BASH

- Snippets
  - Zufallszahlen
  - Prüfen ob eine Variable einen Integerwert enthält
  - mit Unicode-Zeichen malen
  - Datum/Zeit-Berechnung
  - Anzahl verschiedener Dateitypen zählen

- Scripte
  - check_interfaces.sh
  - check_ldap_response.sh
  - pingall.sh
  - check_vips.sh
  - count-packets.sh
  - cisco.sh
  - vpn-keepalive.sh
  - ssh-break.sh
  - signature.sh
  - wake.sh
  - do-cmd.sh

- freeze und unfreeze der Eingabe
- Heredoc Beispiele und Tipps (Here Document)
- bash-completion
- Programm- oder Funktionsrückgabe auswerten
- BASH-History mit Datum
- meine .bashrc und .profile

Snippets

Snippets

Zufallszahlen

Zufallszahlen aus /dev/urandom mit einem bestimmten Bereich erzeugen

```
# Zahlen von 1 - 10
RND=`od -vAn -N1 -tu1 < /dev/urandom` && echo $(( $RND % 10 + 1 ))
```

Snippets

Prüfen ob eine Variable einen Integerwert enthält

```
#!/bin/bash

var="test"
#var=5

if [[ $var =~ ^-?[0-9]+$ ]]; then
  echo "$var ist int"
else
  echo "$var ist kein int"
fi
```

hat diese Ausgabe:

```
$ bash -x test
+ var=5
+ [[ 5 =~ ^-?[0-9]+$ ]]
+ echo '5 ist int'
5 ist int

$ bash -x test
+ var=test
+ [[ test =~ ^-?[0-9]+$ ]]
+ echo 'test ist kein int'
test ist kein int
```

Snippets

mit Unicode-Zeichen malen

Die Codetabelle gibts hier: http://unicode-table.com/de/

Verwenden läßt sich das dann wie folgt:

```
$ echo -e '\u00A9'
©
$ printf '\u00A9\n'
©
```

Damit lassen sich dann z.B. auch Spielfelder, Dialogboxen und ähnliches zeichnen (wobei für Dialoge das Tool „dialog“ besser geeignet wäre).

Snippets

Datum/Zeit-Berechnung

Datum und Zeit-Berechnung und -Manipulation mit BASH-Boardmitteln ist leider sehr mühselig. Das GNU date-Kommando kann uns hier sehr helfen:

Bei allen Beispielen gilt:

`DATUM="2009-02-25 10:30:38"`

Ein Datum in UNIX-Timestamp umwandeln:

`STAMP=`date --utc --date "$DATUM" +%s``

Und wieder zurück:

`date --utc --date "1970-01-01 $STAMP sec" "+%Y-%m-%d %T"`

Sekunden/Minuten/Stunden/Tage zwischen zwei Daten ausgeben:

```
dateDiff (){
    case $1 in
        -s)   sec=1;      shift;;
        -m)   sec=60;     shift;;
        -h)   sec=3600;   shift;;
        -d)   sec=86400;  shift;;
        *)    sec=86400;;
    esac
    dte1=$(date2stamp $1)
    dte2=$(date2stamp $2)
    diffSec=$((dte2-dte1))
    if ((diffSec < 0)); then abs=-1; else abs=1; fi
    echo $((diffSec/sec*abs))
}

# Beispiel:
# -s in sec. | -m in min. | -h in hours  | -d in days (default)
 dateDiff -s "2006-10-01" "2006-10-32"
 dateDiff -m "2006-10-01" "2006-10-32"
 dateDiff -h "2006-10-01" "2006-10-32"
 dateDiff -d "2006-10-01" "2006-10-32"
 dateDiff  "2006-10-01" "2006-10-32"
```

```
# number of seconds between two times
 dateDiff -s "17:55" "23:15:07"
 dateDiff -m "17:55" "23:15:07"
 dateDiff -h "17:55" "23:15:07"

# number of minutes from now until the end of the year
 dateDiff -m "now" "2006-12-31 24:00:00 CEST"
```

weitere Standardfeatures von date, die nicht sehr ausführtlich dokumentiert sind:

```
# add 2 days, one hour and 5 sec to any date
date --date "$DATUM  2 days 1 hour 5 sec"

# substract from any date
date --date "$DATUM 3 days 5 hours 10 sec ago"
date --date "$DATUM -3 days -5 hours -10 sec"

# or any mix of +/-. What will be the date in 3 months less 5 days
date --date "now +3 months -5 days"

# time conversions into ISO-8601 format (RFC-3339 internet recommended format)
date --date "sun oct 1 5:45:02PM" +%FT%T%z
date --iso-8601=seconds --date "sun oct 1 5:45:02PM"
date --iso-8601=minutes

# time conversions into RFC-822 format
date --rfc-822 --date "sun oct 1 5:45:02PM"
```

