# 11. Shells

Shell - die Kommandosprache des Unix.
sh, csh, ksh, tcsh, bash oder wie sag ich's meinem Computer.

**Etwas Geschicht:**

```
    sh   - "Bourne Shell" - Urvater aller Shells [Steve Bourne]

    csh  - "C-Shell" - der Versuch einer neuen Shell [BSD-Entwicklungslabor]

    ksh  - "Korn-Shell" - von der sh abgeleitet  [David Korn (AT&T)]
           ksh   - 1983
           ksh86 - 1986 - mit Co-Prozessen
           ksh88 - 1988 - Detailverbesserungen
           ksh93 - 1993 - Geschwindigkeitsverbesserungen
                          mächtiger Zeileneditor
    tcsh - "Tourbo C-Shell" - C-Shell+Zeileneditor+bessers Eingabeverhalten

    bash - "Bourne-Again-Shell" - konsequente Weiterentwicklung der
                                  Bourne-Shell [GNU-Projekt]

    zsh  - "Z-Shell" - eierlegendes "Wollmilchschwein", emuliert
                       Bourne Shell, C-Shell, Korn-Shell[Paul Falstad]
```





Shells in Betriebssystemen:

| | |
|---|---|
| Solaris | ksh(sh) |
| IRIX | ksh(sh) |
| AIX | sh,ksh |
| HP-UX | sh,ksh |
| MacOS X | zsh(sh) |
| Linux | bash(sh), pdksh(ksh), ash, zsh |
| OpenBSD | pdksh(sh,ksh), bash, zsh |
| FreeBSD | pdksh(sh,ksh), bash, zsh |
| NetBSD | sh, pdksh(ksh) |

bash und zsh sind für alle Betriebssystem verfügbar.

**Einige Grundbegriffe**
**--------------------**

**Befehl**
    **Shellbefehl oder Bezeichner eines ausführbaren Programms**
      **z.B.:  echo, cd, ls, cp, umask**
**Programm**
    **ausführbares Programm, im File gespeichert**

**Shellbefehl**
    **"built-in" Befehl der Shell. Ist innerhalb der Shell implementiert**

**Kommandozeile (command line)**
    **Eingabezeile, die eine Shell zu interpretieren hat**

**Metazeichen (metacharacter)**
    **Zeichen, das von der Shell eine besonder Bedeutung zugewiesen bekommen hat**

**Trennzeichen**
    **Leerzeichen oder Tabulatoren**

**Bezeichner (identifier)**
    **Folge von Buchstaben, Ziffern und Unterstrichen, beginnend mit Buchstabe oder Unterstrich**





**Wort (word)**
    **Folge von Zeichen, die durch ein oder mehrere der folgenden Zeichen begrenzt ist**

      **';' '&' '(' ')' '|' '<' '>' <Newline> <Blank> <Tabulator>**

    **Steht vor diesen Begrenzerzeichen ein'\', gehört das Zeichen zum Wort.**

**Beispiel:**

```
> xxx=asdf xyz
> sh: xyz: not found
> xxx=asdf\ xyz
> echo $xxx
asdf xyz
>
```

```
11.1. Die Bourne-Shell und Bourne-Again-Shell
=============================================

Metazeichen
-----------
   |       - Pipe
   *       - kein, ein oder mehrere Zeichen
   ?       - ein beliebiges Zeichen
   [...]   - eines der in den Klammern angegebenen Zeichen
   [!...]  - nicht eines der in den Klammern angegebenen Zeichen
   ;       - Trennzeichen für Kommandos
   &       - Kommando in Hintergrund, E/A-Verkettung
   `kommando` - Ersetzung durch Standardausgabe
   (  )    - Subshell benutzen
   $       - 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  - Ausgabe
   <       - E/A-Umlenkung  - Eingabe
```





```
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> }

                  <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
```

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

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

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

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

**s0**

**||** - **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
```

**s0**

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





**Erweiterung der Definition von <Kommando>**

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

**"(" und ")"** - **Zusammenfassung von Kommandos, die in einer Subshell abgearbeitet werden**

**"{" und "}"** - **Zusammenfassung von Kommandos, die in der gleichen Shell ablaufen.**

**Beispiel für die Wirkungsweise von "( ... )" und "{ ... }"**

