45

AIX update

July 1999

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 550955
From USA: 01144 1635 33823
E-mail: HarryLewis@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about AIX, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you actively be helping the free exchange of information, which benefits all AIX users, but you will also gain professional recognition for your expertise and that of your colleagues, as well as being paid a publication fee – Xephon pays at the rate of £170 ($250) per 1000 words for original material published in AIX Update.

To find out more about contributing an article, see *Notes for contributors* on Xephon's Web site, where you can download *Notes for contributors* in either text form or as an Adobe Acrobat file.

Editor

Harold Lewis

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.

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 1995 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

AIX Update on-line

Code from *AIX Update* is available from Xephon's Web page at *www.xephon.com/aixupdate* (you'll need the user-id shown on your address label to access it).

Printed in England.

Assorted shell programming techniques

INTRODUCTION

The Unix shells (Korn, Bourne, or C shell) provide a powerful programming environment. The Korn shell, with its support for arrays, is particularly powerful. While shell programming is not usually considered to be mainstream programming, I find that good programming practice can be extended to it. This article describes what is considered good programming practice in traditional programming and extends it to shell programming.

VARIABLES AND DATATYPES

Defining variables

1 Double quotes are required if there are spaces between words

2 *VAR1* below is defined as 'null'

3 *VAR2* below is assigned whatever is returned by command 'Function'.

Example:

```
NUM=2
NAME=ARIF
FULL_NAME="Arif Zaman"
VAR1=
VAR2=`Function`
```

Data types

All variables are treated as 'char'.

Variable assignment

Variables can be assigned directly or using the command 'echo' enclosed in back quotes ('`').

Example:

```
NUM1=10
NUM2=
NUM2=$NUM1   or  NUM2=`echo  $NUM1`
```

Using and assigning default values variables

The construct *${variable:-word}* is interpreted as ‘if *variable* is undefined, use *word*’. This assigns *word* to *variable* if *variable* is undefined or null, otherwise leaving it unchanged.

Example:

```
FULLNAME="${FIRST_NAME:-Mr} ${LAST_NAME:-XXXX}"
```

If *FIRST_NAME* and *LAST_NAME* are not defined, *FULLNAME* has the value ‘Mr XXXX’.

Providing a message for a missing variable value

This can be achieved using the construct ‘${variable:?message}’:

```
${NAME:?"Not defined"}
```

If *NAME* is not defined, the following is output:

```
filename: NAME: Not defined
```

Finding string length

You can find a string length using the construct below.

```
STRING="ABCDEGFH"
STRLEN=`echo ${STRING}\c | wc -c`
```

Testing for the NULL string

The segment of code below shows you how to test for ‘NULL’ strings.

```
STRING1=

if   [-z "${STRING1}"]
  then
    echo "STRING1 is NULL"
fi

if   ["${STRING1}" = ""]
  then
```

```
        echo "Variable is not set"
    fi
```

Extracting substrings from variables

1 *cut* is the command most commonly used for this purpose.

2 The first example below uses *cut* with the ‘column option’.

3 The second example uses *cut* with the ‘delimiter option’.

Example:

```
NAME="Arif Zaman"
FNAME=`echo $NAME | cut -c 1-4`
LNAME=`echo $NAME | cut -d' ' -f2`
```

Appending strings

1 One or more variables or words can be joined together using double quotes.

2 Self-referencing is allowed as long as the reference is to an existing variable, as shown in the second part of the code fragment below.

Example:

```
FIRST_NAME = "Arif"
LAST_NAME  = "Zaman"
FULL_NAME  = "${FIRST_NAME} ${LAST_NAME}"

FULL_NAME  = "Arif"
FULL_NAME  = "$FULL_NAME Zaman"
```

Korn shell arrays

The Korn shell supports only one dimensional arrays, such as:

```
NAME[ ]
ADDRESS[ ]
```

You define array variables as follows:

```
NAME[0]="ARIF ZAMAN"
NAME[1]="HARVEY JONES"
```

You can manipulate array variables using literal indexes:

```
echo ${NAME[0]}
echo ${NAME[2]}
```

(the first line outputs ‘ARIF ZAMAN’, while the second outputs ‘HARVEY JONES’), or using a variable index (the code below outputs all names from ‘NAME’):

```
INDEX=0
while ["${NAME[$INDEX]}" != ""]
  do
    echo ${NAME[$INDEX]}
      INDEX=`expr $INDEX + 1`
  done
```

CONTROL STRUCTURES

‘if ... then ... else’

1 The following operators can be used:

- *Greater than*
  Either ‘>’ or ‘-gt’
- *Less than*
  Either ‘<’ or ‘-lt’
- *Equal to*
  Either ‘=’ or ‘-eq’.

