98

AIX update

December 2003

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1999 issue, are available separately to subscribers for £16.00 ($24.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aix; you will need to supply a word from the printed issue.

Editor

Trevor Eddolls

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £100 ($160) per 1000 words and £50 ($80) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £20 ($32) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Synchronize passwords

Here is a small script that synchronizes password files between two HACMP (cluster) nodes. It's really critical to keep both nodes identical, especially when a take-over happens. If administrators fail to change passwords on both nodes, then, after a crash, when the second node takes over, a lot of problems can occur with the applications, users, etc.

Here is a script to synchronize passwd files (this script can be executed regularly from node_1 crontab).

PASSWD_SYNC.SH

```
#!/bin/ksh
# Adnan Akbas , 05.06.2002
# The script synchronizes passwd file between two hacmp nodes.
node_1=venus
node_2=mars

# Check if the other node is up

if ping -c 1 $node_2  > /dev/null 2>&1; then
   # Compare if the files are identical or not

   if [[ 'cksum /etc/security/passwd | awk '{print $1}'' -ne 'rsh
${node_2}
cksum /etc/security/passwd | awk '{print $1}'` ]] ; then
      # Finding out which file is newer
      rcp -p ${node_2}:/etc/security/passwd /tmp/passwd.${node_2}
      cp -p /etc/security/passwd /tmp/passwd.${node_1}
      ls -t /tmp/passwd.* | read first junk

      if [[ "/tmp/passwd.${node_1}" = $first ]] ; then
        rcp -p /tmp/passwd.${node_1} ${node_2}:/etc/security/passwd
      else
        cp -p /tmp/passwd.${node_2} /etc/security/passwd
      fi
   fi
fi
```

Adnan Akbas
System Administrator
TURKCELL (Germany)

