```
 B r e L U G Information Circular  Nr.: 7     05/2000
-------------------------------------------------------
Thema: mirr_arc, mirr_clean und mirr_sync
       3 kleine Shell-Spielereien zum Spiegeln und
       Löschen von Dateien und Verzeichnissen
-------------------------------------------------------
```

Hallo liebe Linuxer in Bretten und Umgebung,

mich gibt es auch noch! Obwohl ich nur alle Jubeljahre zu einem unserer Treffen kommen kann (Mist Schichtdienst!), habe ich mein Interesse an Linux nicht verloren. Und man glaubt es kaum: Ab und zu sitze ich auch mal vor dem Rechner und spiele damit, und ab und an kommt auch etwas dabei heraus! Das Ergebnis sind 3 mehr oder weniger kleine Shell-Scripte, die ich in Form einer neuen BIC einmal in unseren Mail-Lister (und auch in die Hompepage?) stellen möchte, damit auch die anderen etwas davon haben und/oder sehen, was man mit so kleinen Spielereien alles anstellen kann. Und vielleicht kann der Eine oder Andere ja selber auch diese Spielereien anwenden. Vor allem aber würde ich mich über Kommentare, Kritik und Anregungen zu diesen Scripten freuen.

Allerdings sind es wieder "Kommandozeilenbefehle" geworden. Ich gebe gerne zu, dass ich Kommandozeilen-geschädigt bin, aber ich glaube das es keine KDE- oder Windows-Spielerei gibt, die das leistet was ich gerne will. Und ein wenig Kommandozeile tut doch ab und zu auch mal wieder gut!

Nun also zur Sache! Erst einmal die Vorgeschichte der 3 kleinen Programme. Ich habe bei mir hier zuhause inzwischen 3 Rechner im Einsatz, meine "große" Maschine, ein K6-300 mit 20GB Plattenkapazität, davon inzwischen ca. 14GB für Linux reserviert (Debian 2.1). Desweiteren ist im Einsatz ein Laptop, 6GB Platte, davon 5GB unter Linux (ebenfalls Debian 2.1). Und last not least noch meine ganz alte Maschine, 486-er mit 4GB Platte, komplett mit Linux (Suse 6.3) belegt. Ein richtiger M$-freier Rechner! In allen Rechnern sind Netzkarten installiert, der K6 und der 486-er sind permanent übers Netz miteinander verbunden, der kleine nur ab und zu.

Schon seit vielen Jahren nutze ich die Computerei auch privat recht heftig, Briefeschreiben, Bilder bearbeiten, Geburtstagswünsche als selbstgemachte Geburtstagskarte entwerfen, Haushaltsgeld verwalten usw.usw. Dazu kommen noch Programmtexte von diversen Programmier-Spielereien, die (wenn ich mal ein wenig Zeit habe) auch immer sehr viel Spass machen. Das allermeiste von diesen Dingen läuft bei mir jetzt unter Linux, mit Star Office, Gnu-Cash, meinem eigenen Überweisungsprogramm (ist noch nicht ganz fertig, kann ich dann auch mal in den Maillister stellen), emacs und noch ein paar anderen Anwendungen. So ist in den Jahren doch eine ganze Menge an "Schrott" und Daten zusammengekommen, die ich natürlich alle unter Linux gespeichert habe und verwalte.

Zu diesem Zweck gibt es bei mir auf der festen Platte ein eigenes Verzeichnis

```
/home/daten
```

unter dem dann ein ganzer Unterverzeichnisbaum existiert. Zugriff auf dieses Verzeichnis hat in meinem System nur ein ganz bestimmter user und natürlich root. In diesem Verzeichnis gibt es dann Unterverzeichnisse wie

```
/home/daten/briefe
/home/daten/faxe
/home/daten/programmierung
```

usw. usw. usw. Auch in diesen Verzeichnissen sind weitere Unterverzeichnisse vorhanden. Gesamtmenge der Daten, die sich so angesammelt haben, beträgt inzwischen weit über 400MB!