Snippets

Anzahl verschiedener Dateitypen zählen

manchmal sehr praktisch:

```
/tmp$ for i in `find -name "*.*" | sed 's/.*\.\(.*\)$/\1/' | grep . | sort -uf`; do echo "$i: `find -name \"*.$i\" | wc -l`"; done
pdf: 11
pub: 1
txt: 3
```

Scripte

Scripte

check_interfaces.sh

```
#!/bin/sh

# checkt ob alle Interface up sind und Link haben und liefert einen fuer Nagios gueltigen Return-Wert

CARDS="dev0 eth1 eth2 eth3"

DEBUG=0
MESG=""
INTOK="OK:"

for card in $CARDS
do
        if [ $DEBUG -gt 0 ]
        then
                echo -n "Testing $card "
        fi
        ret=$(ethtool -t $card online > /dev/null 2>&1 ; echo $?)
        if [ $ret -gt 0 ]
        then
                echo "CRITICAL: Interface $card failed!"
                exit 2
        else
                INTOK="$INTOK $card"
        fi
done
echo "$INTOK operating normally"
exit 0
```

Scripte

check_ldap_response.sh

zuerst muss mit dem Parameter -g ein Diff-File erzeugt werden, dann können mit dem Parameter -f die Antworten der LDAP-Server mit dem File verglichen werden.

```
#!/bin/sh

# 2005 Oliver Voelker <code@magenbrot.net>
# checkt die LDAP-Server auf korrekte Antworten und Syncronisation

# usage:
# check_ldap_response.sh -H <host> [-p <port>] -b <base_dn> -D <binddn> -P <bindpassword> -f
<checkfile> -v
#
# generate check-file:
# check_ldap_response.sh -H <host> [-p <port>] -b <base_dn> -D <binddn> -P <bindpassword> -g
<checkfile> -v

LDAPSEARCH=`which ldapsearch`

PORT="389"
LDAPLOOKUP="uid=testuser"
TMP=/tmp/ldap.$$
DIFF=$(dirname $0)/ldap-checkfile

while getopts "H:p:b:D:P:g:f:v" flag
do
  #echo "$flag" $OPTIND $OPTARG
  case "$flag" in
    H) HOST=$OPTARG
       ;;
    p) PORT=$OPTARG
       ;;
    b) BASE=$OPTARG
       ;;
    D) BINDDN=$OPTARG
       ;;
    P) BINDPW=$OPTARG
```

```
      ;;
    g) DIFF=$(dirname $0)/ldap-$OPTARG
      echo "lege Vergleichsfile $DIFF an..."
      $LDAPSEARCH -h $HOST -p $PORT -b $BASE -D $BINDDN $LDAPLOOKUP -x -w $BINDPW > $DIFF
      exit 0
      ;;
    f) DIFF=$(dirname $0)/ldap-$OPTARG
      ;;
    v) $LDAPSEARCH -h $HOST -p $PORT -b $BASE -D $BINDDN $LDAPLOOKUP -x -w $BINDPW > $TMP
      diff -y $TMP $DIFF
      ;;
    *) echo "Ungueltiges Kommandozeilenargument"
      exit 1
      ;;
  esac
done

if [ -z $HOST ] || [ -z $BASE ] || [ -z $BINDDN ] || [ -z $BINDPW ]; then
  echo "Fehlendes Argument!"
  echo "usage:"
  echo "check_ldap_response.sh -H <host> [-p <port>] -b <base_dn> -D <binddn> -P <bindpassword> -f
<checkfile>"
  echo "generate checkfile:"
  echo "check_ldap_response.sh -H <host> [-p <port>] -b <base_dn> -D <binddn> -P <bindpassword> -g
<checkfile>"
  exit 2
fi

if [ -z $LDAPSEARCH ]; then
  echo "ldapsearch (openldap-clients) nicht gefunden, bitte installieren."
  exit 2
fi

if [ ! -r $DIFF ]; then
  echo "Vergleichsfile $DIFF existiert nicht oder kann nicht gelesen werden, bitte mit parameter -g aufrufen!"
  exit 2
fi

$LDAPSEARCH -h $HOST -p $PORT -b $BASE -D $BINDDN $LDAPLOOKUP -x -w $BINDPW > $TMP
```

```
diff $TMP $DIFF > /dev/null 2>&1

if [ $? != "0" ]; then
  echo "LDAP Server nicht synchron bzw. Antwort fehlerhaft"
  exit 2
fi

rm $TMP
echo "LDAP Server ist synchron"
```

