" gesetzt ist wird das Ende
     jedes Jobs sofort gemeldet.

stop  {<job>}
     Anhalten des spezifizierten Jobs

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

wait
     Warten auf das Ende eines Jobs
```

```
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] [-n|-v]
dirs -S|-L [filename] (+)
dirs -c (+)
dirs [-l]
     Ausgabe des Directory-Stacks
```



```
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 %
```

```
Signalbehandlung
----------------

Die Turbo-C-Shell kann folgende Signale abfangen. :
      intr   - <CNTRL C>
      hangup - Beenden einer Verbindung
      terminate - Signal 15 - kill

      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 des obigen Signale wird zur
           Marke verzweigt.

     Beispiel:

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



```
Kommandozeileneditor
--------------------
wie csh:

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
$               - letztes Argument
<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
```

```
<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 weiter
                  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 %
```



```
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 %
```

```
Besonderheiten bei der Turbo-C-Shell bei der Kommandoeingabe:

Key-Bindungen:

  Die Key-Bindungen für das Editieren der Kommanodzeile können verändert
  werden mittels des Kommandos     bindkey

      bindkey   -d    -   default, wie csh
      bindkey   -v    -   wie vi
      bindkey   -e    -   wie emacs

vi-Modus:

   Nach dem Setzen der Key-Bindung vi ist man im Insertmodus.
   Nach Ausführung eines Kommandos ist man ebenfalls wieder im Insertmodus.
   Den Insertmodus verläßt man durch <ESC>.
   Ein <Enter> bewirkt das Ausführen des momentan editierten Kommandos
   Im Kommandomodus wirken die Tasteneingaben wie beim Kommadomodus des vi.
```



```
Kommandozeile und Initialisierung von tcsh
----------------------------------------

Aufrufsyntax der Turbo-C-Shell:

   tcsh  [-bcdefimnqstvVxX] [-Dname[=value]] [arg ...]

   -b   - vorherige Optionen vergessen

   -c <commando> - ausführen des Kommandos

   -d   - Laden des Directorystacks von ~/.cshdirs

   -Dname=[=value] - Definition einer Umgebungsvariable

   -e   - tcsh exit bei abnormalem Kommandoende

   -f   - Fast Start

   -i   - interaktive TCSH

   -l   - login TCSH  ( Argument 0 gleich "-")

   -n   - nur Kommando-Parsing

   -q   - SIGQUIT unterstützt, kein Jobcontrol

   -s   - Commando input von Standardeingabe
```

```
-t   - Eine Zeile von Standardeingabe als Eingabe für die Shell

-v   - verbose, echo des Kommandos vor der Ausführung

-x   - setzen der Shellvariablen echo (verbose voll nach
       Filenamenexpansion)

-V   - wie -v, aber bereits vor Abarbeitung von ~/.tcshrc wirksam

-X   - wie -x, aber bereits vor Abarbeitung von ~/.tcshrc wirksam
```



```
Initialisierung
---------------
Wenn Turbo-C-Shell als login-Shell läuft, werden folgende Dateien abgearbeite

     1. /etc/cshrc oder /etc/csh.cshrc oder /etc/.cshrc
              Systemweite tcsh-Grundeinstellungen
     2. /etc/login  oder /etc/csh.login oder /etc/.login
              Systemweite Grundeinstellungen für login-Shell
     3. $home/.tcshrc oder $home/.cshrc
              nutzerspezifische tcsh-Einstellungen
     4. $home/.login
              nutzerspezifische csh-Einstellungen für login
     4. $home/.cshdirs
              cd-Cache lesen und initialisieren  für login

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

     1. /etc/tcshrc oder /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, werden die Datei

      /etc/logout oder /etc/.logout
    und
      $home/.logout

abgearbeitet.
```

```
Für Profis: Abarbeiten einer Kommandozeile durch die Turbo-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 Turbo-C-Shell: buildin-Kommando
       fork - neuer Prozeß: sonstige Kommandos )
```