2 The keyword ‘then’ must be on a separate line unless used along with a semicolon(‘;’), as shown below.

```
NAME="ARIF"

if ["${NAME}" = "ARIF"]
  then
    echo "Name is $NAME"
  else
    echo "Name is not $NAME"
fi

NAME="Arif Zaman"

if ["${NAME}" -eq "Arif Zaman"] ; then echo "Name is $NAME"
  else
    echo "Name is not $NAME"
fi
```

Testing for the existence of files and directories

There are many options available for carrying out this check, including the ones shown below.

Testing for the existence of files:

```
FILE=file

if [-f ${FILE}]
  then
    echo "$FILE exists"
  else
    echo "$FILE does not exist"
fi
```

Testing for the existence of directories:

```
DIR=dir

if [ -d  $DIR ]
  then
    echo "$DIR exists"
  else
    echo "$DIR does not exist"
fi
```

Error handling

In the code fragment below, if the word ‘ERROR’ is found in the file */users/afz/err/xx.err*, the **grep** command returns ‘true’. As we’re not interested in any form of output, we redirect standard error and standard output to the null device (‘/dev/null 2>&1’).

```
if
    grep "ERROR" /users/afz/err/xx.err > /dev/null 2<&1
  then
    echo "Error in the file"
fi
```

while [condition]

Below is an example of using a ‘while’ construct.

```
COUNTER=0
while [ $COUNTER -lt 10 ]
  do
    echo $COUNTER
    COUNTER=`expr $COUNTER + 1`
  done
```

Another form of ‘while’ is ‘while true’:

```
COUNTER=0
while true
  do
    echo $COUNTER
    COUNTER=`expr $COUNTER + 1`
    if [ $COUNTER -gt 10 ]
      then break
    fi
  done
```

Note that the condition ‘true’ remains true forever. This means that ‘break’ must be used to break out of the loop.

‘for’ loops

‘for’ loops behave much as they do in other programming languages, however:

1 ‘continue’ can be used to abandon the current loop and start processing the next value of the control variable.

2 The loop below is automatically terminated when the last value of the variable ‘NAMES’ is read and processed.

```
NAMES="ARIF DEV MADHU SATISH ANDY PANDY"

for NAME in $NAMES
  do
    if [ "${NAME}" = "ARIF" ]
      then
        continue
    fi
    echo "Name is $NAME"
  done
```

FUNCTIONS

In shell programming it is possible to define functions using a method similar to that used in C programs. One exception is that the function must be defined in the shell script before it is called.

```
display ( )    # define a function that accepts one argument
{
  echo $1
}
```

```
main ( )                 # define a main function
{
    display "Welcome"    # call display
    VAR1=10
}

main                     # call the main function
```

When a function is called, the argument count is held in the variable ‘#’:

```
function ( )
{
    if [ $# != 2 ]
    then
        echo "Wrong number of arguments."
    fi
    ...
}
```

Assigning arguments to variables

Use the dollar symbol (‘$’) when assigning arguments to variables:

```
function1(  )
{
    VAR1=$1       # assigning first argument
    VAR2=$2       # assigning second argument
}
```

Scope and visibility of function variables

Consider the two programs below.

First program (**p1.sh**):

```
VAR1=10
VAR2=12
```

Second program (**p2.sh**):

```
FNAME=Arif
LNAME=Zaman
echo  $VAR1
```

The script below illustrates the scope of the programs’ function variables.

```
p2.sh          # p2.sh is executed in a child shell. This
               # means it won't see variable VAR1 as it's
```

```
                   # defined in the parent shell and has not
                   # been exported to child shells

echo $FNAME        # This statement won't echo anything as p1.sh
                   # is executed in the parent shell and FNAME
                   # is defined in the child shell
```

Now consider the alternative example below.