Manage FTP process – part 2

This month we conclude the code for the mfp utility, which automates the whole process of performing FTP.

```
cp ${TEMP_FILE_1}  ${TEMP_FILE_2}
# allow user to mofify the file
vi ${TEMP_FILE_1}
# establish  the changed record id
OLD_CONFIG_RECORD=""
OLD_CONFIG_RECORD='diff  ${TEMP_FILE_1}  ${TEMP_FILE_2} | tail -1 | awk
{'print $2'}'
# establish bad configuration record which is the output from diff and
# without the sign  >
BAD_CONFIG_RECORD="'diff  ${TEMP_FILE_1}  ${TEMP_FILE_2} | tail -1 | sed
s/'>'//'"
if [ "${OLD_CONFIG_RECORD}" = ""  ]
then
    DisplayMessage I "${NO_CHANGE}" N
    return $FALSE
elif ! echo  "${OLD_CONFIG_RECORD}" | grep  "Remote_Host" > /dev/null
2>&1
then
    # bad configuration record selected
    # format the record for displaying
    BAD_CONFIG_RECORD='echo "${BAD_CONFIG_RECORD}" | sed s/'>'//g | sed
s/'\\/'/'\\/'/g'
    DisplayMessage E "${INVALID_CONFIG_RECORD}"  N
    return $FALSE
fi
# tokenize   current  configuration  record
OLD_REMOTE_HOST='echo      "${OLD_CONFIG_RECORD}" | cut -d':' -f1 | cut
-d'=' -f2'
OLD_REMOTE_DIR='echo       "${OLD_CONFIG_RECORD}" | cut -d':' -f2 | cut
-d'=' -f2'
OLD_REMOTE_FILE='echo      "${OLD_CONFIG_RECORD}" | cut -d':' -f3 | cut
-d'=' -f2'
OLD_REMOTE_UID='echo       "${OLD_CONFIG_RECORD}" | cut -d':' -f4 | cut
-d'=' -f2'
OLD_REMOTE_PWD='echo       "${OLD_CONFIG_RECORD}" | cut -d':' -f5 | cut
-d'=' -f2'
OLD_LOCAL_DIR='echo        "${OLD_CONFIG_RECORD}" | cut -d':' -f6 | cut
-d'=' -f2'
OLD_LOCAL_FILE='echo       "${OLD_CONFIG_RECORD}" | cut -d':' -f7 | cut
-d'=' -f2'
OLD_ACTION='echo           "${OLD_CONFIG_RECORD}" | cut -d':' -f8 | cut
```

```
-d'=' -f2'
OLD_COMMENT='echo          "${OLD_CONFIG_RECORD}" | cut -d':' -f9 | cut
-d'=' -f2'
# set multi-pass flag to $FALSE  because
# old values exist
FIRST_PASS="${FALSE}"
# {
while  true
do
   # get remote hostname
   if ! GetRemoteHostName
   then
        return $FALSE
   fi
   # get remote dir name
   if ! GetRemoteDirectoryName
   then
        return $FALSE
   fi
   # get remote file name
   if ! GetRemoteFileName
   then
        return $FALSE
   fi
   # get remote user id
   if ! GetRemoteUserId
   then
        return $FALSE
   fi
   #get remote password
   if ! GetRemotePassword
   then
        return $FALSE
   fi
   # get local dir name
   if ! GetLocalDirectoryName
   then
        return $FALSE
   fi
   # get local file name
   if ! GetLocalFileName
   then
        return $FALSE
   fi
   # get action
   if ! GetAction
   then
        return $FALSE
   fi
   # get comment
```

```
   if ! GetComment
   then
        return $FALSE
   fi
   # assemble new configuration record
  NEW_CONFIG_RECORD="Remote_Host=${REMOTE_HOST}:Remote_Dir=${REMOTE_DIR}"
   NEW_CONFIG_RECORD="${NEW_CONFIG_RECORD}:Remote_File=${REMOTE_FILE}"
 NEW_CONFIG_RECORD="${NEW_CONFIG_RECORD}:Remote_Uid=${REMOTE_UID}:Remote_Pwd=${REMOTE_PWD}"
 NEW_CONFIG_RECORD="${NEW_CONFIG_RECORD}:Local_Dir=${LOCAL_DIR}:Local_File=${LOCAL_FILE}:
Action=${ACTION}:Comment=${COMMENT}"
   # get confirmation
   REPLY=""
   while true
   do
     tput clear
     echo "Config Record:${NEW_CONFIG_RECORD}"
     echo "Confirm configuration record(Y/N):\c"
     read REPLY
     case ${REPLY}  in
        Y)   break ;;
        N)   break ;;
        *)   :  ;;
     esac
   done
   if  [ "${REPLY}"  = "Y"  ]
   then
        break
   else
      # store previous values and allow  next pass
      OLD_REMOTE_HOST="${REMOTE_HOST}"
      OLD_REMOTE_DIR="${REMOTE_DIR}"
      OLD_REMOTE_FILE="${REMOTE_FILE}"
      OLD_REMOTE_UID="${REMOTE_UID}"
      OLD_REMOTE_PWD="${REMOTE_PWD}"
      OLD_LOCAL_DIR="${LOCAL_DIR}"
      OLD_LOCAL_FILE="${LOCAL_FILE}"
      OLD_ACTION="${ACTION}"
      OLD_COMMENT="${COMMENT}"
   fi
done
#}
# check for configuration file existence
if [ ! -s  ${MFP_CONFIG_FILE}   ]
then
   # initialize the file with appropriate header
   CONFIG_FILE_HEADER="#\n# FTP  Configuration file\n# Do not Edit by
Hand\n# Use mfp.ksh to add/remove Entries\n# Apply Appropriate File
Protection to Safeguard Password Entries\n# #C# Records are commented
out\n# #D# Records are logically deleted\n# #D##C# Records are commented
out and then logically deleted\n#"
```

```
   echo "${CONFIG_FILE_HEADER}"  > ${MFP_CONFIG_FILE}
else
   # check for duplicate entries
   if fgrep -x "${NEW_CONFIG_RECORD}"  ${MFP_CONFIG_FILE}  > /dev/null
   then
     DisplayMessage  E "${ENTRY_EXISTS}" Y
     return $FALSE
   fi
fi
#Prepare Record Entry for the file
#DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
#RECORD_ENTRY="#\n# Entry History\n# ------------\n# ${DATETIME}
Creation\n#\n${CONFIG_RECORD}"
cp ${MFP_CONFIG_FILE}  ${TEMP_FILE_1}
#echo "${RECORD_ENTRY}"    >> ${MFP_CONFIG_FILE}
sed s/"${OLD_CONFIG_RECORD}"/"${NEW_CONFIG_RECORD}"/ ${TEMP_FILE_1}
>${MFP_CONFIG_FILE}
# update history
# get line number for the entry
ENTRY_LINE_NO='fgrep -xn "${NEW_CONFIG_RECORD}" ${MFP_CONFIG_FILE} | cut
-d':' -f1'
#  update history record should be two lines above this
#  because we're using i command for  ed, we need the
#  line number  for one line   above
HISTORY_REC_LINE_NO='expr ${ENTRY_LINE_NO} - 1'
DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
HISTORY_REC="# ${DATETIME} Entry Replaced"
ed <<! ${MFP_CONFIG_FILE} > /dev/null
${HISTORY_REC_LINE_NO}i
${HISTORY_REC}
.
w
q
!
if  IsFTPDaemonRunning
then
    DisplayMessage I "${RELOAD_CONFIG_FILE}" N
fi
}
################################################################################
#  Name     : GetRemoteHostName
#  Overview : The function allows users to enter a remote host name.
#  Returns  : $TRUE  if a value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
################################################################################
GetRemoteHostName  ()
{
tput clear
```

```
REMOTE_HOST=""
while  [  -z "${REMOTE_HOST}"   ]
do
   tput clear
   if [ "${FIRST_PASS}" = "${TRUE}"  ]
   then
         echo "Enter  remote host name ( q to quit ):\c"
   else
         echo "Enter  remote host name or "
         echo "Press Enter to accept old value(${OLD_REMOTE_HOST}) ( q
to quit ):\c"
   fi
   read  REMOTE_HOST
   if [ "${REMOTE_HOST}"  = "q" ]
   then
          return $FALSE

   elif [  "${REMOTE_HOST}"  != ""   ]
   then
          return $TRUE
   elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${REMOTE_HOST}"  = ""  ]
   then
          REMOTE_HOST="${OLD_REMOTE_HOST}"
          return $TRUE
   fi
done
}
###############################################################################
#  Name     : GetRemoteDirectoryName
#  Overview : The function allows users to enter a remote directory
#             name.
#  Returns  : $TRUE  if a value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
###############################################################################
GetRemoteDirectoryName ()
{
tput clear
REMOTE_DIR=""
while  [  -z "${REMOTE_DIR}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter  remote directory name ( q to quit ):\c"
      else
         echo "Enter  remote directory name or "
         echo "Press Enter to accept old value(${OLD_REMOTE_DIR}) ( a to
quit ):\c"
```

```
      fi
      read  REMOTE_DIR
      if [ "${REMOTE_DIR}"  = "q" ]
      then
          return $FALSE
      elif [  "${REMOTE_DIR}"  != ""   ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${REMOTE_DIR}"  = ""  ]
      then
          REMOTE_DIR="${OLD_REMOTE_DIR}"
          return $TRUE
      fi
done
}
###############################################################################
#  Name     : GetRemoteFileName
#  Overview : The function allows users to enter a remote file name.
#  Returns  : $TRUE  if a  value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
###############################################################################
GetRemoteFileName ()
{
tput clear
REMOTE_FILE=""
while  [  -z "${REMOTE_FILE}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter  remote file name ( q to quit ):\c"
      else
         echo "Enter  remote file name or "
         echo "Press Enter to accept old value(${OLD_REMOTE_FILE}) ( q
to quit ):\c"
      fi
      read  REMOTE_FILE
      if [ "${REMOTE_FILE}"  = "q" ]
      then
          return $FALSE
      elif [  "${REMOTE_FILE}"  != ""   ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${REMOTE_FILE}"  = ""  ]
      then
          REMOTE_FILE="${OLD_REMOTE_FILE}"
          return $TRUE
      fi
```

```
done
}
################################################################################
#  Name     : GetRemoteUserId
#  Overview : The function allows users to enter a remote user id.
#  Returns  : $TRUE  if a  value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
################################################################################
GetRemoteUserId  ()
{
tput clear
REMOTE_UID=""
while  [  -z "${REMOTE_UID}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter  remote username ( q to quit ):\c"
      else
         echo "Enter  remote username or "
         echo "Press Enter to accept old value(${OLD_REMOTE_UID}) ( q to
quit ):\c"
      fi
      read  REMOTE_UID
      if [ "${REMOTE_UID}"  = "q" ]
      then
          return $FALSE
      elif [  "${REMOTE_UID}"  != ""   ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${REMOTE_UID}"  = ""  ]
      then
          REMOTE_UID="${OLD_REMOTE_UID}"
          return $TRUE
      fi
done
}
################################################################################
#  Name     : GetRemotePassword
#  Overview : The function allows users to enter a remote password
#  Returns  : $TRUE  if a  value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
################################################################################
GetRemotePassword  ()
{
tput clear
```

```
REMOTE_PWD=""
while  [  -z "${REMOTE_PWD}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter remote password ( q to quit ):\c"
      else
         echo "Enter remote password or "
         echo "Press Enter to accept old value(${OLD_REMOTE_PWD}) ( q to
quit ):\c"
      fi
      read  REMOTE_PWD
      if [ "${REMOTE_PWD}"  = "q" ]
      then
          return $FALSE
