```
HOWTO zur UNIX-Kommandozeile

(C) 2007-2017 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
              OSTC Open Source Training and Consulting GmbH
              http://www.ostc.de

$Id: unix-command-line-HOWTO.txt,v 1.12 2017/09/11 22:59:53 tsbirn Exp $

Dieses Dokument beschreibt Aufbau von Kommandozeilen und die Suche nach
Kommandos unter UNIX.

_____________________________________________________________________________

INHALTSVERZEICHNIS

1) Allgemeine Syntax
2) Optionen
3) Spezielle Optionen
4) Line Continuation
5) Internal Field Separator (IFS)
6) Kommandosuche
6.1) Unterschied zu Windows
6.2) Simulation der Kommandosuche
6.3) Konfiguration der Kommandosuche
7) Kommando-Substitution
8) Shell-Aliase
9) Shell-Funktionen
_____________________________________________________________________________


1) Allgemeine Syntax
--------------------
Die allgemeine Syntax von UNIX-Kommandoaufrufen lautet (die einzelnen
Elemente CMD, OPTION und ARGUMENT werden durch ein oder mehr
"Whitespaces"=Leerzeichen/Tabulator/Zeilenvorschub getrennt):

   CMD [OPTION...] [--] [ARGUMENT...]

Eine Kommandozeile wird erst durch Drücken der <RETURN>-Taste der Shell
zur Interpretation und Ausführung übergeben, bis dahin kann sie beliebig
geändert werden. Die Shell zerlegt anschließend den auf der Kommandozeile
eingegebenen Text in Token/Worte (anhand der "Whitespaces") und interpretiert
eine Reihe von Metazeichen/Sonderzeichen (z.B. * ? & >). Sollen Trennzeichen
bzw. Metazeichen als normale Zeichen behandelt werden, so sind sie durch
"Quotieren" (per \, "..." oder '...') vor der Shell zu schützen.

Das 1. Wort der Kommandzeile hat eine Sonderbedeutung, es wird als
auszuführendes Kommando CMD interpretiert. Alle direkt darauf folgenden
Worte mit "-" als 1. Zeichen sind Optionen, sie beeinflussen das Verhalten
des Kommandos. Fehlt die Angabe von Optionen, so zeigt jedes Kommando ein
Standardverhalten:

  ls                            # Standard: Inhalt akt. Verz. anzeigen


2) Optionen
-----------
Optionen sind entweder EINZELZEICHEN bzw. bei GNU-Kommandos und vielen anderen
Kommandos auch WORTE. Jeder Optionszeichen/jedes Optionswort steht für ein
englisches Wort, das seine Bedeutung beschreibt (Merkhilfe!). Zu einer Option
kann ein weiterer Parameter nötig sein, der direkt dahinter anzugeben ist (ohne
"-" davor, egal ob ohne/mit Leerzeichen von Option getrennt):

  ls -l                         # Eine Option
  ls -w10                       # Eine Option mit Parameter "10"
  ls -w 10                      # Eine Option mit Parameter "10"
  ls -l -R -t                   # Drei Einzelzeichen-Optionen einzeln

Einzelzeichen-Optionen ohne Argumente können einzeln für sich (jede mit "-"
davor) oder gemeinsam ohne Leerzeichen aneinandergehängt (nur ein "-" am Anfang
notwendig) angegeben werden:

  ls -lRt                       # Drei Einzelzeichen-Optionen zusammengefasst

Optionsworte sind durch "--" einzuleiten, sie können nicht hinter einem "--"
zusammengefasst werden:

  ls --long --recursive --time  # Drei Optionsworte

Das 1. Wort einer Kommandozeile, das nicht mit einem "-" beginnt (und nicht
Parameter einer Option ist), leitet die Liste der sonstigen Kommandoargumente
ein. Typischerweise sind das vom Kommando zu verarbeitenden Dateinamen oder
```

```
Texte. Danach noch vorhandene Argumente mit "-" als 1. Zeichen werden (meist)
nicht mehr als Optionen interpretiert (bei Linux evtl. doch).

Die meisten Optionen verändern nur das Verhalten eines Kommandos (Flag), manche
Optionen benötigen zusätzlich die Angabe eines Parameters (direkt angehängt
oder durch Leerzeichen oder "=" getrennt):

  -t :           oder    -t:         # Trennzeichen angeben (terminator)
  -f DATEI       oder    -fDATEI     # Dateinamen angeben (file)
  --rows 123     oder --rows=123     # Anzahl Zeilen (rows)


3) Spezielle Optionen
---------------------
Die Liste der Optionen kann auch explizit durch "--" beendet werden, alle
danach aufgeführten Argumente (auch wenn sie als 1. Zeichen "-" haben)
werden NICHT als Optionen interpretiert. Typischerweise sind das die Namen
der vom Kommando zu verarbeitenden Dateien.

  ls -l -R -t *.c *.h                # Ende der Optionsliste bei "*.c"
  ls -l -- -*                        # Explizites Ende der Optionsliste
  touch -- -i                        # Datei "-i" anlegen
  rm -- -i                           # Datei "-i" löschen

Ein "-" für sich allein als Argument steht für die Standard-Ein/Ausgabe
(hilfreich falls ein Kommando keine Dateiumlenkung kann, z.B. "tar"):

  gunzip ARCHIVE.tgz | tar tvf -


4) Line Continuation
--------------------
Steht ein "\" am Zeilenende (KEIN Zeichen mehr danach!), dann wird die
Kommandozeile auf der nächsten physikalischen Zeile fortgesetzt (der
Zeilenvorschub wird "ausmaskiert"):

  ls -l \                            # Entspricht "ls -l -R -t"
     -R \
     -t


5) Internal Field Separator (IFS)
---------------------------------
Definiert, welche Zeichen als Trennzeichen für die unter 1) beschriebene
Zerlegung in Token/Worte verwendet werden. Die Standardbelegung ist
"<LEERZEICHEN><TABULATOR><NEWLINE>" (3 Zeichen). Das 1. Zeichen hat noch eine
weitere Sonderbedeutung, indem es an einigen Stellen bei der Ausgabe zur
Trennung von Parametern verwendet wird (z.B. bei $* zwischen den Parametern).
Weiterhin werden die Zeichen in IFS als Worttrennzeichen beim Einlesen einer
Zeile mit "read" verwendet.

  echo -n "$IFS" | od -c                       # -> SPACE \t \n

  echo "eins:zwei:drei" > /tmp/zeile
  IFS=":" read VAR1 VAR2 VAR3 < /tmp/zeile     # VAR1=eins VAR2=zwei VAR3=drei


6) Kommandosuche
----------------
Zur Ausführung von auf der Kommandozeile angegebenen Kommandos CMD sucht die
Shell in einer bestimmten Reihenfolge an unterschiedlichen Stellen im
Dateisystem nach einem passenden (ausführbaren!) Programm. Wird eines gefunden,
so bricht sie die Suche an dieser Stelle ab und führt das gefundene Programm
aus. Einige Kommandos sind aus Gründen der Effizienz oder der prinzipiellen
Notwendigkeit direkt in die Shell eingebaut. Sie heißen "Built-ins" und werden
grundsätzlich zuerst gefunden:

  +---+-------------------------------------------------------------------+
  |   | Kommandoart                                                       |
  +---+-------------------------------------------------------------------+
  | 1 | Built-in CMD/Keyword (z.B. cd, pwd, exit, ...)                    |
  | 2 | Alias CMD            (z.B. alias ll="ls -lF")                     |
  | 3 | Funktion CMD         (z.B. ll() { /bin/ls -lF $*; })              |
  +---+-------------------------------------------------------------------+
  | 4 | Von links nach rechts in allen im Suchpfad PATH angegebenen       |
  |   | Verzeichnissen (durch ":" getrennt) nach einer Datei namens CMD   |
  |   | suchen, diese muss zusätzlich ausführbar (+ lesbar bei Skript) sein, |
  |   | z.B. ergibt PATH="/bin:/usr/bin:." folgende Aufrufversuche:       |
  |   |   4a - Programm /bin/CMD                                          |
  |   |   4b - Programm /usr/bin/CMD                                      |
  |   |   4c - Programm ./CMD                                             |
  +---+-------------------------------------------------------------------+
```

```
  | 5 | Nicht gefunden => Meldung: "sh: CMD: command not found"              |
  +---+-----------------------------------------------------------------------+
```