First program (**p1.sh**):

```
VAR1=10; export VAR1
VAR2=12; export VAR2
```

Second program (**p2.sh**):

```
FNAME=Arif
LNAME=Zaman
echo $VAR1
```

The effect of running the same shell script on these two programs is somewhat different:

```
p2.sh              # Even though p2.sh is executed in child, it
                   # can see the variable VAR1, which is exported.

echo $FNAME        # p1.sh is still unable to see variable FNAME,
                   # which is not defined in the current shell.
```

However, consider the script below, which uses the same versions of **p1.sh** and **p2.sh** as the previous example.

```
. p2.sh            # p2.sh is executed in the current shell, which
                   # means that it is able to see the variable
                   # VAR1, even if it is not exported. Note that
                   # the dot ('.') makes p2.sh execute in the
                   # current shell, and that there must be a
                   # space after dot.

echo $FNAME        # p1.sh can now see the variable FNAME as it
                   # is defined in the current shell.
```

Returning values from a function call

In shell programming, functions can return values using the keyword *return*, though, in contrast with the C language, ‘TRUE’ is defined as ‘0’ and ‘FALSE’ as ‘1’.

```
# define TRUE and FALSE
TRUE=0
```

```
FALSE=1

# define a function
StringSearch ( )
{
    STRING="$1"
    FILE="$2"

    if grep "${STRING}" ${FILE} > /dev/null 2>&1
        then
            # string exists in the file
            return $TRUE
        else
            return $FALSE
    fi
}

# define main function
main ( )
{
    WORD="ORA-1403"
    ERROR_FILE="/tmp/oracle.err"

    # call function
    if StringSearch ${STRING} ${ERROR_FILE}
        then
            echo "Error found"
        else
            echo "No error found"
    fi
}

# invoke main
main
```

Below is an illustration of a method of returning a value.

```
# define a function
AddNumbers ( )
{
    NUM1="$1"
    NUM2="$2"

    TOTAL=`expr $NUM1 + $NUM2`
    echo  ${TOTAL}
}

# define main function
main ( )
{
    NUMBER1=10
    NUMBER2=100
    SUM=
    # call function
    SUM=`AddNumbers $NUMBER1 $NUMBER2`
}

# invoke main
main
```

Include and macro files

If a program has many variable definitions, then it makes sense to have a separate definition file that can be included in the shell in which the main program runs. Similarly, if a program has many macros, it also makes sense to have a separate macro definition file. An example is shown below.

Variable definition file (**x.def**):

```
PROG="x.def"
DIR="/users/afz/sh"
VAR1=10
VAR2=12
```

Macro file (**x.mac**):

```
display ( )
    {
        echo $1
    }
```

Main program (**main.sh**):

```
# execute definition file in the current shell
. var.def

# execute macro file in the current shell
. mac.def

# call macro display to display the message
display " Welcome to the utility program"
```

ESCAPE SEQUENCES

When programming shell scripts, embedded escape sequences can be useful, especially if the script communicates with a terminal. In this section, a library of these escape sequences is included for you to try out and embed in your own scripts.

To enter an escape sequence, use the following procedure:

- Press *Ctrl-V*
- Press *Esc* key (to display the escape character)
- Enter the remaining characters.

To capture an escape sequence for function key, use the method outlined below.

- Launch **vi**
- Enter 'insert mode'
- Press *Ctrl-V*
- Press the required function key.

In a future article I'll present a function library that should take much of the hard work out of coding output for VT420 (and compliant) terminals.

USING COMMON UNIX UTILITIES

Below are a few examples of the use of Unix utilities in shell scripts. These are intended to show how common utilities are used and not as a tutorial on the Unix toolkit.

The grep command

Below is an example of using **grep** to search a string.

```
ERROR_FILE=/tmp/file.arr

if grep ORA- ${ERROR_FILE}
then
    # if string ORA- is found in the file ...
    echo "Error found"
fi
```

grep uses regular expressions. While this subject has been discussed before in *AIX Update* (see Issues 30 and 31), some of the commands most commonly used are summarized below.

- *. (period)*
  Match any single character.
- ** (asterisk)*
  Match zero or more repetitions of the preceding characters.
- *[ ] (square brackets)*
  Match any of the characters enclosed in the brackets.
- ^ *(caret)*
  The beginning of the line.
- *$ (dollar)*
  The end of the line.

Example:

```
grep  '^\.D[SE]$'  file1
```

Note:

1 The symbols ^ and *$* indicate that the search string must span the entire line.

2 The backslash ('\') 'escapes' the dot, preventing it from being interpreted as part of a regular expression, and allowing the formation of such search patterns as '.DS' and '.DE'.