So entstand bei mir der Wunsch, das ganze /home/daten Unterverzeichnis irgendwo komprimiert zu archivieren, es mehrfach auf Datenträger zu sichern (Band oder CD-ROM), und im eigentlichen /home/daten Verzeichnis nur die neueren und aktuellen Dinge zu halten. Dafür wollte ich gerne ein kleines Werkzeug oder Programm haben, mit den ich dann in regelmässigen Abständen alle Änderungen, neuere Versionen oder neu erstellte Daten in das komprimierte Archiv schreiben könnte.

Zu diesem Zweck hatte ich schon einmal ein kleines Shell-Script geschrieben, und es auch schon einmal in der BIC 03 Feb/99 vorgestellt und beschrieben. Damals hiess das Script mirror, jetzt ist es umbenannt in mirr_arc. Inzwischen ist das Script ein wenig verändert worden, aber im Prinzip ist die Funktion davon genauso wie im BIC 03 beschrieben.

Nun haben sich aber bei mir noch einige weitere Erfordernisse eingestellt. Inzwischen bearbeite ich meine Daten mal am großen Rechner, mal am tragbaren, und ab und zu auch am 486er! Auf jeden Rechner gibt es ein /home/daten Verzeichnis, mit den darunter gehaltenen Gestrüpp an Unterverzeichnissen. Und nun beginnt das Durcheinander, ob ich nun will oder nicht!

So schreibe ich dann einen Brief auf dem tragbaren, arbeite mehrere Tage daran, mache meine Ausbildungsstundenabrechnung auf dem großen, will sie dann auf dem kleinen ansehen und verändern, ich glaube ihr wisst schon was ich meine. Die Daten sollten ja auf allen 3 Rechnern, mindestens aber auf dem Großen und dem tragbaren, immer konsistent und parallel sein. So müsste ich immer gleich, wenn ich mal auf dem kleinen gearbeitet habe, den kleinen an das Netzkabel anschließen, den großen hochfahren, und dann fein säuberlich alle geänderten oder neuen Dateien auf den großen an die richtige Stelle rüberspielen. Und natürlich dergleichen in die andere Richtung. Meistens aber kommt man nicht gleich dazu, und schon ist die Übersicht weg, welche Dateien auf welchem Rechner denn nun neuer oder akuteller sind!

Sicher gibt es für dieses Problem eine ganze Latte an vorhandenen Lösungen, und die Netzwerkfreaks unter Euch werden jetzt sicher ein "Gähn" von sich geben! Ich selber habe auch mal in die HOWTO's geschaut, und in die man-pages, aber so richtig habe ich da keine Lösung für mich gefunden. rdist und rsync wurde da erwähnt, aber die Manual Seiten von diesen Befehlen sind so etwas von kryptisch! Nach ein paar vergeblichen Versuchen, diese Dinger über mein kleines Netz auszuprobieren und zu verstehen, habe ich dann die Sache hingeworfen. Dann gab es da ein perl-Script im Linux Magazin, ich glaube im April, geschrieben zum gleichen Thema. Also habe ich das ausprobiert, und was tat sich? "Syntax error line 425!" Mit perl kenne ich mich überhaub nicht aus, so habe ich das Script schneller wieder vom Rechner gelöscht, als ich es per download vom Linuxmagazin hergekriegt hatte! Dann habe ich mir überlegt, ob ich es auf eine primitive Art mit Shell-Scripten nicht selber hinkriegen könnte. Und siehe da: Ich glaube es tut!

So entstanden dann das geänderte Script mirr_arc und die beiden anderen Srcipte mirr_sync und mirr_clean. Ich füge (wie auch schon in der BIC 03) alle 3 Scripte am Ende zum Ausschneiden, Abspeichern und Ausprobieren an.