Scripte

pingall.sh

```
#!/bin/sh

SERVERLIST=serverlist.txt

if [ -e $SERVERLIST ]; then
  for SERVER in `cat $SERVERLIST`; do
    trap 'exit 0' 2
    ping -c2 $SERVER
    echo
  done
fi
```

Scripte

check_vips.sh

```
#!/bin/sh

# quick n very dirty check

# wieviele IPs sollte ich haben?
SHOULD=10

VIPs=`/sbin/ip a | grep "inet " | wc -l | awk -F" " '{ print $1 }'`

if [ ! "$VIPs" -eq "$SHOULD" ]; then
  echo "CRITICAL: owned IPs: $VIPs - should be: $SHOULD"
  exit 2
else
  echo "OK: owned IPs: $VIPs"
  exit 0
fi
```

Scripte

count-packets.sh

```
#!/bin/bash
#
# 2008 Oliver Voelker <info(at)ovtec.it>
#
# Pakete pro Sekunde fuer ein bestimmtes Interface ermitteln
#

if [ -z $1 ]; then
  echo "Usage: $0 <interface> [<time>]"
  echo "Default measure time is 10 seconds"
  exit 0
fi

if [ -z $2 ]; then
  TIME=10
else
  TIME=$2
fi

DEV=$1

RUN=`ifconfig $DEV | grep "X packets"`
if [ $? -ne 0 ]; then
  echo "Fehler!"
  exit 1
fi

RX1=`echo $RUN | cut -d" " -f2 | cut -d":" -f2`
TX1=`echo $RUN | cut -d" " -f8 | cut -d":" -f2`

#echo "RX-1: $RX1"
#echo "TX-1: $TX1"

sleep $TIME
```

```
RUN=`ifconfig $DEV | grep "X packets"`
if [ $? -ne 0 ]; then
  echo "Fehler!"
  exit 1
fi

RX2=`echo $RUN | cut -d" " -f2 | cut -d":" -f2`
TX2=`echo $RUN | cut -d" " -f8 | cut -d":" -f2`

#echo "RX-2: $RX2"
#echo "TX-2: $TX2"

RX=$(($RX2 - $RX1))
TX=$(($TX2 - $TX1))
RXps=$(($RX / $TIME))
TXps=$(($TX / $TIME))

echo "RX in $TIME seconds: $RX (RX per second: $RXps)"
echo "TX in $TIME seconds: $TX (TX per second: $TXps)"
```

Scripte

cisco.sh

Dieses Script ermöglicht einen automatischen Login auf Cisco-Switche inkl. automatischem „enable“ (sofern die Kennwörter gleich sind):

```
#!/bin/sh

# Passwort
PASS="passwort"
PROGDIR=`dirname $0`

rm -f $PROGDIR/login.expect
touch $PROGDIR/login.expect
chmod u+x $PROGDIR/login.expect

cat << EOF > $PROGDIR/login.expect
#!/usr/bin/expect
spawn telnet $1
expect {
 "Password:" {
  send "$PASS\r"
  exp_continue
 }
 "Name:" {
  send "enable\r"
  exp_continue
 }
 "#" {
  interact
 }
 "% Bad passwords" {
  send_user "invalid password or account\n"
  exit
 }
 timeout {
  send_user "connection timed out\n"
  exit
 }
```

```
  eof {
    send_user "connection to $host failed: $expect_out(buffer)"
    exit
  }
}
EOF

$PROGDIR/login.expect
rm -f $PROGDIR/login.expect
```

Aufruf mit „cisco.sh cat1.mydomain.de“ oder via IP.