      elif [  "${REMOTE_PWD}"  != ""   ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${REMOTE_PWD}"  = ""  ]
      then
          REMOTE_PWD="${OLD_REMOTE_PWD}"
          return $TRUE
      fi
done
}
##############################################################################
#  Name     : GetLocalDirectoryName
#  Overview : The function allows users to enter a local directory name.
#  Returns  : $TRUE  if a  value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
##############################################################################
GetLocalDirectoryName ()
{
tput clear
LOCAL_DIR=""
while  [  -z "${LOCAL_DIR}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter local directory name ( q to quit ):\c"
      else
         echo "Enter local directory name or "
         echo "Press Enter to accept old value(${OLD_LOCAL_DIR}) ( q to
quit ):\c"
      fi
      read  LOCAL_DIR
```

```
      if [ "${LOCAL_DIR}"  = "q" ]
      then
          return $FALSE
      elif [  "${LOCAL_DIR}"  != ""   ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${LOCAL_DIR}"  = ""  ]
      then
          LOCAL_DIR="${OLD_LOCAL_DIR}"
          return $TRUE
      fi
done
}
###############################################################################
#  Name     : GetLocalFileName
#  Overview : The function allows users to enter a local file name.
#  Returns  : $TRUE  if a value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
###############################################################################
GetLocalFileName ()
{
tput clear
LOCAL_FILE=""
while  [  -z "${LOCAL_FILE}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter local file  name ( q to quit ):\c"
      else
         echo "Enter  local file name or "
         echo "Press Enter to accept old value(${OLD_LOCAL_FILE}) ( q to
quit ):\c"
      fi
      read  LOCAL_FILE
      if [ "${LOCAL_FILE}"  = "q" ]
      then
          return $FALSE
      elif [ "${LOCAL_FILE}"  != ""  ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${LOCAL_FILE}"  = ""  ]
      then
          LOCAL_FILE="${OLD_LOCAL_FILE}"
          return $TRUE
      fi
done
}
```

```
#############################################################################
#  Name     : GetAction
#  Overview : The function allows users to enter an action (put or get)
#  Returns  : $TRUE  if a value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
#############################################################################
GetAction ()
{
tput clear
ACTION=""
while  true
do
         tput clear
         if [ "${FIRST_PASS}" = "${TRUE}"  ]
         then
            echo "Enter action ( put or get ) ( q to quit ):\c"
         else
            echo "Enter  action ( put or get ) or "
            echo "Press Enter to accept old value(${OLD_ACTION}) ( q to
quit ):\c"
         fi
         read  ACTION
         ACTION='echo "${ACTION}" | tr "A-Z" "a-z"'
         case  ${ACTION}  in
           q)   return $FALSE  ;;
     put|get)   return $TRUE ;;
          * ) if [ "${FIRST_PASS}"  = "${FALSE}" -a "${ACTION}"  = ""  ]
              then
                  ACTION="${OLD_ACTION}" ;
                  return $TRUE ;
              else
                    DisplayMessage E "${INVALID_ENTRY}"  N ;
              fi ;;
       esac
done
}
#############################################################################
#  Name     : GetComment
#  Overview : The function allows users to enter a comment.
#  Returns  : $TRUE  if a value entered
#             $FALSE if entered q to quit
#  Notes    : 1. For second or subsequent pass, user can enter the old
#                value by pressing return.
#############################################################################
GetComment ()
{
tput clear
COMMENT=""
```

```
while  [  -z "${COMMENT}"   ]
do
      tput clear
      if [ "${FIRST_PASS}" = "${TRUE}"  ]
      then
         echo "Enter comment for this entry ( q to quit ):\c"
      else
         echo "Enter  comment for this entry or "
         echo "Press Enter to accept old value(${OLD_COMMENT}) ( q to
quit ):\c"
      fi
      read  COMMENT
      if [ "${COMMENT}"  = "q" ]
      then
          return $FALSE
      elif [ "${COMMENT}" != ""  ]
      then
          return $TRUE
      elif [ "${FIRST_PASS}"  = "${FALSE}" -a "${COMMENT}"  = ""  ]
      then
          COMMENT="${OLD_COMMENT}"
          return $TRUE
      fi
done
}
###############################################################################
#  Name     : AddEntryToConfigFile
#  Overview : The function allows users to add an entry to configuration
#             file.
#  Notes    :1. Record entry structure is as follows
#               # Entry History
#               # -------------
#               # ${DATETIME} CReation
#               #
#             Remote_Host=<host>:Remote_Dir=<dir>:
#                                 Remote_File=<file>:Remote_Uid=<uid>:\
#             Remote_Pwd=<pwd>:Local_Dir=<dir>:
#                               Local_File=<file>:Action=<put or  get>:\
#              Comment=
#            2. Each entry takes up 6 lines
#            3. The function calls following functions:
#                          o GetRemoteHostName
#                          o GetRemoteDirectoryName
#                          o GetRemoteFileName
#                          o GetRemoteUserId
#                          o GetRemotePassword
#                          o GetLocalDirectoryName
#                          o GetLocalFileName
#                          o GetAction
#                          o GetComment
```

```
###############################################################################
AddEntryToConfigFile ()
{
# clear the flag for multi-pass
FIRST_PASS="${TRUE}"
while true
#{
do
   # get remote hostname
   if ! GetRemoteHostName
   then
       return $FALSE
   fi
   # get remote dir name
   if ! GetRemoteDirectoryName
   then
      return $FALSE
   fi
   # get remote file name
   if ! GetRemoteFileName
   then
      return $FALSE
   fi
   # get remote user id
   if ! GetRemoteUserId
   then
      return $FALSE
   fi
   #get remote password
   if ! GetRemotePassword
   then
      return $FALSE
   fi
   # get local dir name
   if ! GetLocalDirectoryName
   then
      return $FALSE
   fi
   # get local file name
   if ! GetLocalFileName
   then
      return $FALSE
   fi
   # get action
   if ! GetAction
   then
      return $FALSE
   fi
   # get comment
   if ! GetComment
```

```
   then
      return $FALSE
   fi
   # assemble configuration record
   CONFIG_RECORD="Remote_Host=${REMOTE_HOST}:Remote_Dir=${REMOTE_DIR}"
   CONFIG_RECORD="${CONFIG_RECORD}:Remote_File=${REMOTE_FILE}"
 CONFIG_RECORD="${CONFIG_RECORD}:Remote_Uid=${REMOTE_UID}:Remote_Pwd=${REMOTE_PWD}"
 CONFIG_RECORD="${CONFIG_RECORD}:Local_Dir=${LOCAL_DIR}:Local_File=${LOCAL_FILE}:
Action=${ACTION}:Comment=${COMMENT}"
   # get confirmation
   REPLY=""
   while true
   do
     tput clear
     echo "Configuration Record:${CONFIG_RECORD}"
     echo "Confirm configuration record(Y/N):\c"
     read REPLY
     case ${REPLY}  in
        Y)   break ;;
        N)   break ;;
        *)   :  ;;
     esac
   done
   if  [ "${REPLY}"  = "Y"  ]
   then
        break
   else
      # store previous values and allow  next pass
      FIRST_PASS=${FALSE}
      OLD_REMOTE_HOST="${REMOTE_HOST}"
      OLD_REMOTE_DIR="${REMOTE_DIR}"
      OLD_REMOTE_FILE="${REMOTE_FILE}"
      OLD_REMOTE_UID="${REMOTE_UID}"
      OLD_REMOTE_PWD="${REMOTE_PWD}"
      OLD_LOCAL_DIR="${LOCAL_DIR}"
      OLD_LOCAL_FILE="${LOCAL_FILE}"
      OLD_ACTION="${ACTION}"
      OLD_COMMENT="${COMMENT}"
   fi
done
#}
# check for configuration file existence
if [ ! -s  ${MFP_CONFIG_FILE}   ]
then
   # initialize the file with appropriate header
   CONFIG_FILE_HEADER="#\n# FTP  Configuration file\n# Do not Edit by
Hand\n# Use mfp.ksh to add/remove Entries\n# Apply Appropriate File
Protection to Safeguard Password Entries\n# #C# Records are commented
out\n# #D# Records are logically deleted\n# #D##C# Records are commented
out and then logically deleted\n#"
```

```
   echo "${CONFIG_FILE_HEADER}"  > ${MFP_CONFIG_FILE}
fi
# check for duplicate entries
if fgrep -x "${CONFIG_RECORD}" ${MFP_CONFIG_FILE}  > /dev/null  2>&1
then
     DisplayMessage  E "${ENTRY_EXISTS}" Y
     return $FALSE
fi
#Prepare Record Entry for the file
DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
RECORD_ENTRY="#\n# Entry History\n# ------------\n# ${DATETIME}
Creation\n#\n${CONFIG_RECORD}"
echo "${RECORD_ENTRY}"    >> ${MFP_CONFIG_FILE}
if  IsFTPDaemonRunning
then
    DisplayMessage I "${RELOAD_CONFIG_FILE}" N
fi
}
################################################################################
#  Name     : ModifyConfigFileEntry
#  Overview : The function allows users to remove, comment, and
#             uncomment an entry from configuration file
#  Parameter: C   ( to comment out entry )
#             D   ( to delete an entry   )
#             U   ( to uncomment an entry)
################################################################################
ModifyConfigFileEntry ()
{
#assign parameter
P_ACTION="${1}"
if [ ! -f ${MFP_CONFIG_FILE} ]
then
     DisplayMessage I "${CONFIG_FILE_NOT_EXISTS}"  Y
     return $FALSE
elif [ ! -s ${MFP_CONFIG_FILE} ]
then
     DisplayMessage I "${NO_CONFIG_FILE_ENTRIES}"  Y
     return $FALSE
fi
# prepare  header for list of values file
# header consists of 4 lines
echo  "Configuration File Entries on ${DATETIME}"   >  ${TEMP_FILE_1}
echo  "================================================" >>
${TEMP_FILE_1}
echo  "To Delete/Comment/Uncomment an Entry Delete Corresponding Line
and Save the File\n" >> \
                                                        ${TEMP_FILE_1}
LOV_FILE_NO_HEADER_LINES=4
# prepare a list of values file with  all entries
if  [ "${P_ACTION}" = "C"  ]
```

```
then
    # find all entries except  marked for commented and deletion
    grep "Remote_Host=" ${MFP_CONFIG_FILE} | grep -v "^#C#" | grep -v
"^#D#" >> ${TEMP_FILE_1}
elif  [ "${P_ACTION}" = "D"  ]
then
    # find all  entries except marked for deletion
    grep "Remote_Host=" ${MFP_CONFIG_FILE} | grep -v "^#D#" >>