So ist die Situation bei mir:

Zur besseren Übersicht beschreibe ich nur die Lage zwischen meinem grossen Rechner und dem kleinen. Der grosse Rechner hat im Netz den Rechnernamen df6om.ampr.org (kurz df6om), dieser Rechnername und Domainname ist sogar höchstoffiziell! Als IP-Nummer benutze ich die diesem Rechner offiziell zugeteilte IP-Nummer. Der kleine ist dann in meinem Netz als df6om-p.ampr.org (kurz df6om-p oder auch nur p) bekannt, mit einer nur in meinem internen Netz gültigen IP-Nummer. Auf allen Rechnern bei mir ist NFS aktiviert, ich habe auch immer NFS beim Bauen eines neuen Kernels aktiviert.

In der Datei /etc/exports des grossen Rechners, also df6om, ist nun folgende Zeile eingetragen

```
/home/daten        df6om-p(rw)
```

und in der Datei /etc/exports auf dem kleinen Rechner ist die Zeile

```
/home/daten        df6om(rw)
```

eingetragen. Diese Zeilen bedeuten, dass der grosse Rechner das Verzeichnis /home/daten über NFS dem kleinen (df6om-p) zur Verfügung stellt, und umgekehrt der kleine sein Verzeichnis /home/daten dem grossen (df6om) auch über NFS zur Verfügung stellt. Beiden werden jeweils Lese- und Schreibrechte per NFS eingeräumt.

Auf beiden Rechnern habe ich im /home Verzeichnis ein leeres Verzeichnis /home/daten_remote erzeugt. Sitze ich nun am kleinen Rechner, und habe seit einiger Zeit auf ihm Arbeiten mit meine Daten durchgeführt, aber inzwischen auch schon am großen Rechner Daten erzeugt oder verändert, so muß ich ihn natürlich erst mal ans Netzkabel (nicht das Stromnetzkabel, sondern das Ethernetkabel!) anschliessen. Als user root gebe ich dann folgenden Befehl

```
mount -t nfs df6om:/home/daten /home/daten_remote
```

und schwupp hängt das /home/daten Verzeichnis des grossen Rechners bei mir auf dem kleinen Rechner unter dem Verzeichnis /home/daten_remote! Mit dem Midnight Commander oder anderen File-Managern kann man dann sehr schön sehen, wie die Dateien in beiden Verzeichnissen liegen.

Nun gebe ich als der user, der das Verzeichnis /home/daten auf beiden Rechnern bearbeiten darf, den Befehl

```
mirr_sync /home/daten /home/daten_remote
```

und mein Shell-Script spielt alle Daten neueren Datums vom kleinen rüber auf den grossen, und umgekehrt alle neueren Dinge vom grossen rüber auf den kleinen. Wie ich schon im BIC 03 beschrieben hatte, kann ich alle 3 mirr_... Scripte mit der -n Option aufrufen, dann kriege ich auf dem Bildschirm (oder im xterm) angezeigt was kopiert werden würde, aber nur ohne diese Option wird wirklich kopiert. - Wenn ich gleich darauf diesen Befehl noch einmal aufrufe, dann kommt nur noch der prompt zurück, es zeigt sich dass nun nichts mehr hin- und herkopiert wird, die beiden Verzeichnisse sind also wieder syncron!

Abschliessend muss ich dann als user root das externe Verzeichnis wieder abmontieren mit dem Befehl

```
umount /home/daten_remote
```

und die Aktion ist fertig.

Seitdem das mirr_sync Script tut, klappt es nun prima mit der Syncronisierung meiner Datenverzeichnisse auf beiden Rechnern.