Die obige Kommandosuche kann vollständig umgangen werden, indem ein Kommando mit relativem oder absoluten Pfad angegeben wird (d.h. enthält mindestens einen "/"). Auf diese Art ist es auch möglich, über den aktuellen Suchpfad nicht erreichbare Kommandos aufzurufen (z.B. als "root" im aktuellen Verzeichnis), allerdings immer unter der Voraussetzung, dass die Ausführungsrechte des Kommandos dies zulassen:

```
  ./CMD          # CMD im aktuellen Verzeichnis aufrufen
  /usr/bin/ls    # "ls" aus Verzeichnis "/usr/bin" aufrufen
```

Die Suche nach Aliasen und Funktionen kann durch einen Backslash vor dem Kommando verhindert werden, es wird dann das Original-Kommando "ls" gemäß Pfadsuche aufrufen, nicht ein gleichnamiger Alias oder eine gleichnamige Funktion.

```
  \ls
```

6.1) Unterschied zu Windows

Im Unterschied zu Windows wird standardmäßig NICHT im aktuellen Verzeichnis (Arbeitsverzeichnis) nach einem zum Kommando passenden ausführbaren Programm gesucht (bei Windows wird dort ZUERST gesucht). Weiterhin wird nach einem Programm mit EXAKT dem angegebenen Kommandonamen gesucht, bei Windows kann die Endung (z.B. ".exe", ".com", ".bat", ...) weggelassen werden.