```
# pwd
/home/bell/Tools
# ( cd Texte ; pwd ; ) ; pwd
/home/bell/Tools/Texte
/home/bell/Tools
# pwd
/home/bell/Tools
#
# { cd Texte ; pwd ; } ; pwd
/home/bell/Tools/Texte
/home/bell/Tools/Texte
# pwd
/home/bell/Tools/Texte
#
```

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

      Wertzuweisung für Shellvariable:

           <Bezeichner>=<wert>

      Zugriff auf eine Shellvariable:

            $<Bezeichner>   oder ${<Bezeichner>}

      Löschen von Shellvariablen:
            unset <Shellvariable>

      Export von Shellvariablen:
            export <shellvariable> [ <shellvariable> ]

      Beispiele:                                                        s1
          $ XX=asdf
          $ echo $XX
          asdf
          $ XX=asdf asdf
          sh: asdf: command not found
          $ XX="asdf asdf"
          $ echo $XX
          asdf asdf
          $
```





```
$ XX=Anfang
$ echo $XX
$ echo $XXswort

$ echo ${XX}swort
Anfangswort
$ (/bin/echo $XX)
Anfang
$ cat echo_xx
#!/bin/sh
echo $XX
$ ./echo_xx

$ set | grep XX
XX=asdf
$ env | grep XX
$ export XX
$ env | grep XX
XX=asdf
$ ./echo_xx
asdf
XX=asdf
$ unset XX
$ echo $XX

$
```

**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:
                                           "\"  "`"  "$"
                       werden als normale Zeichen interpretiert.
```

**Beispiele:**

```
$ touch xx\*
$ touch xxx
$ ls
xx*    xxx
$ mv xx* yy
mv: Beim Verschieben mehrerer Dateien muß das letzte Argument
                                          ein Verzeichnis sein
$ ls "xx*"
xx*
$ ls 'xx*'
xx*
$
```





**Vordefinierte Shellvariable:**

```
$? - Returnwert des letzten Kommandos
      $ true ; echo $?
      0
      $false ; echo $?
      1

$$ - Prozeßnummer der aktuellen Shell
      $ echo $$
      1234
      $ touch xxx$$
      $ ls -lisa xxx*
      -rw-r--r--    1 bell  unixsoft   0 Okt 30 08:57 xxx1234
      $

$# - Zahl der Positionsparameter
      $ echo $#
      0
      $

$* - bezeichet  alle Parameter
      $ echo $*

$0,...,$9 - bedeutet die Parameter beim Aufruf eines Shellscripts
      $ echo $0
      sh
      $
```

```
einige Standard-Shell-Variable:

  PS1      - Primär-Promptstring
             (Standard: $ - normaler Nutzer, # - root)
               PS1="`pwd` $ "
               PS1="$USER@`hostname` `pwd` > "
               PS1="[\u@\h \W]\$"    - bei Linux wenn sh eine
                                       eingeschränkte bash
               \d  - Datum      \h - Hostname    \n - <CR><NL>
               \s  - Shell      \t - Zeit        \u - Nutzername
               \w  - Work.Dir.  \W - Basename WD \# - Nr. des Kommandos
  PS2      - Sekundär-Promptstring, wenn sich eine Kommandozeile über
             mehrere Zeilen erstreckt (Standard: > ).
  HOME     - Homedirectory

  PATH     - Pfad für ausführbare Kommandos

  MAIL   - Mailfolder

  TERM   - Terminaltype, wichtig für vi

  LOGNAME, USER  - Login-Name des Nutzers

  SHELL  - Name der aktuellen Shell

  TZ     - Timezone

  MANPATH - Suchpfade für man-Kommando
```





```
Expandieren von Dateinamen
--------------------------

    * - beliebige Zeichenfolge ( auch leer)

    ? - ein beliebiges Zeichen (nicht leer)

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

    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 .?

  bash: auf der Kommandozeile:
     Expandieren eines Filenamens mit der TAB-Taste  - sehr praktisch
```

```
Ein- und Ausgabe
----------------

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

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

fd> file       - Umlenkung des Ausgabekanals  fd  in das File   file
                      $ echo start von asdf 1> protokollfile