Scripte

vpn-keepalive.sh

```
#!/bin/bash
# keepalive for ipsec
# 2007 Oliver Voelker <info(at)ovtec.it>

failmax=3 # beim dritten Fehler restarten
keepalive=30 # alle $keepalive Sekunden testen
maxage=120 # maximales Alter der Checkdatei in Sekunden
nextrestart=3600 # nach einem neustart erst wieder in X sekunden probieren

CHECKFILE="/tmp/keep-alive" # Dieses File muss minuetlich durch die VPN-Gegenseite erzeugt werden, z.B.
durch einen Cronjob: "ssh user@bla.de -C touch /tmp/keep-alive"
TMPFILE="/tmp/vpntest-$$"
ADMIN="admin@bla.de" # wird bei Stoerungen informiert
TUNNEL="ipsec-tunnel" # Tunnelname aus ipsec.conf

# do not edit anything beyond this point!

fail=0

MESSAGE=""

function tunnelrestart () {
  MESSAGE="Maxfail ($failmax) reached: restarting tunnel $TUNNEL (age of checkfile $DIFF seconds)"
  logger -p local2.info -t TUNNEL "$MESSAGE"
  /usr/sbin/ipsec auto --down $TUNNEL
  sleep 5
  /usr/sbin/ipsec auto --up $TUNNEL
  echo "tunnel $TUNNEL on `hostname -f` was restarted, because checkfile $CHECKFILE was too old. Please
check!" | mail -s "VPN-Problem on `hostname -f`!" $ADMIN
  touch $TMPFILE
  sleep 120
}

while (true); do
  CHECK=`stat -c"%Y" $CHECKFILE`
```

```
  NOW=`date +%s`
  DIFF=`echo $NOW - $CHECK | bc`

  if [ "$DIFF" -lt "$maxage" ]; then
    MESSAGE="Tunnel $TUNNEL OK (age of checkfile $DIFF seconds)"
    logger -p local2.info -t TUNNEL "$MESSAGE"
    fail=0
  else
    fail=`echo $fail+1|bc`
    MESSAGE="Tunnel $TUNNEL DOWN: $fail (age of checkfile $DIFF seconds with maxage of $maxage)"
    logger -p local2.info -t TUNNEL "$MESSAGE"
  fi

  if [ "$fail" -ge "$failmax" ] ; then

    if [ -f $TMPFILE ]; then
      ATMP=`stat -c"%Y" $TMPFILE`
      SSLR=`echo $NOW - $ATMP | bc` # seconds since last restart
      if [ "$SSLR" -ge "$nextrestart" ]; then
        rm -f $TMPFILE
        tunnelrestart
        fail=0
      else
        MESSAGE="Maxfail ($failmax) reached, but tunnel was already restarted $SSLR seconds ago. Only one
restart per $nextrestart seconds."
        logger -p local2.info -t TUNNEL "$MESSAGE"
        echo "tunnel $TUNNEL should have been restarted, but this already happened $SSLR seconds ago. Please
check!" | mail -s "BIG VPN-Problem on vpn.meinserver.de!" $ADMIN
      fi
    else
      tunnelrestart
    fi
  fi
  sleep $keepalive
done
```