Soll doch im aktuellen Verzeichnis gesucht werden, so ist in den Suchpfad PATH das Verzeichnis "." mit aufzunehmen (steht für das aktuelle Verzeichnis). Zu entscheiden ist dann, an welcher Stelle: vorne, in der Mitte, am Ende. Abhängig davon werden passende Programme im aktuellen Verzeichnis VOR allen UNIX-Kommandos, TEILWEISE VOR und TEILWEISE NACH den UNIX-Kommandos oder NACH allen UNIX-Kommandos gefunden.

Ein einzelner ":" am Anfang oder am Ende oder zwei direkt aufeinanderfolgende "::" in der Mitte von PATH stehen ebenfalls für das aktuelle Verzeichnis.

Aus Sicherheitsgründen sollte das aktuelle Verzeichnis AUF KEINEN FALL im Suchpfad der "root" stehen (Schutz vor Trojanern). Ebenso sollte es bei normalen Benutzer HÖCHSTENS am Ende des Suchpfades stehen, damit nicht versehentlich statt einens UNIX-Kommandos ein gleichnamiges anderes Programm aufgerufen werden kann.

```
  PATH=/usr/sbin:/bin:/usr/bin:/sbin:/usr/X11R6/bin    # root
  PATH=/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:.   # Normaler Benutzer
```

Nimmt man "." in den Suchpfad auf, dann kann man andere Benutzer ohne deren Wissen dazu verleiten, von einem selbst geschriebene Kommandos ("Trojaner") auszuführen, indem man sie in das Verzeichnis mit dem Kommando wechseln lässt und es dort ausführen lässt. Typische Kandidaten dafür sind "ls" oder Tippfehler bekannter Kommandos.

Man verzichtet also besser auf die Aufnahme von "." im Suchpfad PATH, auch wenn Windows-Kenner das als sehr umständlich empfinden. Falls doch einmal ein Kommando im aktuellen Verzeichnis aufzurufen ist, dann muss man das immer ganz bewußt in der Form "./CMD" durchführen.

6.2) Simulation der Kommandosuche

Die obige Suche nach einem passenden Programm kann mit "type" (bash), "which" (csh, tcsh) oder "whence" (ksh) simuliert und die erste Fundstelle angezeigt werden. Mit dem Schalter "-a" (all) kann man ALLE passenden Kommandos anzeigen, ausgeführt wird aber immer das ZUERST gefundene.

```
  type ls
  type -a ls
  which ls
  whence ls
```

Einige Shells merken sich den Pfad zu einem Kommando nach dem ersten Aufruf in einem "Hash", um es bei erneuten Aufrufen schneller zu finden. Der Befehl "type" zeigt dies z.B. dann folgendermaßen an:

```
  type ls    # =>  /bin/ls
  ls         # Kommando 1x aufrufen
  type ls    # =>  ls ist hashed (/bin/ls)
```