>&-            - Schließen Standardausgabe
                      $ ls >&-
                      ls: Schreibfehler: Ungültiger Dateideskriptor
                      $

fd>&-          - Schließen des Ausgabekanals  fd
                      $ ls 1>&-
                      ls: Schreibfehler: Ungültiger Dateideskriptor
                      $

< file         - Umlenkung Standardeingabe von  file
                      $ cat < eingabefile

fd<file        - Umlenkung des Eingabekanals fd von  file
                      $ cat 0< eingabefile
```





```
<&-            - Schließen Standardeingabe
                      $cat <&-
                      cat: -: Ungültiger Dateideskriptor

fd<&-          - Schließen des Eingabekanals  fd
                      $cat 0<&-
                      cat: -: Ungültiger Dateideskriptor

fd1>&fd0       - Umlenkung der Ausgabe des Kanals fd1 auf den eröffneten
                 Kanal fd0, in der crontab beliebt
                      $ dauerlaeufer 1> /dev/null 2>&1 &

>> file        - Umlenkung Standarausgabe mit Anfügen
                      $ echo Anfang des Scripts >> Protokollfile

fd>> file      - Umlenkung des Ausgabekanals  fd  mit Anfügen
                      $ echo Anfang des Scripts 1>> Protokollfile

<<ENDE         - Lesen aus Shellscript bis ENDE

                      $ SUMME=`bc <<EOF
                      1+3
                      EOF`
                      $ echo $SUMME
                      4
                      $                                           s12
                                                                  s13
```

```
read <variable> - Lesen einer den Wert von der Standardeingabe und
                  Wertzuweisung zur Variablen

                  #!/bin/sh
                  echo -n "Eingabe: "
                  read INPUT
                  echo $INPUT

`Kommando`    - Umlenkung der Standardausgabe in eine Zeichenkette

                   echo "Start des Scripts: `date`" >> protokoll
                                                                    s14,s15
```





```
Shell-Scripte
-------------

Shell-Script: File mit gültigen Shell-Kommandos

  Aufruf:  sh <Shell-Script-Name>
    Das Shell-Script-File muß kein EXEC-Bit haben. Es kann dann
    aber auch nur mittels "sh" aufgerufen werden. Wenn das
    EXEC-Bit gesetzt ist, kann das Script direkt aufgerufen werden.
    Achtung: Es wird dann immer mit der aktuellen Shell ausgeführt.
    Am Anfang eines Shell-Scriptes sollte deshalb immer die benutzte
    Shell als Spezialkommentar (Major-Number: #!/bin/sh) eingetragen
    sein.
                                                                  s16,s17

    Können Shell-Scripte mit Parameter umgehen?
      JA - erstmal die Parameter 1..9
           durch
               $1 .. $9
           addressierbar
                                                                      s18

    Was passiert bei mehr als 9 Parameter?

       Alle Parameter werden mittels "shift" um eine Position
       nach links verschoben. Wenn keine Parameter mehr vorhanden
       sind werden die Parameter auf "leer" gesetzt.
                                                                      s19
```

```
Kommandos - 2.Teil
------------------

   <Kommando> ::= <einfaches Kommando> |
                  "(" <Liste von Kommandos> ";" ")" |
                  "{" <Liste von Kommandos> ";" "}" |
                   <if-Kommando> | <case-Kommando> |
                   <while-Kommando> | <until-Kommando> |
                   <for-Kommando> | ...

   <if-Kommando>::= "if" <Liste von Kommandos>
                         "then" <liste von Kommandos>
                        {"elif" <Liste von Kommandos>
                            "then" <Liste von Kommandos> }
                        ["else" <Liste von Kommandos>]
                    "fi"
```

Die Kommandoliste nach "if" wird abgearbeitet. Der Returnwert des letzten abgearbeiteten Kommandos bestimmt die Verzweigungsbedingung. Ist der Wert gleich Null, wird die Kommandoliste nach dem "then" abgearbeitet. Ist der Wert ungleich Null, wird die Kommandoliste nach dem "else" abgearbeitet, falls diese vorhanden ist. Ist ein "elif" Abschnitt vorhaden, wird mit diesem verfahren, wie bei "if". Der Abschnitt wird anstelle von "else" abgearbeitet.
Beachte: Vor "then", "elif", "else", "fi" muß ein <NL> oder ein ";" als Trennzeichen stehen(werden als einfache Kommandos aufgefasst).

s20