Scripte

ssh-break.sh

```
#!/bin/bash

# scannt in diesem Beispiel den Netzwerkbereich 192.168.10.1 bis 192.168.20.255 via SSH mit dem User root und dem Passwort <enter password to check for>
# (<enter password to check for> durch gewuenschtes Passwort ersetzen!)

# von
A1="192."
B1="168."
C1="10."
D1="1"

# bis
A2="192."
B2="168."
C2="20."
D2="255"

# Passwort
PASS="<enter password to check for>"

PROGDIR=`dirname $0`

rm -f $PROGDIR/login.expect
touch $PROGDIR/login.expect
chmod u+x $PROGDIR/login.expect

for a in `seq $A1 $A2`
do
  for b in `seq $B1 $B2`
  do
    for c in `seq $C1 $C2`
    do
      for d in `seq $D1 $D2`
      do
```

```
        trap 'exit 0' 2
        echo "Teste $a.$b.$c.$d:"

cat << EOF > $PROGDIR/login.expect
#!/usr/bin/expect
spawn ssh -o PubkeyAuthentication=no -o ConnectTimeout=1 -o NumberOfPasswordPrompts=1
root@$a.$b.$c.$d "uptime"
expect {
  password: {
    sleep 1
    send "$PASS\r"
    exp_continue
  } "connecting (yes/no)?" {
    send "yes\r"
    exp_continue
  } incorrect {
    send_user "invalid password or account\n"
    exit
  } timeout {
    send_user "connection timed out\n"
    exit
  } eof {
    exit
  }
}
EOF

        $PROGDIR/login.expect
        echo -e
"##############################################################################
###########"
      done
    done
  done
done

rm -f $PROGDIR/login.expect
```

Scripte

signature.sh

```
#!/bin/sh


echo
cat ~/.signature
echo
echo "---------------------------------"
echo
/usr/bin/fortune
```

Scripte

wake.sh

Dieses Script holt einen Rechner/Server/HTPC aus dem Schlaf (sofern Wake On LAN (WOL) im BIOS aktiviert ist) und teilt einem mit, wenn die Maschine eine Netzwerkverbindung hat. Im Beispiel müsst Ihr einfach die Hostnamen und MAC-Adressen durch Eure eigenen ersetzen.

```
#!/bin/sh
#
# Wake On LAN Script
# 2010 Oliver Voelker <info@ovtec.it>
#

case "$1" in
  brot)
    echo "waking up brot"
    wakeonlan 00:1f:c6:0b:9e:3e
    while ( ! ping -c 1 brot.magenbrot.net 2>&1 >/dev/null ); do
      echo -n .
      sleep 1
    done
    ;;
  htpc|*)
    # htpc und default
    echo "waking up htpc"
    wakeonlan 00:01:2e:2b:91:6e
    while ( ! ping -c 1 htpc.magenbrot.net 2>&1 >/dev/null ); do
      echo -n .
      sleep 1
    done
    ;;
esac
echo -e "\nbehold! $1 has awoken!"
```