Mit "hash" und "rehash" (bzw. "hash -r" = remove oder "hash -d" = delete) kann diese Hash-Liste angezeigt und manipuliert bzw. gelöscht werden.

```
6.3) Konfiguration der Kommandosuche
------------------------------------
Mit Hilfe des Suchpfades PATH, Aliasen und Funktionen ist ein UNIX-System
gut konfigurierbar und kann auf die Bedürfnisse aller oder einzelner
Anwender zugeschnitten werden. In den Konfigurationsdateien "/etc/profile"
und "~/.profile" (bzw. "~/.bashrc", "~/.login" oder "~/.tcshrc") wird dazu
einfach der Suchpfad ergänzt/verändert und Aliase oder Funktionen definiert
(die gleichnamige Systemkommandos auch überdecken dürfen).

Z.B. kann ein Anwender ein eigenes Kommando "ls" als Alias definieren, das
zuerst gefunden und ausgeführt wird, wenn er "ls" eingibt. Ruft dieses Kommando
allerdings selbst das Original-"ls" auf, so muss es dies (meist) über einen
absoluten Pfad tun, um keinen rekursiven Aufruf seiner selbst zu bewirken:

  alias ls="/bin/ls -F"   # bash, ksh
  alias ls "/bin/ls -F"   # csh, tcsh

Ein stark konfiguriertes System führt allerdings mit der Zeit dazu, dass
der Anwender sich an die Bequemlichkeiten gewöhnt und die Originalbefehle
vergisst. Muss er auf ein anderes UNIX-System ohne diese ganzen Einstellungen
wechseln, so ist er eventuell erst einmal völlig aufgeschmissen, bis seine
Umgebung wieder analog eingerichtet ist.

Man sollte auch keinesfalls den Fehler begehen, MS-DOS-Kommandos wie "dir",
"md", "rd", "ren", "del",... in ihre UNIX-Äquivalente umzusetzen. Man wird
sonst nie die Original-UNIX-Befehle lernen. Allenfalls sollte man mit "echo"
Meldungen ausgeben, wie der entsprechende UNIX-Befehl wirklich heißt, z.B.

  alias dir="echo "error: use 'ls -l'"   # dir -> Meldung "error: use 'ls -l'"

SuSE-Systeme haben sich sehr lange Zeit so verhalten.


7) Kommando-Substitution
------------------------
Ein in einer Kommandozeile innerhalb von `...` stehendes Kommando CMD wird
ZUERST ausgeführt, seine GESAMTE AUSGABE wird an dieser Stelle eingesetzt und
erst dann wird die gesamte Kommandozeile ausgeführt:

  echo "Heute ist `date`"

ACHTUNG: Das Zeichen ` (backquote, backtick) nicht mit dem Quotierungszeichen
' (quote) verwechseln. In der Bash/Korn-Shell gibt daher es die alternative
Schreibweise $(CMD), die an den Zugriff auf den Wert einer Variablen per
$VAR angelehnt ist. Diese Schreibweise ist erheblich besser lesbar als die
klassische Variante `CMD`:

  echo "Heute ist $(date)"

Mit dem folgendem Kommando können zum Beispiel alle C-Dateien im aktuellen
Verzeichnis, die den Text "error" enthalten, herausgesucht und der Editor
"vi" für sie aufgerufen werden:

  vi  `grep -l error *.c`        # -l=list of filenames
  vi $(grep -l error *.c)        # -l=list of filenames

Oder an alle eingeloggten Benutzer die Datei "brief.txt" als Mail schicken:

  mail  `who | cut -d" " -f1 | sort | uniq` < brief.txt
  mail $(who | cut -d" " -f1 | sort | uniq) < brief.txt

Eine typische Anwendung der Kommando-Substitution ist auch das Initialisieren
von Variablen mit dem Ergebnis eines Kommandos:

  CFILES=`ls *.c`
   DATUM=$(date +%d.%m.%Y)
    ZEIT=`date +%H:%M`
 USERCNT=`who | wc -l`


8) Shell-Aliase
---------------
Aliase stellen eine Kommando-Abkürzung dar, sie führen zu einer Ersetzung
des ersten Wortes auf der Kommandozeile durch eine beliebige Folge von
Zeichen. Diese Ersetzung wird durchgeführt, BEVOR das eigentliche Kommando
ausgeführt wird. Alle anderen Argumente der Kommandozeile bleiben so stehen,
wie sie eingegeben wurden. Die Kommandos zum Verwalten der Aliase lauten:

  alias ll='/bin/ls -lF'    # Alias "ll" definieren (Rekursion vermeiden!)
    ll *.c *.h              #   => wird zu "/bin/ls -lf *.c *.h" expandiert
```

(C) 2017 OSTC GmbH (http://www.ostc.de)


```
  alias                     # Alle Aliase anzeigen
  alias ll                  # Alias "ll" anzeigen
  unalias ll                # Alias "ll" löschen

ACHTUNG: In der "csh" und in der "tcsh" das "=" weglassen.

  alias ll "/bin/ls -lF"

Alias sind gegenüber Shell-Funktionen eingeschränkt, z.B. können ihre Argumente
nicht vom Alias selbst umsortiert und verändert werden. Bei Shell-Funktionen
ist dies möglich.


9) Shell-Funktionen
-------------------
Shell-Funktionen sind eine Liste von Kommandos, die unter einem
(Funktions)Namen zusammengefaßt werden. Einer Funktion können beim Aufruf
Argumente übergeben werden (analog einem Shell-Skript) und sie kann einen
Exit-Code zurückgeben.

Mit ihrer Hilfe lassen sich z.B. Skripte strukturieren (zusammengehörige
Code-Teile am Skriptanfang in Funktionen verpacken und am Skriptende nur noch
die Funktionen aufrufen). Die Kommandos zum Verwalten der Funktionen lauten:

  ll() {                    # Funktion "ll" definieren (Rekursion vermeiden!)
      /bin/ls -lF           #
  }                         #
  ll() { /bin/ls -lF; }     # Analog in einer Zeile (; + Leerzeichen nötig!)

  typeset -f                # Alle Funktionen anzeigen
  typeset -f ll             # Funktion "ll" anzeigen

  unset -f ll               # Funktion "ll" löschen

ACHTUNG: In der "csh" und der "tcsh" gibt es keine Funktionen!
```