Wofür ist nun das mirr_clean Script? Die Idee bzw. die Notwendigkeit für dieses Script kam sofort auf, nachdem das mirr_sync Script halbwegs richtig funktionierte! Wenn ich nun mit dem (syncronisierten) /home/daten Verzeichnis auch mein komprimiertes Archiv auf dem neuesten Stand gebracht hatte, ergab sich die Situation, dass ich auf dem kleinen Rechner einiges an "Ausmistarbeit" im /home/daten Verzeichnis durchführen wollte. Eine Vielzahl von Dateien sollten rausfliegen, doch halt ..., sie mussten ja nun wieder auf beiden Rechnern rausfliegen. Würde ich demnächst irgendwann wieder mirr_sync aufrufen, würde mein Scriptchen ja die gelöschten Dateien vom anderen Rechner wieder zurückholen. Dafür also nun das mirr_clean Script. Vorbereitung wie oben beschrieben, root muss per nfs das ferne /home/daten Verzeichnis am lokalen Rechner mounten. Nehmen wir wieder an ich sitze an df6om-p und das /home/daten Verzeichnis von df6om ist an /home/daten_remote von df6om-p montiert. Ausgemistet wurde das Verzeichnis /home/daten auf df6om-p, somit muss jetzt das gleiche auch auf df6om passieren. So muss dann mirr_clean aufgerufen werden

```
mirr_clean /home/daten /home/daten_remote
```

und das Script vergleicht die beiden Verzeichnisse und löscht alle Dateien in /home/daten_remote, die in /home/daten NICHT enthalten sind. Oder anders ausgedrückt, das Script lässt in /home/daten_remote nur übrig, was auch in /home/daten existiert! Bingo! Somit hatte ich was ich wollte!

An der Bedienung von mirr_arc hat sich im Grunde nichts geändert, ausser dass wie bei den beiden anderen Scripten auch die Pfadnamen immer als absolute Pfade angegeben werden müssen. Bei mir existiert das komprimierte Datenarchiv auf dem grossen Rechner unter einer eigenen Partition, die am Verzeichnis /extra montiert ist. Das eigentliche Archiv liegt dann in einem Unterverzeichnis /extra/mirr_arc.d So bringe ich das Archiv mit dem Aufruf

```
mirr_arc /home/daten /extra
```

auf den neuesten Stand. Die genauen Details der Bedienung sind wie schon gesagt in der BIC 03 nachzulesen. Auch im Kopf aller 3 Scripte ist eine Kurzbeschreibung enthalten, und ruft man jedes Script ohne alle Optionen oder Argumente auf (z.B. nur mirr_arc), so meldet sich das jeweilige Programm auch mit einer Kurzbeschreibung, wie jedes "richtige" Unix-tool.

Natürlich funktionieren alle 3 Scripte nicht nür über per NFS montierte Verzeichnisse, sondern zwischen allen Verzeichnissen in einem System. Ich bitte um heftiges Nachspielen, aber VORSICHT mit dem mirr_clean Script!! Probiert es am besten mit ungefährlichen Dummy-Dateien aus, und ich habe mir angewöhnt, es immer erst einmal mit der -n Option zu starten, um zu sehen was es an Dateien löscht! Der Aufruf ist dann also

```
mirr_clean -n /home/daten /home/daten_remote
```

und es wird noch nix gelöscht, sondern nur angezeigt was gelöscht würde!

So, das wars eigentlich schon, ich wollte diesmal ganz auf die Schnelle eine BIC herunterklopfen und ich habe nur etwas mehr wie eine Stunde getippt! Ich hoffe dass das Thema nicht zu trivial ist und für einige von Euch trotzdem interessant ist. Über Kommentare, Kritik und vielleicht auch andere Lösungen von Euch würde ich mich sehr freuen.

Leider bin ich beim nächsten Linux-Treffen im Juni wieder mal nicht dabei, im Juni habe ich meinen Jahresurlaub reserviert bekommen und dann bin ich mit meiner Frau für 2 1/2 Wochen wieder auf Achse, Richtung Norden und dann immer geradeaus! Vielleicht klappt es dann im Juli wieder. Auf alle Fälle wünsche ich Euch viel Spass beim Lesen und Ausprobieren!