${TEMP_FILE_1}
elif  [ "${P_ACTION}" = "U"  ]
then
    # find all commented out entries
    grep "^#C#" ${MFP_CONFIG_FILE}  >> ${TEMP_FILE_1}
else
    DisplayMessage E "${INVALID_ACTION_PARAM}"  N
    return $FALSE
fi
# does the list of values file have any entries
LOV_FILE_LINES='wc -l ${TEMP_FILE_1} | awk {'print $1'}'
if [ ${LOV_FILE_LINES} -le  ${LOV_FILE_NO_HEADER_LINES}   ]
then
   # no lov entries generated
   if  [ "${P_ACTION}" = "C"  ]
   then
       DisplayMessage I "${NO_ENTRY_TO_COMMENT_OUT}" Y
   elif  [ "${P_ACTION}" = "D"  ]
   then
       DisplayMessage I "${NO_ENTRY_TO_REMOVE}" Y
   elif  [ "${P_ACTION}" = "U"  ]
   then
       DisplayMessage I "${NO_ENTRY_TO_UNCOMMENT}" Y
   fi
   return $FALSE
fi
cp ${TEMP_FILE_1}  ${TEMP_FILE_2}
# allow user to mofify the file
vi ${TEMP_FILE_1}
# establish  the changed record id
CONFIG_RECORD=""
CONFIG_RECORD="'diff  ${TEMP_FILE_1}  ${TEMP_FILE_2} | tail -1 | awk
{'print $2'}'"
# prepare bad config record which is the ouput from diff without > sign
BAD_CONFIG_RECORD="'diff  ${TEMP_FILE_1}  ${TEMP_FILE_2} | tail -1 | sed
s/'>'//'"
# has the user selected any record ?
if [ "${CONFIG_RECORD}" = ""  ]
then
    DisplayMessage I "${NO_CHANGE}" N
    return $FALSE
elif  ! echo  "${CONFIG_RECORD}" | grep  "Remote_Host" > /dev/null 2>&1
```

```
then
    # bad configuration record (one of the header lines) selected
    # format the record for displaying
    BAD_CONFIG_RECORD='echo "${BAD_CONFIG_RECORD}" | sed s/'>'//g | sed
s/'\\/'/'\\/'/g'
    DisplayMessage E "${INVALID_CONFIG_RECORD}"  N
    return $FALSE
fi
# re-work the original configuration  file
cp  ${MFP_CONFIG_FILE}  ${TEMP_FILE_1}
if  [ "${P_ACTION}" = "C"  ]
then
   # comment out this entry
   # reformat record (add escape characters for forward slash) for sed
   FORMATTED_CONFIG_RECORD='echo "${CONFIG_RECORD}" | sed s/'\/'/'\\\\\/
'/g'
   sed s/"${FORMATTED_CONFIG_RECORD}"/"#C#${FORMATTED_CONFIG_RECORD}"/
${TEMP_FILE_1} \
                                                    >${MFP_CONFIG_FILE}
   # update history
   # get line number for the entry
   ENTRY_LINE_NO='fgrep -xn "#C#${CONFIG_RECORD}" ${MFP_CONFIG_FILE} |
cut -d':' -f1'
   #  history record should be two lines above $ENTRY-LINE_NO
   #  because we're using i command for ed, we need the
   #  line number for one line   above
   HISTORY_REC_LINE_NO='expr ${ENTRY_LINE_NO} - 1'
   DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
   HISTORY_REC="# ${DATETIME} Entry commented out"
   ed <<! ${MFP_CONFIG_FILE} > /dev/null
${HISTORY_REC_LINE_NO}i
${HISTORY_REC}
.
w
q
!
elif  [ "${P_ACTION}" = "D"  ]
then
   # logically delete this entry by prefixing the record with #D#
   # but check its existence first
   # reformat record ( add escape characters for forward slash ) for sed
   FORMATTED_CONFIG_RECORD='echo "${CONFIG_RECORD}" | sed s/'\/'/'\\\\\/
'/g'
   sed s/"${FORMATTED_CONFIG_RECORD}"/"#D#${FORMATTED_CONFIG_RECORD}"/
${TEMP_FILE_1} \
                                        >${MFP_CONFIG_FILE}
   # update history
   # get line number for the entry
   ENTRY_LINE_NO='fgrep -xn "#D#${CONFIG_RECORD}" ${MFP_CONFIG_FILE} |
cut -d':' -f1'
```

```
   #  history record should be two lines above $ENTRY_INE_NO
   #  because we're using i command for ed, we need the
   #  line number  for one line   above
   HISTORY_REC_LINE_NO='expr ${ENTRY_LINE_NO} - 1'
   DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
   HISTORY_REC="# ${DATETIME} Entry deleted"
   ed <<! ${MFP_CONFIG_FILE} > /dev/null
${HISTORY_REC_LINE_NO}i
${HISTORY_REC}
.
w
q
!
elif  [ "${P_ACTION}" = "U"  ]
then
   # logically delete this entry by prefixing the record with #D#
   # but check its existence  first
   # uncomment this entry
   UNCOMMENTED_CONFIG_RECORD='echo "${CONFIG_RECORD}" |  sed  s/"#C#"//'
   # check for creation against duplicate entry
   if fgrep -x "${UNCOMMENTED_CONFIG_RECORD}"  ${TEMP_FILE_1}  > /dev/
null  2>&1
   then
        DisplayMessage E "${ENTRY_EXISTS}" Y
        return $FALSE
   fi
   # reformat record ( add escape characters for forward slash ) for sed
   FORMATTED_CONFIG_RECORD='echo "${CONFIG_RECORD}" | sed s/'\/'/'\\\\\/
'/g'
   FORMATTED_UNCOMMENTED_CONFIG_RECORD='echo
"${UNCOMMENTED_CONFIG_RECORD}" | \
                                              sed s/'\/'/'\\\\\/'/g'
   sed s/"${FORMATTED_CONFIG_RECORD}"/
"${FORMATTED_UNCOMMENTED_CONFIG_RECORD}"/ \
                               ${TEMP_FILE_1} >${MFP_CONFIG_FILE}
   # update history
   # get line number for the entry
   ENTRY_LINE_NO='fgrep -xn "${UNCOMMENTED_CONFIG_RECORD}"
${MFP_CONFIG_FILE} | cut -d':' -f1'
   #  history record should be two lines above $ENTRY_LINE_NO
   #  because we're using i command for ed, we need the
   #  line number for one line  above
   HISTORY_REC_LINE_NO='expr ${ENTRY_LINE_NO} - 1'
   DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
   HISTORY_REC="# ${DATETIME} Entry uncommented"
   ed <<! ${MFP_CONFIG_FILE} > /dev/null
${HISTORY_REC_LINE_NO}i
${HISTORY_REC}
.
w
```

```
q
!
fi
if  IsFTPDaemonRunning
then
    DisplayMessage I "${RELOAD_CONFIG_FILE}" N
fi
}
################################################################################
#  Name     : ProcessMenuOption
#  Overview : The function processes selected option.
################################################################################
ProcessMenuOption  ()
{
case  $OPTION  in
       5)   StartFTPDaemon    ;;
      10)   StopFTPDaemon     ;;
      15)   KillFTPDaemon     ;;
      20)   ViewConfigFile    ;;
      25)   AddEntryToConfigFile  ;;
      30)   ModifyConfigFileEntry "D" ;;
      35)   ModifyConfigFileEntry "C" ;;
      40)   ModifyConfigFileEntry "U" ;;
      45)   ReplaceEntryInConfigFile    ;;
      50)   PurgeDeletedEntriesFromConfigFile ;;
      55)   ReloadConfigurationFile  ;;
      60)   AdjustFtpInterval   ;;
      65)   SwitchOnDebug  ;;
      70)   SwitchOffDebug  ;;
      75)   ViewLogFile    ;;
      80)   TrimLogFile  ;;
      85)   ShowRemoteServerStatus   ;;
      99)   ProcessExit  $SEC ;;
      * ) DisplayMessage E "${INVALID_ENTRY}" ;;
esac
}
################################################################################
#  Name     : PerformSanityCheck
#  Overview : The function performs sanity check on expected variable
#             settings and executables and directories.
################################################################################
PerformSanityCheck ()
{
# scrpt root directory
if [  -z "${MFP_ROOT_DIR}" ]
then
    VAR="\${MFP_ROOT_DIR}"
    DisplayMessage E "${VARIABLE_NOT_SET}"  N
    return $FALSE
fi
```

```
if [ ! -d  "${MFP_ROOT_DIR}" ]
then
    DIR="${MFP_ROOT_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}"  N
    return $FALSE
fi
if [  -z "${MFP_LOG_DIR}" ]
then
    VAR="\${MFP_LOG_DIR}"
    DisplayMessage E "${VARIABLE_NOT_SET}"  N
    return $FALSE
fi
if [ ! -d  "${MFP_LOG_DIR}" ]
then
    DIR="${MFP_LOG_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}"  N
    return $FALSE
fi
if [  -z "${MFP_CONFIG_DIR}" ]
then
    VAR="\${MFP_CONFIG_DIR}"
    DisplayMessage E "${VARIABLE_NOT_SET}"  N
    return $FALSE
fi
if [ ! -d  "${MFP_CONFIG_DIR}" ]
then
    DIR="${MFP_CONFIG_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}"  N
    return $FALSE
fi
if [  -z "${MFP_SCRIPT_DIR}" ]
then
    VAR="\${MFP_SCRIPT_DIR}"
    DisplayMessage E "${VARIABLE_NOT_SET}"  N
    return $FALSE
fi
if [ ! -d  "${MFP_SCRIPT_DIR}" ]
then
    DIR="${MFP_SCRIPT_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}"  N
    return $FALSE
fi
if [ !  -f  "${MFP_SCRIPT_DIR}/pfari.ksh" ]
then
    SCRIPT="${MFP_SCRIPT_DIR}/pfari.ksh"
    DisplayMessage E "${SCRIPT_NOT_FOUND}"  N
    return $FALSE
fi
if [  -z "${MFP_TEMP_DIR}" ]
then
```

```
    VAR="\${MFP_TEMP_DIR}"
    DisplayMessage E "${VARIABLE_NOT_SET}"  N
    return $FALSE
fi
if [ ! -d  "${MFP_TEMP_DIR}" ]
then
    DIR="${MFP_TEMP_DIR}"
    DisplayMessage E "${DIR_NOT_EXIST}"  N
    return $FALSE
fi
return $TRUE
}
#############################################################################
#  Name     : IsFTPDaemonRunning
#  Overview : The function establishes whether or not the ftp daemon
#             script is running.
#  Returns  : $TRUE if daemon script is running
#             $FALSE  otherwise
#############################################################################
IsFTPDaemonRunning ()
{
if  ps -eaf | grep -v "grep" | grep "${MFP_SCRIPT_DIR}/pfari.ksh" > /
dev/null 2>&1
then
    return $TRUE
else
    return $FALSE
fi
}
#############################################################################
#  Name     : InstanceCheck
#  Overview : The function establishes whether or not more than one
#             instance of this script is running.
#  Returns  : $TRUE if the script is not running
#             $FALSE  otherwise
#############################################################################
InstanceCheck ()
{
ps -eaf | grep "mfp.ksh" | grep -v "grep" | wc -l > ${TEMP_FILE_1}
NO_OF_INSTANCE='cat ${TEMP_FILE_1}'
if [ $NO_OF_INSTANCE -gt 1 ]
then
      DisplayMessage E "${INSTANCE_RUNNING}" N
      return $FALSE
else
      return $TRUE
fi
}
#############################################################################
#  Name     : main
```

```
#  Overview : The function implements processing structure.
#  Notes    : The function calls the following functions:
#                     o InitialiseVariables
#                     o InitialiseLogFile
#                     o PerformSanityCheck
#                     o DisplayMenu
#                     o ProcessMenuOption
#                     o InstanceCheck
################################################################################
main ()
{
# set umask
umask Ø77
InitialiseVariables
if [ "${USER}" != "${AUTHORISED_USER}"  ]
then
     DisplayMessage E "${NOT_AUTHORISED_USER}"  N
     return $FALSE
fi
if ! PerformSanityCheck
then
    return $FALSE
fi
InitialiseLogFile
if ! InstanceCheck
then
    return $FALSE
fi
while true
do
   DisplayMenu
   ProcessMenuOption
done
}
# invoke main ()
# define traps
trap "HandleInterrupt "   INT  TERM  TSTP
main
```