Scripte

do-cmd.sh

uralt... Heutzutage nimmt dafür lieber ansible oder ähnliches :)

```
#!/bin/bash
#
# 2006 Oliver Voelker <info(at)ovtec.it>
#

SERVERLIST=serverlist.txt
COMMAND=$*
REPORT=/tmp/report.$$
NOTIFY="scripts@magenbrot.net"
DATE=`date +%d.%m.%y-%H%M`

PROGDIR=`dirname $0`
cd $PROGDIR

if [ ! -f $SERVERLIST ]; then
  echo "$SERVERLIST nicht gefunden."
  exit 1
fi

if [ -z "$1" ]; then
  echo "Usage '$0 <remotecmd>'"
  echo "Executes <remotecmd> on all servers in $SERVERLIST"
  echo
  echo "-check-update           checks for updates on the servers"
  echo "-update                 installs available updates"
  echo
  echo "notifies are mailed to $NOTIFY"
  exit 0
fi

echo "Scriptlauf am `date +%c`" > $REPORT
echo "Scriptlauf am `date +%c` geloggt nach $REPORT"
echo "--------------------------------------------------------------------------------" >> $REPORT
```

```
if [ -e $SERVERLIST ]; then
  echo "--------------------------------------------------------------------------------"
  exec 5<&0
  cat $SERVERLIST | while read LINE; do
    trap 'exit 0' 2
    if [ "`echo $LINE | head -c1`" != "#" ] && [ "`echo $LINE | head -c1`" != "" ]; then
      #echo $LINE | awk -F" - " '{printf "Server: %s / Distri: %s / Updater: %s / Comment: %s\n", $1, $2, $3, $4}'
      SERVER=`echo $LINE | awk -F" - " '{print $1}'`
      DISTRI=`echo $LINE | awk -F" - " '{print $2}'`
      UPDATER=`echo $LINE | awk -F" - " '{print $3}'`
      COMMENT=`echo $LINE | awk -F" - " '{print $4}'`
      if [ -z "$SERVER" ] || [ -z "$DISTRI" ] || [ -z "$UPDATER" ] || [ -z "$COMMENT" ]; then
        echo "$SERVERLIST hat das falsche Format in Zeile $LINE"
        echo "$SERVERLIST hat das falsche Format in Zeile $LINE" >> $REPORT
        continue
      fi
      echo "Executing '$COMMAND' on $SERVER ($DISTRI / $COMMENT)..."
      echo "Executing '$COMMAND' on $SERVER ($DISTRI / $COMMENT)..." >> $REPORT
      case "$COMMAND" in
        "-check-update")
          case "$UPDATER" in
            "yum")
              ssh $SERVER "sudo su root -c 'yum check-update'" >> $REPORT 2>&1 <&5
              ;;
            "apt")
              ssh $SERVER "sudo su root -c 'apt-get -qq update'" >> $REPORT 2>&1 <&5
              ssh $SERVER "sudo su root -c 'apt-get -q -u -s upgrade'" >> $REPORT 2>&1 <&5
              ;;
            "up2date")
              ssh $SERVER "sudo su root -c 'up2date --dry-run'" >> $REPORT 2>&1 <&5
              ;;
            *)
              echo "kein gueltiger updater erkannt"
              ;;
          esac
          ;;
        "-update")
          case "$UPDATER" in
            "yum")
```

```
            ssh $SERVER "sudo su root -c 'yum -y update'" >> $REPORT 2>&1 <&5
            ;;
          "apt")
            ssh $SERVER "sudo su root -c 'apt-get -y upgrade'" >> $REPORT 2>&1 <&5
            ;;
          "up2date")
            # wenn auch Kernel-Updates mit installiert werden sollen, muss
            # up2date um den Parameter -f ergaenzt werden
            ssh $SERVER "sudo su root -c 'up2date -u'" >> $REPORT 2>&1 <&5
            ;;
          *)
            echo "kein gueltiger updater erkannt"
            ;;
        esac
        ;;
      *)
        ssh $SERVER "sudo su root -c '$COMMAND'" >> $REPORT 2>&1 <&5
        ;;
    esac
    echo "--------------------------------------------------------------------------------" >> $REPORT
  fi
 done
 exec 5<&-
fi

echo "--------------------------------------------------------------------------------"
echo "done... notifying $NOTIFY"
mail -s "updater-report" $NOTIFY < $REPORT
cp $REPORT `dirname $0`/report-$DATE.log
echo "Bericht gespeichert unter: `dirname $0`/report-$DATE.log"
echo "--------------------------------------------------------------------------------"

rm -f $REPORT
```

freeze und unfreeze der Eingabe

Ein Bash-Shell läßt sich mit einer einfachen Tastenkombination „einfrieren“ und wieder „auftauen“. Den Sinn dahinter sehe ich grade nicht, aber es funktioniert ;) Bei mir kam es leider auch schon öfter mal vor, dass aus Versehen diese Kombi gedrückt wurde, schlecht wenn man die Unfree-Tasten nicht kennt ;)

Das hier ist also ein kleiner Reminder für mich, wenn sowas mal wieder passiert.

Freeze/Einfrieren:

`CTRL+s`

Unfreeze/Auftauen:

`CTRL+q`

Heredoc Beispiele und Tipps (Here Document)

Heredocs dienen zur Definition von Textabschnitten. Im Unterschied zur herkömmlichen Ausgabe per echo oder printf (jaja, ich weiß, dort geht das auch) werden enthaltene Zeilenumbrüche, Einzüge und einige Sonderzeichen im Text bewahrt. Die Bash ermöglicht auch die Verwendung von Variablen innerhalb des Heredoc.

Beispiel

Beispiel für eine sich wiederholende Konfigurationsoption (viele neue Hosts müssen einer Munin-Konfiguration hinzugefügt werden):

```
for i in `seq 100 150`; do
cat <<EOF
[xen$i.meinecloud.de]
        address xen$i.meinecloud.de
        use_node_name yes

EOF
done
```

bash-completion

Wer einen gewissen Komfort bei der Steuerung von Linux via Kommandzeile vermisst, sollte sich das Paket bash-completion via yum installieren.

```
yum -y install bash-completion
```

Danach einfach ein neues Terminal öffnen, reloggen oder das Kommando „. /etc/bash_completion“ ausführen, um die neuen Features zu laden.