73 de jumbo (Juergen)

```
------------------------- schnipp ----------------------------
#!/bin/bash
# mirr_arc
# script to create/update a mirror archive from a chosen source
# directory, storing/updating all files including subdirectories
# in a directory "mirr_arc.d" present or created under the
# target directory.
# All mirrored files will be compressed using the gzip utility.
# If program is called with the "-n" option, all actions are
# shown (echoed) on the screen, but no action is
# actually performed (= showmode).
#
# programmed Dec 03-08, 1998 by jumbo (df6om@delug.de)
# modified Dec 08, 1998 by woody (woody@walbrecht.com)
# updated Jan 11, 1999 by jumbo (df6om@delug.de)
# modified and updated May 12, 2000 by jumbo (df6om@delug.de)
# modified and updated May 28, 2000 by jumbo (df6om@delug.de)
# modified and uptdate May 30, 2000 by jumbo (df6om@delug.de)



# setup some message strings used throughout the program

PROGNAME="mirr_arc"
ARCDIR="mirr_arc.d"
USE_STR="usage: $PROGNAME [-n] /Path_to_SourceDir /Path_to_TargetDir"



# function usage(), called whenever program was invoked
# with incorrect syntax, arguments or options

usage()
{
  echo $1; echo $USE_STR; exit 1
}



# the following case-block checks number and content of
# arguments/options of invocation of program
# Valid is only a number of 2 or 3 arguments after the program
# call itself, either source- and target-dir paths, or a preceding
# "-n" option before the source- and target-dir paths.

case $# in
  0) usage "$PROGNAME: missing options and/or arguments!" ;;
  1) usage "$PROGNAME: missing options and/or arguments!" ;;
  2) if [ ! -d $1 ]
     then
      usage "$PROGNAME: directory $1 does not exist!"
     fi
     if [ ! -d $2 ]
     then
      usage "$PROGNAME: directory $2 does not exist!"
     fi;;
  3) if [ $1 = -n ]
     then
      SHOWMODE=TRUE
    shift
    if [ ! -d $1 ]
    then
      usage "$PROGNAME: directory $1 does not exist!"
    fi
    if [ ! -d $2 ]
    then
      usage "$PROGNAME: directory $2 does not exist!"
    fi
     else
      usage "$PROGNAME: incorrect option!"
     fi;;
  *) usage  "$PROGNAME: too many options and/or arguments";;
```

```
esac



# now we check if the directory-arguments contain a leading slash "/",
# it is required to keep the script simpler!

if ! expr "$1" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: source must be an absolute pathname to a directory!"
fi
if ! expr "$2" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: target must be an absolute pathname to a directory!"
fi



# strip trailing slashes from source and target

SOURCE=`echo $1 | sed 's,/$,,'`
TARGET=`echo $2 | sed 's,/$,,'`



# check if the directory mirr_arc.d exists in the
# target-directory, if not then just make it!

[ ! -d $TARGET/mirr_arc.d ] && mkdir $TARGET/mirr_arc.d


# concat target-directory-name with a trailing "/mirr_arc.d"
# and the source-directory-name.
# The slash after $ARCDIR is omitted since $SOURCE already
# contains a leading slash!

TARGET=$TARGET/$ARCDIR$SOURCE



# if a copy of the source directory does not exist under
# mirr_arc.d in the target-directory, then just make it!

[ ! -d $TARGET ] && mkdir $TARGET


# now change to source-dir

cd $SOURCE



# and finally the most important part
# find all files with type "f" (ordinary files) in source(dir)
# form a list of pathnames of these files including embedded
# directories
# compare if files are newer than possible existing backups
# in target or if they are missing in target completely
# if so, then copy to target-dir creating any embedded directories
# and maintaining file-ownerships, time-stamps and permissions
# finally gzip the copied files


find -type f -printf "%P\n" | while read FILE