PFARI.KSH

```
#! /usr/bin/ksh
################################################################################
#   Name     : pfari.ksh (peform ftp at regular interval)
#   Overview : The shell script performs ftp at regular intervals using
#              data from configuration file, ${MFP_CONFIG_FILE}. The
#              script is started at the background by master script,
#              mfp.ksh.
#   Notes    : 1.  The  script contains the following functions:
```

```
#                   o InitialiseVariables
#                   o ParseCommandLine
#                   o LogMessage
#                   o ProcessExit
#                   o PerformFtp
#                   o HandleTermination
#                   o SetDebug
#                   o UnsetDebug
#                   o ReloadConfigFile
#                   o AdjustFTPInterval
#                   o PrepareFtpScript
#                   o ExecuteFTPProcess
#                   o main
#              2. Host server is defined as the server on which this
#                 script is running.
#           3. The following exported variables are used in this script:
#                       $MFP_LOG_FILE
#                       $MFP_CONFIG_FILE
#                       $MFP_TEMP_DIR
#                       $MFP_FTP_INTERVAL_FILE
#                       $MFP_SCRIPT_DIR
#              4. The script  must be started by master script mfp.ksh
################################################################################
#  Name     : InitialiseVariable
#  Overview : The function initializes all module constants and
#             working variables.
################################################################################
InitialiseVariables  ()
{
# time interval in seconds, between each  pass of ftp for all
# configuration records. This can be adjusted from master script mfp.ksh
FTP_INTERVAL=900  # 15 minutes
# when config file is missing or no records found in configuration file,
# process just waits forever, logging appropriate message
WAIT_ON_CONFIG_INFO=60
# variables for ftp parameters
INDEX=0
REMOTE_HOST_NAMES[$INDEX]=""
REMOTE_DIRS[$INDEX]=""
REMOTE_FILES[$INDEX]=""
REMOTE_UIDS[$INDEX]=""
REMOTE_PWDS[$INDEX]=""
LOCAL_DIRS[$INDEX]=""
LOCAL_FILES[$INDEX]=""
ACTIONS[$INDEX]=""
COMMENTS[$INDEX]=""
# no of configuration records
NO_CONFIG_RECORDS=0
#  sccs  command directory
SCCS_BIN_DIR="/usr/ccs/bin"
```

```
#USER='id  | cut -d'(' -f2 | cut -d')' -f1'
# terminal capabilities
BOLDON='tput smso'
BOLDOFF='tput rmso'
ESC="\ØØ33["
ERROR="pfari.ksh:ERROR:"
INFO="pfari.ksh:INFO:"
# fuction return values
TRUE=Ø
FALSE=1
# exit status
SEC=Ø
FEC=1
# sleep while  for mfp.ksh is logging a message
MFP_LOGGING_INTERVAL=1Ø
# background sleep  process  id
SLEEP_PROC_ID=""
# these variables are set when appropriate
# signals are received
STOP_DAEMON=$FALSE
DEBUG=$FALSE
RELOAD_CONFIG_FILE=${FALSE}
# define  signals
SIGINT=2    ;  export SIGINT     # ctrl-c command
SIGTERM=15  ;  export SIGTERM    # kill  command
SIGTSTP=18  ;  export SIGTSTP    # ctrl-z  command
# temporary file
TEMP_FILE_1="${MFP_TEMP_DIR}/pfari_$$_1.tmp"
TEMP_FILE_2="${MFP_TEMP_DIR}/pfari_$$_2.tmp"
FTP_SCRIPT_FILE="${MFP_TEMP_DIR}/pfari_$$_ftp.dat"
FTP_RESULT_FILE="${MFP_TEMP_DIR}/pfari_$$_ftp.result"
# messages
INTERRUPT="Program Interrupted\; daemon will exit after all ftps
completed"
PREPARING_SCRIPT="Preparing script file"
WORKING="Working"
OS_ERROR="\${ERR_MSG}"
DEBUG_SET="Debug set"
DEBUG_UNSET="Debug unset"
FTP_INTERVAL_RESET="Ftp interval has been reset to  \${FTP_INTERVAL}
second\(s\)"
RELOADING_CONFIG_FILE="Reloading configuration file"
PERFORMING_FTP="Performing FTP for configuration record
\${CONFIG_RECORDS[\$INDEX]}"
EXECUTING_FTP="Executing ftp command"
FTP_OK="Successfully performed FTP"
FTP_FAILED="Failed to perform FTP"
NO_CONFIG_RECORDS_FOUND="No configuration records found"
NO_CONFIG_FILE="Configuration file, \${MFP_CONFIG_FILE} does not exist"
SLEEPING="Sleeping for \${FTP_INTERVAL}"
```

```
}
################################################################################
#  Name     : ReadConfigurationFile
#  Overview : The function reads the configuration file into memory.
#  Notes    : 1. The configuration file structure is :
#              Remote_Host:Remote_Dir:Remote_File:Remote_Uid:
#                                                        Remote_Pwd:\
#                Local_Dir:Local_File:Action:Comment
#             2. Any line beginning with # is treated as comment
################################################################################
ReadConfigurationFile  ()
{
trap "HandleTermination " TERM
trap "SetDebug          " USR1
trap "UnsetDebug        " USR2
trap "ReloadConfigFile  " PIPE
trap "AdjustFTPInterval " EMT
# look for configuration file
while true
do
  # if  termination signal  is  received
  # exit  the  program
  if  [  "${STOP_DAEMON}" =  "${TRUE}"  ]
  then
         ProcessExit  "${FEC}"
  fi
  if  [ ! -s  ${MFP_CONFIG_FILE} ]
  then
     LogMessage E "${NO_CONFIG_FILE}"
     echo "\n"  >> ${MFP_LOG_FILE}
     sleep  ${WAIT_ON_CONFIG_INFO} &
     SLEEP_PROC_ID=$!
     wait ${SLEEP_PROC_ID}
  else
     break
  fi
done
# read  configuration file entries into memory
while true
do
  NO_CONFIG_RECORDS=Ø
  INDEX=Ø
  cat ${MFP_CONFIG_FILE}  | while read  CONFIG_RECORD
  do
     FIRST_CHAR='echo "${CONFIG_RECORD}" | cut -c1-1'
     if [  "${FIRST_CHAR}" = "#"  ]
     then
        continue
     fi
     CONFIG_RECORDS[$INDEX]="${CONFIG_RECORD}"
```

```
     REMOTE_HOST_NAMES[$INDEX]='echo "${CONFIG_RECORD}" | cut -d':' -f1
| cut -d'=' -f2'
     REMOTE_DIRS[$INDEX]='echo       "${CONFIG_RECORD}" | cut -d':' -f2
| cut -d'=' -f2'
     REMOTE_FILES[$INDEX]='echo      "${CONFIG_RECORD}" | cut -d':' -f3
| cut -d'=' -f2'
     REMOTE_UIDS[$INDEX]='echo       "${CONFIG_RECORD}" | cut -d':' -f4
| cut -d'=' -f2'
     REMOTE_PWDS[$INDEX]='echo       "${CONFIG_RECORD}" | cut -d':' -f5
| cut -d'=' -f2'
     LOCAL_DIRS[$INDEX]='echo        "${CONFIG_RECORD}" | cut -d':' -f6
| cut -d'=' -f2'
     LOCAL_FILES[$INDEX]='echo       "${CONFIG_RECORD}" | cut -d':' -f7
| cut -d'=' -f2'
     ACTIONS[$INDEX]='echo           "${CONFIG_RECORD}" | cut -d':' -f8
| cut -d'=' -f2'
     COMMENTS[$INDEX]='echo          "${CONFIG_RECORD}" | cut -d':' -f9
| cut -d'=' -f2'
    INDEX='expr $INDEX + 1'
    NO_CONFIG_RECORDS=$INDEX
  done
  if [ $NO_CONFIG_RECORDS -eq  Ø ]
  then
      LogMessage E "${NO_CONFIG_RECORDS_FOUND}"
      sleep ${WAIT_ON_CONFIG_INFO} &
      SLEEP_PROC_ID=$!
      wait ${SLEEP_PROC_ID}
  else
      break
  fi
done
}
################################################################################
#  Name     : HandleTermination
#  Overview : The function processes signal TERM
#  Notes    :
################################################################################
HandleTermination ()
{
LogMessage I "${INTERRUPT}"
STOP_DAEMON=$TRUE
}
################################################################################
#  Name     : SetDebug
#  Overview : The function processes signal USR1.
#  Notes    :
################################################################################
SetDebug ()
{
LogMessage I "${DEBUG_SET}"
```

```
DEBUG=${TRUE}
}
################################################################################
#  Name     : UnsetDebug
#  Overview : The function processes signal USR2.
################################################################################
UnsetDebug ()
{
LogMessage I "${DEBUG_UNSET}"
DEBUG=${FALSE}
}
################################################################################
#  Name     : ReloadConfigFile
#  Overview : The function processes signal PIPE.
################################################################################
ReloadConfigFile ()
{
LogMessage I "${RELOADING_CONFIG_FILE}"
# prematurely end the background sleep process in order
# to resume activity
kill -15 ${SLEEP_PROC_ID}  > /dev/null  2>&1
RELOAD_CONFIG_FILE=${TRUE}
}
################################################################################
#  Name     : AdjustFTPInterval
#  Overview : The function processes signal EMT.
################################################################################
AdjustFTPInterval ()
{
FTP_INTERVAL='cat ${MFP_FTP_INTERVAL_FILE}'
LogMessage I "${FTP_INTERVAL_RESET}"
}
################################################################################
#  Name     : ProcessExit
#  Overview : The function calls ProcessExit.
#  Input    : Exit Code
################################################################################
ProcessExit ()
{
# assign parameter
EXIT_CODE="$1"
rm -f ${TEMP_FILE_1}
rm -f ${TEMP_FILE_2}
rm -f ${DEBUG_FILE}
rm -f ${FTP_SCRIPT_FILE}
rm -f ${FTP_RESULT_FILE}
exit ${EXIT_CODE}
}
################################################################################
#  Name     : LogMessage
```

```
#  Overview : The function logs a message into log file
#  Input    : 1. Message type (E = Error, I = Informative)
#             2. Error Code as defined in DefineMessages ().
###############################################################################
LogMessage ( )
{
trap "HandleTermination " TERM
MESSAGE_TYPE=$1
MESSAGE_TEXT='eval echo $2'
DATETIME='date "+%d/%m/%Y %H:%M:%S"'
if [ "${MESSAGE_TYPE}" = "E" ]
then
    echo  "${ERROR}${DATETIME}:${MESSAGE_TEXT}"  >>  ${MFP_LOG_FILE}
else
    echo  "${INFO}${DATETIME}:${MESSAGE_TEXT}"   >>  ${MFP_LOG_FILE}
fi
}
###############################################################################
#  Name     : PerformFTP
#  Overview : The function performs  FTP .
###############################################################################
PerformFTP ()
{
LogMessage I  "${EXECUTING_FTP}"
ftp -n < ${FTP_SCRIPT_FILE}  > ${FTP_RESULT_FILE} 2>&1
}
###############################################################################
#  Name     : PrepareFtpScript
#  Overview : The function prepares FTP script file, ${FTP_SCRIPT_FILE}
#             using information from a specific configuration record
#             pointed to by variable INDEX in the arrays.
#  Notes    : 1. The content of script should look as follows:
#                   open   <hostname>
#                   user   <user pwd>
#                   cd     <target_dir>
#                   lcd    <host_dir>
#                   get    <file name>  or
#                   put    <file name>
#                   bye
#            2. Cannot add any comments before the command in ftp script
###############################################################################
PrepareFtpScript ()
{
LogMessage I  "${PREPARING_SCRIPT}"
echo "open ${REMOTE_HOST_NAMES[${INDEX}]} "            >
${FTP_SCRIPT_FILE}
echo "verbose"                                        >>
${FTP_SCRIPT_FILE}
echo "user ${REMOTE_UIDS[${INDEX}]} ${REMOTE_PWDS[${INDEX}]}" >>
${FTP_SCRIPT_FILE}
```

```
echo "cd   ${REMOTE_DIRS[${INDEX}]}"                        >>
${FTP_SCRIPT_FILE}
echo "lcd  ${LOCAL_DIRS[${INDEX}]} "                        >>
${FTP_SCRIPT_FILE}
# check DEBUG flag
if  [ "${DEBUG}"  = "${TRUE}"  ]
then
      echo "hash"  >> ${FTP_SCRIPT_FILE}
      echo "debug" >> ${FTP_SCRIPT_FILE}
fi
if [ "${ACTIONS[${INDEX}]}" = "put" ]
then
    echo "put  ${LOCAL_FILES[${INDEX}]} ${REMOTE_FILES[${INDEX}]}" >> \
                                                          ${FTP_SCRIPT_FILE}
else
   echo "get  ${REMOTE_FILES[${INDEX}]} ${LOCAL_FILES[${INDEX}]}"   >> \
                                                          ${FTP_SCRIPT_FILE}
fi
echo "bye "                                                 >>
${FTP_SCRIPT_FILE}
}
##############################################################################
#  Name     : ExecuteFTPProcess
#  Overview : The function performs FTP for each configuration record.
#             Once all the configuration records have been processed,
#             the function sleeps for ${FTP_INTERVAL} seconds and
#             starts performing FTP all over again.
##############################################################################
ExecuteFTPProcess ()
{
# set traps
trap "HandleTermination " TERM
trap "SetDebug          " USR1
trap "UnsetDebug        " USR2
trap "ReloadConfigFile  " PIPE
trap "AdjustFTPInterval " EMT
# set this variable to $TRUE for first loading of
# configuration file; subsequently, ReloadConfigFile() will
# set this to $TRUE if required signal is received
RELOAD_CONFIG_FILE=${TRUE}
INDEX=Ø
while  true
do
#{
   #   check to see if config file needs reloading
   if  [  "${RELOAD_CONFIG_FILE}" = "${TRUE}" ]
   then
        RELOAD_CONFIG_FILE=${FALSE}
        ReadConfigurationFile
        # re-initialize index to zero to start all over
```

```
        INDEX=Ø
    fi
    # log message
    LogMessage I  "${PERFORMING_FTP}"
    #  prepare ftp script
    PrepareFtpScript
    # append ftp script file to log file for debug
    if  [ "${DEBUG}"  = "${TRUE}"  ]
    then
        echo "FTP Script File"  >> ${MFP_LOG_FILE}
        echo "==============="  >> ${MFP_LOG_FILE}
        cat  ${FTP_SCRIPT_FILE} >> ${MFP_LOG_FILE}
    fi
    # do ftp
    PerformFTP
    if  grep "Transfer complete" ${FTP_RESULT_FILE}  > /dev/null  2>&1
    then
        LogMessage I  "${FTP_OK}"
    else
        LogMessage E  "${FTP_FAILED}"
        #  append FTP result file to log file
        cat  ${FTP_RESULT_FILE}  >>  ${MFP_LOG_FILE}
    fi
    # print a line in the lo file to  separate log records
    echo "\n"  >> ${MFP_LOG_FILE}
    INDEX='expr $INDEX + 1'
    if [  ${INDEX} -eq  ${NO_CONFIG_RECORDS}  ]
    then
        # all ftps are complete
        # if  termination signal is received
        # exit the program
        if  [  "${STOP_DAEMON}" =  "${TRUE}"  ]
        then
            ProcessExit  "${FEC}"
        fi
        LogMessage I  "${SLEEPING}"
        echo "\n"  >> ${MFP_LOG_FILE}
        sleep  ${FTP_INTERVAL} &
        SLEEP_PROC_ID=$!
        wait ${SLEEP_PROC_ID}
        # re-initialize index couner
        INDEX=Ø
    fi
done
#}
}
################################################################################
#  Name     : main
#  Overview : The function implements processing structure.
#  Notes    : The function calls the following functions:
```

```
#                       o InitialiseVariables
#                       o ExecuteFTPProcess
################################################################################
main ()
{
InitialiseVariables
# give  mfp.ksh an opportunity to  log  a  message
sleep ${MFP_LOGGING_INTERVAL}
InitialiseVariables
ExecuteFTPProcess
}
# invoke main
# define traps
trap "HandleTermination "   TERM
# package command  line
ARGC="$#"
ARGV="$@"
main
```