Im Folgenden einige Beispiele der Möglichkeiten, die einem nun offenstehen:

Als root:

```
service send[Tab]
```

dies verfollständigt den Aufruf automatisch zu „sendmail“.

Eine der Besonderheiten ist die Vervollständigung von Paketnamen beim Aufruf von yum. Aus

```
[root@ov ~]# yum -y install never[Tab]
```

wird automagisch

```
[root@ov ~]# yum install neverball.i386
```

Allerdings kommt es hierbei zu einer kurzen Verzögerung der Ausgabe, deshalb nicht gleich auf der Tab-Taste herumhämmern.

drückt man 2x die Tabtaste wird eine Liste von möglichen Ergänzungen angezeigt:

```
 [root@ov ~]# man send[Tab] send sendfile sendiso sendmail.sendmail sendsms send_to_keyboard send_v2trap
send_easy_trap sendfile64 sendmail sendmsg sendto send_trap_vars
```

das funktioniert auch mit svn:

```
[magenbrot@ov:~]$ svn [Tab]
?       blame    cleanup  cp       diff     import   log      move     pe       pl       propedit  ps       rename
st      switch
add     cat      co       del      export   info     ls       mv       pedit    plist    propget   pset     resolved  stat
unlock
ann     checkout  commit   delete   h        list     merge    pd       pg       praise   proplist  remove   revert
status   up
annotate  ci      copy     di       help     lock     mkdir    pdel     pget     propdel  propset   ren      rm
sw      update
```

make:

```
[magenbrot@ov:/usr/src/qc-usb-messenger-1.7]$ make [Tab]
make              makedepend          make_driver_db_lpr   makeindex          makekdewidgets
makempx           makeobj             makestrs
makedb            make_driver_db_cups  makeg               makeinfo           makemap            makempy
makerandom
```

modprobe:

```
[root@ov ~]# modprobe ipt_[Tab]
ipt_addrtype    ipt_CLUSTERIP   ipt_ECN         ipt_MASQUERADE  ipt_recent      ipt_REJECT      ipt_TTL
ipt_ah          ipt_ecn         ipt_LOG         ipt_NETMAP      ipt_REDIRECT    ipt_ttl         ipt_ULOG
```

Vervollständigung der Hostnamen etwa bei ssh:

```
ssh www.[Tab]
www.bla.de  www.fasel.de www.xxx.de
```

Vervollständigung der Directories beim Kopieren mit scp (wieder mit kleiner Verzögerung nach Tab):

```
[magenbrot@ov:~]$ scp liste.txt magenbrot@magenbrot.net:/home/magenbrot/
/home/magenbrot/bin/       /home/magenbrot/firewall/     /home/magenbrot/mail/      /home/magenbrot/srv/
```

Anzeige der unterstützen Parameter bei einem Programm:

```
[magenbrot@ov:~]$ less --
---                 --force             --IGNORE-CASE       --max-back-scroll=  --QUIT-AT-EOF       --squeeze-blank-
lines  --version
--auto-buffers      --help              --jump-target=      --max-forw-scroll=  --quit-if-one-screen  --status-
column      --window=
--buffers=          --hilite-search     --lesskey-file=     --no-init           --quotes=           --tabs=
--chop-long-lines   --HILITE-SEARCH     --LINE-NUMBERS      --no-keypad         --RAW-CONTROL-CHARS  --
tag=
--clear-screen      --hilite-unread     --log-file=         --no-lessopen       --search-skip-screen  --tag-
file=
--color=            --HILITE-UNREAD     --LOG-FILE=         --pattern=          --shift=            --tilde
--dumb              --ignore-case       --LONG-PROMPT       --prompt=           --SILENT            --UNDERLINE-
SPECIAL
```

Programm- oder Funktionsrückgabe auswerten

In Bash-Scripten werden oft externe Programme und Tests ausgeführt deren Rückgabewerte man kontrollieren möchte, z.B. ob das Programm gelaufen ist oder ob es einen Fehler gab.