```
    <case-Kommando>::= "case" <Wort> "in"
                           <Muster>")" <Kommandoliste> ;;
                          {<Muster>")" <Kommandoliste> ;; }
                       "esac"
```

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 (Fortsetzung nach "esac"). Es gelten die gleichen Regeln wie bei der Dateierweiterung ( "[..]", "*", "?").

s21

```
    <while-Kommando>::= "while" <Kommandoliste>
                        "do"
                             <Kommandoliste>
                        "done"
```

Die Kommandolist nach dem "while" wird abgearbeitet. Ist der Returnwert des letzten Kommandos 0 (True) wird die Kommandoliste nach dem "do" abgearbeitet. Danach wird die Kommandoliste nach dem "while" wieder abgearbeitet. Dies geschieht solange, wie der Returnwert des letzten Kommandos der Kommandoliste nach dem "while" gleich 0 (True) ist. Ist der Wert ungleich 0, wird das while-Kommando beendet (Fortsetzung nach dem "done"). Durch das Buildin-Kommando "break" kann das while-Kommando jederzeit beendet werden. Durch das Buildin-Kommando "continue" wird der nächste Schleifendurchlauf gestartet.

s22

```
<until-Kommando>::= "until" <Komandoliste>
                     "do"
                          <Kommandolist>
                     "done"
```

Die Kommandolist nach dem "until" wird abgearbeitet. Ist der Returnwert des letzten Kommandos ungleich 0 (False) wird die Kommandoliste nach dem "do" abgearbeitet. Danach wird die Kommandoliste nach dem "until" wieder abgearbeitet. Dies geschieht solange, wie der Returnwert des letzten Kommandos der Kommandoliste nach dem "until" ungleich 0 (False) ist. Ist der Wert gleich 0 (True), wird das until-Kommando beendet (Fortsetzung nach dem "done"). Durch das Buildin-Kommando "break" kann das until-Kommando jederzeit beendet werden. Durch das Buildin-Kommando "continue" wird der nächste Schleifendurchlauf gestartet.

s23

```
<for-Kommando>::= "for" <Laufvariable> [ "in" <wort> {<wort>} ]
                    "do"
                        <Kommandolist>
                    "done"
```

Die Laufvariable nimmt nacheinander die Werte aus der Wortliste an und mit jedem Wort werden die Kommandos der Kommandoliste abgearbeitet. Fehlt der "in"-Part, wird anstelle der Wordliste die Parameterliste des Shell-Scripts (aktuelle Shell-Funktion) benutzt. Durch das Buildin-Kommando "break" kann das for-Kommando jederzeit beendet werden. Durch das Buildin-Kommando "continue" wird der nächste Schleifendurchlauf gestartet.

s24





**interne Shell-Kommandos**
**-----------------------**

```
    <einfaches Kommando>::=   ....| <interne Shell-Kommando>
```

interne Shell-Kommanod - Kommando innerhalb der Shell realisiert.

```
#
     Kommentar
         # Das ist ein Kommentar bis Zeilenende

: {<Argumente>}
     Nullkommando
       wie Kommentar, aber ";" als Trennzeichen erlaubt, das ein
       weiteres Kommando folgt.
         : das Kommando ls folgt ; ls

. <Kommandodatei>
     einlesen von Kommandos aus dem File in der aktuellen Shell
```

s30

```
break [n]
     verlassen von Schleifenanweisungen (while, until, for).
       n gibt die Anzahl der zu verlassenden Schleigen an.
       Standard ist 1.

cd
     Definition des Working Directory (Current Directory)
       Nur für die aktuelle Shell und nachfolgende Kommandos gültig.
```