Arif Zaman
DBA/Developer (UK)

Date manipulation

The following script aims to address issues that exist because Unix is not very user-friendly regarding date manipulation. Some of my colleagues previously used COBOL and told me that it was very easy to use the date/time manipulation utilities. After due consideration I produced the following script, which addresses the calculation of dates. The script can display the date in several formats, the default being DDMMYY.

One area in which I have seen people struggling with date manipulation is housekeeping routines, where it is necessary to know the previous day's date. This is quite difficult if the present date is the first day of the month. There is a lengthy process of saying that today is 01, then, depending on what the current month is, that yesterday's date is month –1. Then you need to use a look-up table to find out how many days were in the previous month. This then leads to considering whether the

month is January, in which case the year will be different … etc (also, of course, not forgetting to check for leap years). This is very cumbersome and a lot of code is required.

My script uses the epoch (the number of days since the start of 1970) to calculate days and as such always displays the correct day/month with very little coding. A lot of the coding is for additional features – see below.

To calculate the date using the epoch I use a C program */usr/local/bin/now_epoch* written by a colleague of mine to obtain the epoch as of today. The script then uses the C program */usr/local/bin/epoch2date* to translate the epoch back to a readable date. The *epoch2date* program was obtained from an article that was published in *AIX Update*, issue 26, January 1998, pages 46–47, entitled 'Timestamp to date'. The source is listed later in this article.

Calculations are based on how many days you wish to add to or subtract from the epoch to get the manipulated date. For example if today is 26/09/03 and you wish to find the date seven days from now simply type:

```
/usr/local/bin/calc_date –v"+7"
```

This will display:

```
031003
```

If you wish to calculate the date seven days ago, using a forward slash as a field separator to get the format DD/MM/YY, type:

```
/usr/local/bin/calc_date –v"-7" –s /
```

This will display:

```
19/09/03
```

Also, if you wish to use an American date format instead of English type:

```
/usr/local/bin/calc_date –v"-7" –s / -u
```

This will display:

```
09/19/03
```

If you wish to use a four-digit year field to get the format DD/MM/YYYY type:

```
/usr/local/bin/calc_date –v"-7" –s / -y
```

This will display:

```
19/09/2003
```

The script has also been written for simple use, whereby the creation of hard links enables calculations for yesterday, today, and tomorrow.

To hard-link these files:

```
cd /usr/local/bin
ln calc_date yesterday
ln calc_date today
ln calc_date tomorrow
```

To obtain the date for yesterday (if today is 26/09/03) in a DD/MM/YY format:

```
/usr/local/bin/yesterday –s /
```

This will display:

```
25/09/03
```

To obtain the date for tomorrow in a DD.MM.YYYY format:

```
/usr/local/bin/tomorrow –s . -y
```

This will display:

```
25.09.2003
```

To obtain today's date in a DDMMYY format:

```
/usr/local/bin/today
```

This will display:

```
260903
```

All flags will work with all four commands, with the exception of the **-v** flag, which is ignored by the today, tomorrow, and yesterday commands.

There is a help function that is invoked by entering:

```
/usr/local/bin calc_date –h
```

I am sure that there are other uses/additional features that could be inserted into the script for other users' environments. This is the reason why I used the shell command **getopts** to handle command line arguments and options.

For ease of use of the **getopts** command, I discard all errors reported and have the case statement immediately after the call to **getopts** to report any incorrect options entered.