Folgendes wird gerne verwendet, ist aber für einen einfachen Test etwas umständlich:

```
# Beispiel: gibts einen bestimmten Benutzer?
grep -q username /etc/passwd
if (($? > 0)); then
  echo 'Benutzer wurde nicht gefunden' >&2
  exit 1
fi
```

Schneller und besser zu lesen ist dieser Code:

```
if grep -q username /etc/passwd; then
  # nix machen
  :
else
  echo 'Benutzer wurde nicht gefunden' >&2
  exit 1
fi

# oder noch kürzer mit Negation
if grep -q username /etc/passwd; then
  echo 'Benutzer wurde nicht gefunden' >&2
  exit 1
fi
```

Alternativ gehts auch mit einer || (ODER, wenn der Rückgabewert nicht 0 ist) oder && (UND, wenn der Aufruf eine 0 zurückgibt) Verknüpfung:

```
grep -q username /etc/passwd || {
  echo 'Benutzer wurde nicht gefunden' >&2
  exit 1
}
```

Die Auswertung von $? ist damit eigentlich nur notwendig, wenn man tatsächlich auf einen ganz bestimmten Rückgabewert prüfen will. rsync etwa liefert für viele Fehler einen bestimmten Code:

- 0 Success
- 1 Syntax or usage error
- 2 Protocol incompatibility
- 3 Errors selecting input/output files, dirs
- 4 Requested action not supported: an attempt was made to manipulate 64-bit files on a platform that cannot support them; or an option was specified that is supported by the client and not by the server.
- 5 Error starting client-server protocol
- 6 Daemon unable to append to log-file
- 10 Error in socket I/O
- 11 Error in file I/O
- 12 Error in rsync protocol data stream
- 13 Errors with program diagnostics
- 14 Error in IPC code
- 20 Received SIGUSR1 or SIGINT
- 21 Some error returned by waitpid()
- 22 Error allocating core memory buffers
- 23 Partial transfer due to error
- 24 Partial transfer due to vanished source files
- 25 The –max-delete limit stopped deletions
- 30 Timeout in data send/receive
- 35 Timeout waiting for daemon connection

So kann z.B. nur der Fehler in ein Logfile geschrieben werden:

```
rsync -a --delete quelle ziel
if (($? == 30)); then
   echo "Timeout in data send/receive" >> $logfile
   exit 1
fi
```

BASH-History mit Datum

einfach folgende Zeile ans Ende der /etc/bashrc hängen:

`export HISTTIMEFORMAT="%Y-%m-%d - %H:%M:%S "`

und schon wird aus:

```
  988  cd /etc
  989  cd /
  990  cd var
  991  cd log
  992  ll
  993  cd /tftpboot/
  994  ll
  995  cd linux-install/
```

das hier:

```
 1000  2009-05-07 - 20:19:07 vi /etc/bashrc
 1001  2009-05-07 - 20:19:08 history
 1002  2009-05-07 - 20:25:35 cat /etc/bashrc
 1003  2009-05-07 - 20:26:34 history
```

meine .bashrc und .profile

eingesetzt unter Ubuntu 10.04. Inzwischen reichlich veraltet, auf dem Desktop nutze ich inzwischen Fish.

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
# ... or force ignoredups and ignorespace
HISTCONTROL=ignoredups:ignorespace

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=2048
HISTFILESIZE=4096

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "$debian_chroot" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color) color_prompt=yes;;
esac
```

```
# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
	# We have color support; assume it's compliant with Ecma-48
	# (ISO/IEC-6429). (Lack of such support is extremely rare, and such
	# a case would tend to support setf rather than setaf.)
	color_prompt=yes
    else
	color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then

PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'
```

```
    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

#if [ -f ~/.bash_aliases ]; then
#    . ~/.bash_aliases
#fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

# rdesktop
alias rdesktop='rdesktop -g 1270x950 -k de -a 24 -r sound:off -N'
alias rda='rdesktop -uAdministrator -pXXXXX'

# various
alias key='cat ~/key-ssh'
alias xpwgen='pwgen -B -s -y 10'
alias pwgen='pwgen -s -B'
alias irssi='screen -S irssi -xR irssi'
alias transmission-remote='transmission-remote -n magenbrot:XXXXX'
```

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
```

```
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# add android binaries to the search $PATH
export PATH=$PATH:/home/ovoelker/apps/android-sdk-linux_86/tools
```