```
continue [n]
     Beenden von Schleifen in Schleifenanweisung (while, unitil, for)
       Es wird mit der Abarbeitung der Schleifen bedingung fortgesetzt.
       n gibt die Zahl der Schleifen an. Standard ist 1.
                                                                    s31
echo {<argument>}
     Ausgabe der Argumente auf die Standardausgabe

eval {<argument>}
     Interpretation der Argumente durch die Shell, anschließend wird das
     Resultat als Kommando abgearbeitet (1. Argument ist das Kommando)
     Achtung: Argumente werden zweimal durch die Shell interpretiert!!!!
           LSC=ls
           COUNT='|wc -l'
           $LSC $COUNT
           eval $LSC $COUNT
                                                                    s32
exec {<argumente>}
     Ausführen der Argumente als Kommando im aktuellen Shell-Prozeß.
       1. Argumente ist das Kommando. Die Shell wird beendet.
       Ohne Argumente werden nur die E/A-Umlenkungen für die aktuelle
       Shell übernommen.
                                                             s11, run-rdist
exit [<Rückkehrkode>]
     beenden der Shell mit einem Rückkehrkode
                                                                    S33
export {<argument>}
     Übergabe von Variablen der Shell an die Umgebung.
       Definition von Umgebungsvariable.
```





```
getopts <optstring> <name> {<argument>}
     Lesen der Aufruf-Argumenten eines Schellscripts
       <optstring> - {<Optionen mit Argumenten>":"|<Optionen ohne Argument>}
                      m:n:xy  -  Optionen -m und -n mit je einem Argument
                                 Optionen -x und -y ohne Argument
                                    also:  kommando -m abc -n def -x -y
       <name>      - Shellvariable, die die Option aufnimmet (ohne "-")
       <argument>  - Argumente, die getopts anstelle von $1, .. , $9
                     auswertet.
        Shellvariable: OPTARG   - enthält Argument, wenn vorhanden
                       OPTIND   - Anzahl der Argumente + 1
                                                                    s34
hash
     Ausgabe der Hash-Tabelle auf Standardausgabe.
       Hashtabelle enthält die Pfade aller ausgeführten Kommandos.


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

pwd
     Ausgabe des Workingdirectories

read {<variable>}
     Einlesen von Werten für Variable von der Standardeingabe.
       Sollen mehrere Variable eingelesen werden, müssen die zugehörigen
       Eingabewerte in einer Zeile stehen.
                                                                     s35
```

```
readonly {<Shellvariable>}
     Shellvariable als "read only" kennzeichnen.
                                                                             s36
return  [<Rückkehrkode>]
     Rückkehr aus einer Schellfunktion mit Rückkehrkode.

set [optionen [argumente]]
     Setzen von Optionen und Argumenten für die aktuelle Shell.
       Damit können nachträglich Optionen gesetzt werden.

shift
     Verschieben von Parametern um eins nach links.
       Damit ist der Zugriff auf mehr als 9 Parameter möglich.

times
     Anzeigen der verbrauchte CPU-Zeit der aktuellen Shell.

trap [<Kommando>| "" ] [<Signalnummern>]
     Definition von Signalbehandlungsroutinen (s.u.)

type {<Kommando>}
     Anzeigen welches Kommando ausgeführt wird.
           $ type ls echo
           ls is hashed (/bin/ls)
           echo is a shell builtin
           $

ulimit [-SHacdflmnpstuv] <limit>
     Anzeigen der Nutzerspezifischen Systemgrößen
           $ ulimit -a
```





```
umask [<Mask>]
     Setzen der Filecreationmask.
       Gesperrte Zugriffsrechte werden gesetzt.
           $ umask 022
           $ umask 077

unset <Shellvariable>
     Löschen von Variablen.
       Die Variable ist danach undefiniert.
           $ echo $HOME
           /home/bell
           $ unset HOME
           $ echo $HOME

           $

wait [<Prozeßnummer>]
     Warten auf das Ende einer Subshell
         $ sleep 1000 &
         [1] 8539
         $ wait
         ^C
         $ wait $!
         ^C
         $ wait  8539
         ^C
         $ wait 1234
         sh: wait: pid 1234 is not a child of this shell
         $
```

```
Externe "Shell"-Kommandos
-------------------------

test  <Ausdruck>

     Kommando zum Testen von Ausdrücken  (builtin-Kommando und /usr/bin/test)
        Wenn der Ausdruck TRUE ist gibt das Kommano 0 als Retrunwert sonst 1.
        Verkürzte Schreibweise in Shellscripten:    "[ "<Ausdruck>" ]"

          logische Operationen
                "("<Ausdruck>")"  - Klammerung
                "!" <Ausdruck>  - verneinung
                <Ausdruck> "-a" <Ausdruck> - und-Bedingung
                <Ausdruck> "-o" <Ausdruck> - oder-Bedingung