The code for the script is:

```
#!/bin/ksh
#Script Name  : calc_date/today/tomorrow/yesterday
#Script Version    : 1.5
#Written By   : Stewart Robinson
#Cron Time         : n/a
#Description  : Allows for date manipulation
#    : Also hard linked are today/tomorrow and yesterday
#Usage    : calc_date +/- no of days
#Pre-reqs : /usr/local/bin/now_epoch
#    : /usr/local/bin/epoch2date
#Amendments        : V1.1 SR 17/Ø9/Ø3
#
#*****************************************************************************
#declare functions

NOEND=Ø
usage(){
echo "\nUsage: $Ø -v \"+/-\" <number of days> -h -s <separator> -y 'u\n"
if [ $NOEND != 1 ]
then
exit 1
fi
}

display_help_file(){
NOEND=1

usage
echo "\nParameters are:\n"
echo "\t-v <value of days to add/subtract>"
echo "\t-s <date format separator>"
echo "\t-y    Display year field as YYYY instead of default YY."
echo "\t-h    This help file"
echo "\t-u    Display date in US format (Transpose DD/MM) "
```

```
echo "\nPress Enter for examples"
read RESP
echo "\nExamples of how this program works are:"
echo "\n\t$Ø -v \"+2\" -y -s:"
echo "\nOn Monday 22nd September 2ØØ3 will give the output of:"
echo "\n\t24:Ø9:2ØØ3"
echo "\nAnd \t$Ø -v \"-7\" -s /"
echo "\nOn Monday 22nd September 2ØØ3 will give the output of:"
echo "\n\t15/Ø9/2ØØ3"
echo "\nAnd \t$Ø -v \"-7\" -s / -u"
echo "\nOn Monday 22nd September 2ØØ3 will give the output of:"
echo "\n\tØ9/15/2ØØ3\n"

exit Ø
}

gettoday(){
TDEPOCH='/usr/locl/bin/now_epoch'
TDDATE='date +%d%m%Y'
TDDAY='echo $TDDATE | cut -c1-2'
TDMON='echo $TDDATE | cut -c3-4'
TDY='echo $TDDATE | cut -c7-8'
TDYY='echo $TDDATE | cut -c5-8'
}

convert_epoch_to_ddmmyy(){
EPOCHTOCONVERT=$1
EPOCHSTRING='/usr/local/bin/epoch2date $EPOCHTOCONVERT'
echo $EPOCHSTRING |awk '{print $2" "$3" "$5}' | read EPOCHMON EPOCHDAY
EPOCHYY
EPOCHY='echo $EPOCHYY | cut -c3-4'
#Line below for future use ' maybe you wish to use words instead of
#numbers to represent the month???
if [ x$MFLAG = x ]
then
case $EPOCHMON in
        Jan) EPOCHMON=Ø1 ;;
        Feb) EPOCHMON=Ø2 ;;
        Mar) EPOCHMON=Ø3 ;;
        Apr) EPOCHMON=Ø4 ;;
        May) EPOCHMON=Ø5 ;;
        Jun) EPOCHMON=Ø6 ;;
        Jul) EPOCHMON=Ø7 ;;
        Aug) EPOCHMON=Ø8 ;;
        Sep) EPOCHMON=Ø9 ;;
        Oct) EPOCHMON=1Ø ;;
        Nov) EPOCHMON=11 ;;
        Dec) EPOCHMON=12 ;;
esac
fi
```

```
if [ $EPOCHDAY -le 9 ]
then
        EPOCHDAY="Ø"$EPOCHDAY
fi


if [ $UFLAG = 1 ]
then
        TEMPDAY=$EPOCHDAY
        EPOCHDAY=$EPOCHMON
        EPOCHMON=$TEMPDAY
fi

if [ $YYFLAG = 1 -a $SFLAG = 1 ]
then
        echo $EPOCHDAY$SEPARATOR$EPOCHMON$SEPARATOR$EPOCHYY
elif [ $YYFLAG = 1 -a $SFLAG != 1 ]
then
        echo $EPOCHDAY$EPOCHMON$EPOCHYY
elif [ $SFLAG = 1 ]
then
        echo $EPOCHDAY$SEPARATOR$EPOCHMON$SEPARATOR$EPOCHY
else
        echo $EPOCHDAY$EPOCHMON$EPOCHY
fi
}

getparams(){
PARAMS=$1
PLUSMINUS='echo $1 | cut -c1'
NUMOFDAYS='echo $1 | cut -c2-'
DAYSINSECS='echo "$NUMOFDAYS * 864ØØ" | bc'
}

manipepoch(){
MEPOCH='echo "$TDEPOCH $PLUSMINUS $DAYSINSECS" |bc'
}

#######################################
#start of script
#######################################
NAME='basename $Ø'
YYFLAG=Ø
SFLAG=Ø
UFLAG=Ø
VAL="+Ø"
while getopts ydms:huv: OPTION 2>>/dev/null
do
case $OPTION in
```

```
        y) YYFLAG=1 ;;
        d) DFLAG=1 ;;
        m) MFLAG=1 ;;
        s) SEPARATOR=$OPTARG
                SFLAG=1 ;;
        h) display_help_file ;;
        v) VAL=$OPTARG ;;
        u) UFLAG=1 ;;
  *) echo "\nIncorrect option"
         usage ;;
esac
done

case $NAME in
        calc_date) ARG=$VAL  ;;
        yesterday) ARG="-1" ;;
        today) ARG="-Ø" ;;
        tomorrow) ARG="+1" ;;
esac

getparams $ARG
gettoday
manipepoch
convert_epoch_to_ddmmyy $MEPOCH
```

The C source code for *now_epoch* is:

```
/*
 * FUNCTION:
 * Displays epoch time
 *
 * SYNTAX:
 * epoch
 *
 * ARGUMENTS:
 * none
 *
 * RETURNS:
 * The time in seconds since start of 197Ø
 * Compile with:
 * cc -o now_epoch now_epoch.c
 */

#include <time.h>
#include        <stdio.h>
#include        <stdlib.h>
#include        <sys/types.h>
#include        <sys/timeb.h>
#include        <sys/time.h>
#include        <sys/types.h>
```

```
main() {

int rc;
struct timeb timeb;

rc = ftime(&timeb);
printf( "%d\n", timeb.time);

}
```

The source code for *epoch2date* is:

```
/*
 * FUNCTION:
 * Convert epoch timestamp to human-readable date
 *
 * SYNTAX:
 * epoch2date timestamp
 *
 * ARGUMENTS:
 * epoch time (the number of seconds since 1970)
 *
 * RETURNS:
 * The date in the format: www mmm dd hh:mm:ss yyyy
 *     www  weekday
 *     mmm  month
 *     dd   day
 *     hh   hour
 *     mm   minute
 *     ss   second
 *     yyyy year
 *
 * DESCRIPTION:
 * This program converts an epoch timestamp (the number of seconds
 * since the start of 1970) and returns the date in a human-readable
 * format.
 *
 * Compile with:
 * cc -o epoch2date epoch2date.c
 */

#include <time.h>

#include        <stdio.h>
#include        <stdlib.h>
#include        <sys/time.h>
#include        <sys/types.h>

main(
int argc,
char *argv[]
```

```
)
{

time_t  epoch;

                  if(argc==2)
{
                        epoch=atol(argv[1]);
                         printf(asctime(localtime(&epoch)));
                  } else {
                         printf("Usage: %s epochdigits\n",argv[0]);
                  }
}
```

Stewart Robinson
Technical Services Manager (UK)

AIX Update on the Web

Code from individual articles of *AIX Update*, and complete issues in Acrobat PDF format, can be accessed on our Web site, at:

http://www.xephon.com/aix

You will be asked to enter a word from the printed issue.

Understanding the cut and paste commands

DIFFERENCES FROM WINDOWS

If you are familiar with the **cut** and **paste** commands in operating systems such as Windows, you may believe you are already familiar with the commands in AIX. In both Windows and AIX, **cut** and **paste** can manipulate sections of text from one file into another. However, there are significant differences. The understanding of those differences can help you to learn **cut** and **paste** for AIX. First, with Windows, the **cut** operation actually deletes the text from the input file. The AIX **cut** command is similar to the Windows **copy** command, which processes a copy of the data rather than removing it from the file.

With Windows, **cut** and **paste** operate on strings of text that 'flow', such as paragraphs. Sometimes Windows can operate on text that forms 'blocks', such as columns. AIX **cut** and **paste** work only on columns. However, the AIX version offers something that Windows **cut** and **paste** cannot: the ability to specify column separators such as tabs, semicolons, bars, or whatever character delineates the fields on the line, without regard to the length of the various fields. Therefore if Windows is your operating system of choice, and you have a file with variable line lengths and fixed delimiters, you may run **cut** and **paste** from AIX to get your desired output.

Further, Windows **cut** can operate on only one column at a time, but AIX **cut** can extract and copy several columns from a table while skipping columns not needed, all in a single operation.

For **paste**, again there are differences from Windows. The Windows **paste** command will process text that had been previously copied into a clipboard and insert such text into a file at a specific insert point. AIX paste treats its input as columns and concatenates columns vertically across the output. Another difference is that you can **cut** without pasting or **paste** without cutting in AIX.

The basic syntax of the **cut** and **paste** commands is shown below:

```
cut flags filespec
paste flags filespec
```

where:

- flags = an optional flag or flags used to enhance the operation.
- filespec = the file or files on which the operation is to be performed.

THE CUT COMMAND

The **cut** command sees the input file as a series of columns. The columns may be nicely formatted using tabs, or they may be of various lengths separated by a variety of characters, including the space character. As long as the input file has distinct fields across each row, the **cut** command can process them.

Flags for the cut command

The following flags assist the usefulness of the **cut** command:

- **-c** – tells the **cut** command which columns you would like to cut from the input file. If blanks or commas are used to separate the specified columns for the **c** flag, only characters falling on those specific boundaries will be cut. For example, if **cut -c 10 20 30 input_file** or **cut -c 10,20,30 input_file** were specified, only the characters in column 10, 20, and 30 from each line would be processed. However, if a hyphen is used, a range of characters is implied. For example, if **cut -c 10-30 input_file** were entered, all characters between columns 10 and 30 inclusive would be processed for each line.
- **-d** – when used with the **f** flag, the **d** flag tells **cut** which character it should consider as the column separator in the input file. For example, if you had lines in a file with the colon character separating the columns (which may contain variable

length strings and spaces), then the command **cut -f 2,4 -d : input_file** would display the second and fourth strings of data found between the colons.