The cut command

The use of the **cut** utility is best demonstrated by example. Consider an instance where a string comprises a number of items delimited by colons (':'):

```
VAR="AAAA:BBBB:CCCC:DDDD"
```

The **cut** command can be used to parse the string:

```
echo  "${VAR}"  |  cut  -d':' -f1
```

returns 'AAAA', and:

```
echo  "${VAR}"  |  cut  -d':' -f2
```

returns 'BBBB', etc.

Cut can also be used to extract a substring from a string of characters:

```
     VAR="ABCDEFGHIJKLM"

     echo  "${VAR}"  |  cut  -c 1-5
```

Returns ‘ABCDE’.

The tr command

This command ‘translates’ instances of a character in a string to another character. For example:

```
     # define variable
     FULL_NAME=
     echo  "${FULL_NAME}"  |  tr ":"  " "
```

This short script translates each of the strings on the left below to the ones on the right.

```
     Arif:Zaman                    Arif Zaman
     John:Barnes                   John Barnes
     Addul:Hakim                   Addul Hakim
```

The sed command

One use of **sed** is to implement global changes for every occurrence of a string in a file. For example:

```
FILE1=/tmp/file1.dat     # contains instances of $OLD_STRING
FILE2=/tmp/file2.dat     # where to put processed output
OLD_STRING="PATH"
NEW="PATH_NAME"

# This sends the output to the terminal
sed  s/${OLD_STRING}/${NEW_STRING}/g  ${FILE1}

# This sends the output to $FILE2
sed  s/${OLD_STRING}/${NEW_STRING}/g  ${FILE1}  ${FILE2}
```

A detailed discussion of **sed** can be found in *AIX Update* Issue 18.

The awk command

A use of **awk** is to process record items that are delimited by white space. For example:

```
     FULL_NAME="ARIF  ZAMAN"
     FULL_NAME="ANDREW     JONES"
```

```
    # assign first name
    FIRST_NAME=`echo "${FULL_NAME}" | awk {'print  $1'}`

    # assign last name
    LAST_NAME =`echo "${FULL_NAME}" | awk {'print  $2'}`
```

OPERATORS

There are a number of operators that can be used in shell scripts.

The dot operator (‘.’)

This operator was encountered briefly earlier in this article. It is used to indicate the program being invoked is to be executed in the current shell, making its variables accessible to the executing program.

The redirection operator (‘>’)

This operator is fairly well known – it redirects the output of a program.

The concatenation operator (‘>>’)

This operator concatenates the output of a program to the specified device.

The ‘<< !’ operator

This operator is used to supply the string between the ‘<< !’ and ‘!’ symbols as input to a command being executed. This can be used for the execution of SQL statements and other Unix commands, such as **ed** and **sed**.

The best way to clarify the use of this operator is by example, the one below being for an SQL command.

```
    DEPT_EXISTS=
    DEPT_EXISTS=`sqlplus  -s  /  << !
           set     heading  off
           set     feedback off
           SELECT  'Y'
           FROM    dept
           WHERE   deptno = 10;
    !`
```

Note that the closing ‘!’ must be in the first column.

The ‘tee’ operator

This operator is used to send output both to terminals and files. For example:

```
cat names.dat | tee -a file.dat
```

The contents of the file *names.dat* are sent to the terminal and appended to the file *file.dat.*

Back quotes (‘`’)

Back quotes are used to execute a command ‘in silence’. For example:

```
NAME=`echo $FIRST_NAME`
```

The variable *$FIRST_NAME* is echoed internally and the value is assigned to variable *$NAME*.

The colon operator (‘:’)

The colon operator is used as a null statement in an ‘if-then-else’ construct. For example:

```
if [ "$NAME" = "ARIF" ]
    then
        :         # do nothing
    else
        echo "Name is $NAME"
fi
```

LIBRARY FUNCTIONS

ulib, listed below, is both an example of how to write a shell library and a practical library in its own right. Feel free to use it in your programs.

Note the use of the continuation character, ‘➤’, in the code below to indicate that one line of code maps to several lines of print.