do
  IFS=','
  #  this is required so that the test ("[]") utility does
  #  not go nuts on whitespaces in filenames!
  FILE=`echo $FILE | sed 's/\.gz$//'`
  #  strip a trailing .gz in case the source-file
  #  is already gzipped!
  if [ $FILE -nt $TARGET/$FILE.gz -o ! -f $TARGET/$FILE.gz ]
```

```
  then
    echo updating $SOURCE/$FILE to $TARGET/$FILE.gz ...
    if [ ! $SHOWMODE ]
    then
      cp -P $FILE $TARGET
      # chmod -x $TARGET/$FILE
      gzip -f $TARGET/$FILE
    fi
  fi
done
exit 0
-------------------------- schnipp ---------------------------



-------------------------- schnipp ---------------------------
#!/bin/bash
# mirr_clean
# script to clean a target (mirror) directory and containing
# subdirectories from files which are not present (or have been
# previously erased) in a source directory and
# its containing subdirectories.
# Empty subdirectories will presently not be deleted!
#
# If program is called with the "-n" option, all actions are
# shown (echoed) on the screen, but no action is
# actually performed (= showmode).
#
# programmed May 25, 2000 by jumbo (df6om@delug.de)
# modified and updated May 29, 2000 by jumbo (df6om@delug.de)
# modified and updated May 30, 2000 by jumbo (df6om@delug.de)


# setup some message strings used throughout the program

PROGNAME=mirr_clean
USE_STR1="usage: $PROGNAME [-n] /Path_to_SourceDir /Path_to_TargetDir"
USE_STR2=" $PROGNAME will clean (erase) all files in /Path_to_TargetDir"
USE_STR3=" which are NOT contained in /Path_to_SourceDir"



# function usage(), called whenever program was invoked
# with incorrect syntax, arguments or options

usage()
{
  echo $1; echo $USE_STR1; echo $USE_STR2; echo $USE_STR3; exit 1
}



# the following case-block checks number and content of
# arguments/options of invokation of program.
# Valid is only a number of 2 or 3 arguments after the program
# call itself, either source- and target-dir paths, or a preceding
# "-n" option before the source- and target-dir paths.

case $# in
  0) usage "$PROGNAME: missing options and/or arguments!" ;;
  1) usage "$PROGNAME: missing options and/or arguments!" ;;
  2) if [ ! -d $1 ]
     then
      usage "$PROGNAME: directory $1 does not exist!"
     fi
     if [ ! -d $2 ]
     then
      usage "$PROGNAME: directory $2 does not exist!"
     fi;;
  3) if [ $1 = -n ]
     then
      SHOWMODE=TRUE
```

```
    shift
    if [ ! -d $1 ]
    then
      usage "$PROGNAME: directory $1 does not exist!"
    fi
    if [ ! -d $2 ]
    then
      usage "$PROGNAME: directory $2 does not exist!"
    fi
     else
      usage "$PROGNAME: incorrect option!"
     fi;;
  *) usage  "$PROGNAME: too many options and/or arguments";;
esac



# now we check if the directory-arguments contain a leading slash "/",
# it is required to keep the script simpler!

if ! expr "$1" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: source must be an absolute pathname to a directory!"
fi

if ! expr "$2" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: target must be an absolute pathname to a directory!"
fi



# strip trailing slashes from source and target

SOURCE=`echo $1 | sed 's,/$,,'`
TARGET=`echo $2 | sed 's,/$,,'`



# now change to target-dir and remove files which are not
# present in source-dir

cd $TARGET
find -type f -printf "%P\n" | while read FILE
do
  if [ ! -e "$SOURCE/$FILE" ]
  then
    echo "removing $TARGET/$FILE"
    if [ ! $SHOWMODE ]
    then
      rm "$TARGET/$FILE"
    fi
  fi
done



exit 0
--------------------------- schnipp ----------------------------



--------------------------- schnipp ----------------------------
#!/bin/bash