           Vergleiche
                     Ausdruck                        True wenn
                ["-n"] <String>           - String-Länge > 0
                <String> "=" <String>     - Gleichheit der Strings
                <String> "!=" <String>    - Ungleichheit der Strings
                <Integer> "-eq" <Integer> - Gleichheit der Integer-Zahlen
                <Integer> "-ne" <Integer> - Ungleichheit der Integer-Zahlen
                <Integer> "-ge" <Integer> - Größergleich der 1. Integer-Zahl
                <Integer> "-gt" <Integer> - Größer der 1. Integer-Zahl
                <Integer> "-le" <Integer> - Kleinergleich der 1. Integer-Zahl
                <Integer> "-le" <Integer> - Kleiner der 1. Integer-Zahl
```





```
           Eigenschaften von Files
                    Ausdruck                        True wenn
                  -b <Filename>           -  File ist Blockdevice
                  -c <Filename>           -  File ist Characterdevice
                  -d <Filename>           -  File ist Dirctory
                  -e <Filename>           -  File existiert
                  -f <Filename>           -  File ist ein reguläres File
                  -p <Filename>           -  File ist eine Pipe
                  -r <Filename>           -  File ist lesbar
                  -w <Filename>           -  File ist schreibbar
                  -s <Filename>           -  File ist nicht leer
                  -x <Filename>           -  File ist ausführbar
                                u.s.w.


expr <Ausdruck>

     Programm zum Berechnen von arithmetischen un logischen Ausdrücken
       Der Wert des Ausdruckes wird auf Standardausgabe ausgegeben.

       <Ausdruck>::=<Ausdruck> "|" <Ausdruck> | <Ausdruck> "&" <Ausdruck> |
                    <Ausdruck> "<" <Ausdruck> | <Ausdruck> "<=" <Ausdruck> |
                    <Ausdruck> ">" <Ausdruck> | <Ausdruck> ">=" <Ausdruck> |
                    <Ausdruck> "=" <Ausdruck> | <Ausdruck> "!=" <Ausdruck> |
                    <numerischer Ausdruck>
```

```
        <numerischer Ausdruck>::= <Zahl> |
                                <numerischer Ausdruck> "+" <numerischer Ausdruck> |
                                <numerischer Ausdruck> "-" <numerischer Ausdruck> |
                                <numerischer Ausdruck> "*" <numerischer Ausdruck> |
                                <numerischer Ausdruck> "/" <numerischer Ausdruck>


env [-] [-i] {-u <Name>} [--ignore-environment] {--unset=<Name>]
                         {<Name>=<String>} <Kommando>

     Ausführen des Kommandos <Kommando> in einer modifizierten Umgebung
       "-", "-i"        - Löschen aller Umgebungsvariablen
       "-u", "--unset" - Löschen einer einzelner Umbgebungsvariablen
       <Name>=          - Setzen einer einzelnen Umgebungsvariablen
     ohne Parameter werden nur die aktuellen Umgebungsvariablen ausgegeben.


printenv [--help] [--version] {<Variable>}

     Ausgabe der spezifizierten Umgebungsvariablen.
       Wird keine Umgebungsvariable spezifiziert, werden alle ausgegeben.
          $ printenv PATH
          /usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:.
          $
```





```
Kommandozeile und Initialisierung von sh
----------------------------------------

Aufrufsyntax der Shell:

   sh [-aefhiknrtuvx] [<Shellscript>]
   sh [-aefhiknrtuvx] -c <Kommando-Liste>

      -c   Die nachfolgende Kommando-Liste ausführen

Initialisierung:

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

     1. /etc/profile
     2. /etc/profile.d/*.sh
     3. $HOME/.profile

Dadurch werden alle Umgebungsvariablen gesetzt. Wenn die Shell nicht als
login-Shell läuft, werden die Umgebungsvariablen aus der Umgebung des
rufenden Prozesses benutzt.
```