ULIB

```
#! /bin/ksh
#####################################################################
#                                                                   #
# Author      : Arif  Zaman                                         #
# Name        : ulib (utility library)                              #
# Description : Defines all global variables and functions          #
#                                                                   #
# Notes   1   Displayed error messages have the following parts:   #
#                                                                   #
#             <Calling Script>:<Calling Function>:S-ulib:F-         #
#                                       <Called Library Function>   #
#             <Calling Script>:S-ulib:F-<Called Library Function>   #
#                                                                   #
#         2   When calling library functions, the following global  #
#             variables must be set:                                #
#                                                                   #
#             - CALLING_SCRIPT=S-xxx     where S is the script      #
#             - CALLING_FUNCTION=F-xxx   where F is the function    #
#                                                                   #
#         3   Every library function returns either TRUE or FALSE   #
#             and can also assign a return value (converted to      #
#             uppercase, if appropriate) to RETURNED_VALUE.         #
#                                                                   #
#         4   To incorporate this library in another script, add    #
#             the following line as the first executable command    #
#             in the script:                                        #
#                                                                   #
#                . ulib                                             #
#                                                                   #
#         5   The Library contains following functions:             #
#             - IsDigit                                             #
#             - IsAlpha                                             #
#             - Strlen                                              #
#             - AgeFile                                             #
#             - PrintSpoolFile                                      #
#             - GetYNConfirmation                                   #
#             - MoveCursor                                          #
#             - DisplayMessage                                      #
#             - StripCR                                             #
#                                                                   #
#####################################################################

#####################################################################
#                      Define global variables                      #
#####################################################################
DefineGlobalVariables ( )
{
PRINTER=LASER; export PRINTER

TRUE=0 ; export TRUE
FALSE=1; export FALSE

SUCCESS=0; export SUCCESS
FAILURE=1; export FAILURE
```

```
DATE=`date +%d/%m/%y`; export DATE
TIME=`date +%H:%M:%S`; export TIME
EXIT_CODE=${FAILURE} ; export EXIT_CODE

FUNCTION_NAME=
# Export FUNCTION_NAME

EM=":ERROR: "
IM=":INFO: "
ESC="\0033["

OPTION=       #  Selected menu option

MENU_NAME=
MSG_TYPE=
MESSAGE=

SLEEP_DURATION=5

ESC="\0033[["
RVON=" [7m"
RVOFF=" [m"
BON=5m
BOFF=25m

# Used in the display of error messages
CALLING_SCRIPT=        ; export CALLING_SCRIPT
CALLED_SCRIPT="S-ulib" ; export CALLED_SCRIPT
CALLING_FUNCTION=      ; export CALLING_FUNCTION
CALLED_FUNCTION=       ; export CALLED_FUNCTION

# Values are returned through this value, except for TRUE and FALSE
RETURNED_VALUE=        ; export RETURNED_VALUE
}

#####################################################################
#                      Defines global messages                      #
#####################################################################
DefineGlobalMessages ()
{
FILE_NAME_MISSING="Must provide a file name to be printed${RVOFF}"
STRING_MISSING="Must provide a string as a parameter${RVOFF}"
EMPTY_STRING="Parameter string is empty${RVOFF}"
AGE_USAGE="Usage:age \<file name\> \<file generation\>${RVOFF}"
INVALID_GENERATION="\${KEEP_GENERATION}, is an invalid generation
➤  parameter${RVOFF}"
NO_FILE_TO_AGE="cannot age a non existent file, \${FILE}${RVOFF}"
GENERATION_MISSING="\${GENERATION_TO_MOVE} generation of file\
➤  (\${FILE}.\${GENERATION_TO_MOVE}\) is missing${RVOFF}"
}
#####################################################################
#                                                                   #
# GetYNConfirmation                                                 #
#                                                                   #
# This function gets a Y/N confirmation to a message.               #
#                                                                   #
# Notes   1   The function returns FALSE if:                        #
#             - No message is passed                                #
#             - The message string is empty.                        #
#                                                                   #
#         2   The function returns TRUE if:                         #
#             - The user has entered either 'Y' or 'N'.             #
#                                                                   #
#         3   The following return values are passed back:          #
#             - 'Y' or 'N'.                                         #
#                                                                   #
#####################################################################
GetYNConfirmation ()
{