- **-f** – tells the **cut** command which specific fields are to be cut from the file. Unless the **d** flag also appears on the command line specifying a delimiter character, the tab character is assumed. If blanks or commas are used to separate the specified fields for the **f** flag, those specific strings will be cut. For example, if **cut -f 3 5 7 input_file** or **cut -f 3,5,7 input_file** were specified, the third, fifth, and seventh columns from each line would be processed. If a hyphen is used, a range of fields is implied. For example, if **cut -f 3-7 input_file** were entered, all data from the third to the seventh fields would be processed for each line.
- **-s** – when used with the **f** flag, the **s** flag tells **cut** not to display lines that do not have the delimiter character in them. This ensures **cut** will operate only on the desired columnar lines in the file. Without the **s** flag, **cut** will just pass any lines not containing a delimiter.

A summary of the flags used with the **cut** command is:

- **-c** *chars* – specifies the desired character positions to cut.
- **-d** *delim* – specifies the delimiter used between fields.
- **-f** *fields* – specifies the desired fields to cut.
- **-s** – suppresses lines containing no delimiters.

Cut command examples

Example 1

Suppose you had a table called calendar.fil that looked like the following tab-delimited file:

```
SUN     MON     TUE     WED     THU     FRI     SAT

1       2       3       4       5       6       7
Mike    Jim     Bob     Ruby    Thomas  Cindy   Fred
```

```
8       9       1Ø      11      12      13      14
Fred    Thomas  Cindy   Bob     Ruby    Mike    Jim

15      16      17      18      19      2Ø      21
Thomas  Ruby    Jim     Mike    Cindy   Bob     Fred

22      23      24      25      26      27      28
Cindy   Mike    Fred    Thomas  Bob     Jim     Ruby

29      3Ø      31
Thomas  Fred    Mike
```

Let's say you want to copy the text from the second and the fifth columns (Monday and Thursday), and place the text into another file called MonThur.data. If you were to enter the command:

```
cut -f 2,5 calendar.fil > MonThur.data
```

the output file would look like this:

```
MON     THU

2       5
Jim     Thomas

9       12
Thomas  Ruby

16      19
Ruby    Cindy

23      26
Mike    Bob

3Ø
Fred
```

In this case, the **cut** command copied all data from the second and fifth tab delimited field in each line into the output file, MonThur.data. Notice there is no fifth column in the last lines.

Example 2

Now let's take a look at a case where you have an input file with columns of various widths and specific delimiters.

Suppose you had a text file called personnel.fil containing the

following data:

```
Employee Number:Employee Name:Start Date:Manager:Area:Last Rating

Ø136:Mike Jacobs:Ø1/Ø2/1996:Patrick C. McMann III:Sales:Above Average
Ø142:Jim Reilly:Ø3/Ø4/1997:Olivia Adams:R&D:Above Average
Ø151:Bob O'Hara:Ø5/Ø6/1998:Olivia Adams:R&D:Needs to show Improvement
Ø167:Ruby Maria Sanchez-Diego:Ø7/Ø8/1999:Patrick C. McMann
III:Sales:Outstanding
Ø178:Thomas M. Johannssen:Ø9/1Ø/2ØØØ:Olivia Adams:R&D:Above Average
Ø198:Cindy Og:11/12/2ØØ1:Olivia Adams:R&D:Outstanding
Ø2Ø1:Fred O. Williams:Ø1/Ø2/2ØØ2:Patrick C. McMann III:Sales:Average
```

Note that the entries contain many common punctuation marks such as colons, dashes, slashes, apostrophes, ampersands, and periods (full stops). However, it is the colon that delineates the various fields in this table, and this is the character you can tell the **cut** command to use to extract the data you want. Suppose you want to copy the *Employee Number*, *Manager*, and *Last Rating* into a report for a senior manager. By entering the command:

```
cut -f 1,4,6 -d : personnel.fil > report.fil
```

the results would appear as follows:

```
Employee Number:Manager:Last Rating

Ø136:Patrick C. McMann III:Above Average
Ø142:Olivia Adams:Above Average
Ø151:Olivia Adams:Needs to show Improvement
Ø167:Patrick C. McMann III:Outstanding
Ø178:Olivia Adams:Above Average
Ø198:Olivia Adams:Outstanding
Ø2Ø1:Patrick C. McMann III:Average
```

THE PASTE COMMAND

The paste command sees its input as a series of files, each one considered a separate column of data. Unlike the concatenation command **cat**, which merges files one after the other vertically, the **paste** command merges multiple files side by side horizontally.

Flags for the paste command

The following flags assist the usefulness of the **paste** command:

- **-d** – tells **paste** what character or characters to put between the columns in the output. In addition to standard keyboard characters, you may also specify the newline character \n, the tab \t, the backslash \\, or the empty string character \0. The default separation character is the tab.
- **-s** – merges lines from input files into one line. This tells **paste** to replace any newline characters with a tab (or whatever character is specified using the **-d** flag).

A summary of the flags used with the **paste** command is:

- **-d** *char* – specifies the delimiter character to put between each column in the output.
- **-s** – merges lines from input files into one line.

Paste command examples

The examples in this section use these three files as input, containing the following data:

```
months:

Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec


areas:

English Dept
Science Lab
Mathematics
```

```
History Dept
Home Economics
Auditorium
Music Dept
Gymnasium
Health Educ
Driver Educ
Cafeteria
Janitorial
```

```
staff:

Mrs Davies
Mr Green
Mr Williams
Miss Welks
Mrs Thomas
Mr Jacobsen
Miss Leery
Coach Peterson
Dr Mann
Mr Davenport
Ginnie Welch
Bud Lampett
```

Example 1

Let's say you want to merge the corresponding data from each file to show which staff member and area are responsible for the rotating monthly position of hall monitor. To do this, enter:

```
paste months areas staff > monitor.list
```

The results would be a file called monitor.list containing the following entries:

```
Jan     English Dept    Mrs Davies
Feb     Science Lab     Mr Green
Mar     Mathematics     Mr Williams
Apr     History Dept    Miss Welks
May     Home Economics  Mrs Thomas
Jun     Auditorium      Mr Jacobsen
Jul     Music Dept      Miss Leery
Aug     Gymnasium       Coach Peterson
Sep     Health Educ     Dr Mann
Oct     Driver Educ     Mr Davenport
Nov     Cafeteria       Ginnie Welch
Dec     Janitorial      Bud Lampett
```

Example 2

Now suppose you want a file containing areas and staff members with each column separated by an asterisk. Enter:

```
paste  -d '*' areas staff > dept.list
```

This would give you a file called dept.list containing:

```
English Dept*Mrs Davies
Science Lab*Mr Green
Mathematics*Mr Williams
History Dept*Miss Welks
Home Economics*Mrs Thomas
Auditorium*Mr Jacobsen
Music Dept*Miss Leery
Gymnasium*Coach Peterson
Health Educ*Dr Mann
Driver Educ*Mr Davenport
Cafeteria*Ginnie Welch
Janitorial*Bud Lampett
```

Example 3

Suppose you would like to merge every three lines of the staff list into one line. Enter:

```
paste -s -d '\t\t\n' staff > staff.list
```

This would give you a file called staff.list containing:

```
Mrs Davies      Mr Green        Mr Williams
Miss Welks      Mrs Thomas      Mr Jacobsen
Miss Leery      Coach Peterson  Dr Mann
Mr Davenport    Ginnie Welch    Bud Lampett
```

In this case, **-s** tells **paste** to consider all output as one line. Then the **-d** flag tells **paste** to take that one line and to insert a tab after each first and second entry, and to put a newline after each third.

Once you gain experience in using the **cut** and **paste** commands, you will find it useful to pipe the output of the **cut** command into the **paste** command for specialized formatting.

David Chakmakian
Programmer (USA)

Core Technology has announced an update to FileSWEEP/Rapid, its file transfer software suite. The product is a cross-platform, cross-operating system one, running on AIX, SUN-Solaris, HP-UX, SCO, and Windows.

FileSWEEP/Rapid has a client and a server facility on each platform, which facilitates pushing or pulling of files in either direction. The new release has enhanced security, including data and password encryption, improved error report handling, and various minor enhancements on all platforms. Security has been enhanced to include 128-bit encryption and decryption for file transfers.

For further information contact:
Core Technology Corporation, 7435 Westshire Drive, Lansing, MI. 48917, USA.
Tel; (517) 627 1521.
URL: http://www.ctc-core.com/products/rapid/overview.htm.

* * *

CommVault Systems has Version 5 of it Galaxy back-up and recovery software. Enhancements in this version include expanded platform/application support, added data protection support for DB2 7.2 (Solaris 8, AIX 5.1), Netware 6.5, new media agent support for Linux RH AS 2.1, Solaris 9, AIX 5.2, and Netware 6.5. There's also expanded NDMP support, a Unix command-line interface, and numerous user and management-level reports that help track resource and data availability needs.

For further information contact:
CommVault Systems, 2 Crescent Place, PO Box 900, Oceanport, NJ 07757, USA.
Tel: (732) 870 4104.
URL: http://www.commvault.com/products_faq.asp?pid=1.

* * *

FileNet has announced its FileNet Image Services Resource Adapter (ISRA), which provides connectivity from applications built on Sun Microsystems' Java 2 Platform Enterprise Edition (J2EE), a distributed computing development framework, to content managed within FileNet Image Services repositories.

ISRA supports BEA WebLogic 8.1 and 7.0 SP2 and IBM WebSphere 4.0.4AE application servers and AIX 4.3.3 and above, HP/UX 11.0 and above, Windows 2000 and Sun Solaris 8.0 and above.

ISRA, provides a Java-based solution to address connectivity between application servers and legacy enterprise information systems. The solution is designed specifically for industries that have large-scale integrated content management systems in place.

For further information contact:
FileNet, 3565 Harbor Blvd, Costa Mesa, CA 92626-1420, USA.
Tel: (714) 327 3400.
URL: http://www.filenet.com.

* * *