# mirr_sync
# script to create/update/syncronise a mirror directory with
# subdirectories and files from a chosen source directory and
# its containing subdirectories and files
#
# If program is called with the "-n" option, all actions are
# shown (echoed) on the screen, but no action is
# actually performed (= showmode).
```

```
#
# programmed May 13, 2000 by jumbo (df6om@delug.de)
# modified and updated May 28, 2000 by jumbo (df6om@delug.de)
# modified and updated May 30, 2000 by jumbo (df6om@delug.de)



# setup some message strings used throughout the program

PROGNAME=mirr_sync
USE_STR="usage: $PROGNAME [-n] /Path_to_SourceDir /Path_to_TargetDir"



# function usage(), called whenever program was invoked
# with incorrect syntax, arguments or options

usage()
{
  echo $1; echo $USE_STR; exit 1
}



# the following case-block checks number and content of
# arguments/options of invokation of program
# Valid is only a number of 2 or 3 arguments after the program
# call itself, either source- and target-dir paths, or a preceding
# "-n" option before the source- and target-dir paths.

case $# in
  0) usage "$PROGNAME: missing options and/or arguments!" ;;
  1) usage "$PROGNAME: missing options and/or arguments!" ;;
  2) if [ ! -d $1 ]
     then
      usage "$PROGNAME: directory $1 does not exist!"
     fi
     if [ ! -d $2 ]
     then
      usage "$PROGNAME: directory $2 does not exist!"
     fi;;
  3) if [ $1 = -n ]
     then
      SHOWMODE=TRUE
    shift
    if [ ! -d $1 ]
    then
      usage "$PROGNAME: directory $1 does not exist!"
    fi
    if [ ! -d $2 ]
    then
      usage "$PROGNAME: directory $2 does not exist!"
    fi
     else
      usage "$PROGNAME: incorrect option!"
     fi;;
  *) usage  "$PROGNAME: too many options and/or arguments";;
esac



# now we check if the directory-arguments contain a leading slash "/",
# it is required to keep the script simpler!

if ! expr "$1" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: source must be an absolute pathname to a directory!"
fi

if ! expr "$2" : "^/" > /dev/null 2>&1
then
  usage "$PROGNAME: target must be an absolute pathname to a directory!"
```

```
fi



# strip trailing slashes from source and target

SOURCE=`echo $1 | sed 's,/$,,'`
TARGET=`echo $2 | sed 's,/$,,'`


# if target-directory does not exist, then just make it!

[ ! -d "$TARGET" ] && mkdir $TARGET



# now change to source-dir and copy files which are not
# present in target-dir

cd $SOURCE
find -type f -printf "%P\n" | while read FILE
do
  if [ ! -e "$TARGET/$FILE" ]
  then
    echo "copying $SOURCE/$FILE to $TARGET"
    if [ ! $SHOWMODE ]
    then
      cp -p -P "$FILE" $TARGET
    fi
  fi
done

# now change to target-dir and copy files which are not
# present in source-dir

cd $TARGET
find -type f -printf "%P\n" | while read FILE
do
  if [ ! -e "$SOURCE/$FILE" ]
  then
    echo "copying $TARGET/$FILE to $SOURCE"
    if [ ! $SHOWMODE ]
    then
      cp -p -P "$FILE" $SOURCE
    fi
  fi
done


# now change to source-dir and copy files which are newer
# than files in target-dir

cd $SOURCE
find -type f -printf "%P\n" | while read FILE
do
  if [ "$FILE" -nt "$TARGET/$FILE" ]
  then
    echo "copying $SOURCE/$FILE to $TARGET"
    if [ ! $SHOWMODE ]
    then
      cp -p -P "$FILE" $TARGET
    fi
  fi
done


# now change to target-dir and copy files which are newer
# than files in source-dir

cd $TARGET
find -type f -printf "%P\n" | while read FILE
do
```

```
  if [ "$FILE" -nt "$SOURCE/$FILE" ]
  then
    echo "copying $TARGET/$FILE to $SOURCE"
    if [ ! $SHOWMODE ]
    then
      cp -p -P "$FILE" $SOURCE
    fi
  fi
done



exit 0
--------------------------- schnipp ----------------------------
```