CALLED_FUNCTION="F-GetYNConfirmation"

# Has a string been passed?
if [  $# -eq 0 ]
then
    # No parameter was passed
    DisplayMessage E "$STRING_MISSING"
    return ${FALSE}
fi

MESSAGE=$1

if [ -z ${MESSAGE} ]
then
    # String is empty
    DisplayMessage E "$EMPTY_STRING"
    return ${FALSE}
fi

# Get confirmation
while true
do
    clear
    echo "$MESSAGE\c"
    read REPLY
    case $REPLY in
        y|Y ) RETURNED_VALUE="Y";
              return ${TRUE};;

        n|Y ) RETURNED_VALUE="N";
              return ${TRUE};;
```

```
           *) ;;;
    esac
done
}

######################################################################
#                                                                    #
# IsDigit                                                            #
#                                                                    #
# This function checks whether a string contains only digits.        #
#                                                                    #
# Input   : A string                                                 #
#                                                                    #
# Returns : TRUE or FALSE                                            #
#                                                                    #
# Notes   1   The function returns FALSE if:                         #
#             - The string contains one or more letters              #
#             - The string is empty                                  #
#             - No string passed.                                    #
#                                                                    #
#         2   The function returns TRUE if:                          #
#             - The string contains only digits.                     #
#                                                                    #
######################################################################
IsDigit  ()
{
CALLED_FUNCTION="F-IsDigit"

# Has a string been passed?
if [  $# -eq 0 ]
then
    # No parameter was passed
    DisplayMessage E "$STRING_MISSING"
    return ${FALSE}
fi

STRING=$1
if [ -z ${STRING} ]
then
    # String is empty
    DisplayMessage E "$EMPTY_STRING"
    return ${FALSE}
fi

# Get the string length
if Strlen ${STRING}
then
    LEN=$RETURNED_VALUE
else
    return ${FALSE}
fi
```

```
STARTPOS=1
ENDPOS=1

while true
do
    if [ $STARTPOS -gt $LEN ]
    then
        break
    fi
    DIGIT=`echo $STRING | cut -c $STARTPOS-$ENDPOS`
    if [ "$DIGIT" != "0" -a "$DIGIT" != "1" -a "$DIGIT" != "2" -a \
         "$DIGIT" != "3" -a "$DIGIT" != "4" -a "$DIGIT" != "5" -a \
         "$DIGIT" != "6" -a "$DIGIT" != "7" -a "$DIGIT" != "8" -a \
         "$DIGIT" != "9" ]
    then
        return $FALSE
    fi
    STARTPOS=`expr $STARTPOS + 1`
    ENDPOS=${STARTPOS}
done
return $TRUE
}

######################################################################
#                                                                    #
# IsAlpha                                                            #
#                                                                    #
# This function checks whether a string contains only alphabetic     #
# characters.                                                        #
#                                                                    #
# Input   : A string                                                 #
#                                                                    #
# Returns : TRUE or FALSE                                            #
#                                                                    #
# Notes   1   The function returns FALSE if:                         #
#             - The string contains any digits                       #
#             - The string is empty                                  #
#             - No string passed.                                    #
#                                                                    #
#         2   The function returns TRUE if:                          #
#             - The string contains only alphabetic characters.      #
#                                                                    #
######################################################################
IsAlpha  ()
{
CALLED_FUNCTION="F-IsAlpha"

# Has a string been passed?
if [ $# -eq 0 ]
then
```

```
    # No parameter has been passed
    DisplayMessage E "$STRING_MISSING"
    return ${FALSE}
fi

STRING=$1
if [ -z ${STRING} ]
then
    # String is empty
    DisplayMessage E "$EMPTY_STRING"
    return ${FALSE}
fi

# Get the string length
if  Strlen ${STRING}
then
    LEN=$RETURNED_VALUE
else
    return ${FALSE}
fi

STARTPOS=1
ENDPOS=1

while true
do
    if [ $STARTPOS -gt $LEN ]
    then
        break
    fi
    DIGIT=`echo $STRING | cut -c $STARTPOS-$ENDPOS`
    if [ "$DIGIT" = "0" -o "$DIGIT" = "1" -o "$DIGIT" = "2" -o \
         "$DIGIT" = "3" -o "$DIGIT" = "4" -o "$DIGIT" = "5" -o \
         "$DIGIT" = "6" -o "$DIGIT" = "7" -o "$DIGIT" = "8" -o \
         "$DIGIT" = "9" ]
    then
        return $FALSE
    fi
    STARTPOS=`expr $STARTPOS + 1`
    ENDPOS=${STARTPOS}
done
return $TRUE
}

######################################################################
#                                                                    #
# MoveCursor                                                         #
#                                                                    #
# This function moves the cursor to a specified position.            #
#                                                                    #
#  Input  : y coordinate                                            #
#           x coordinate                                            #
#                                                                   #
#####################################################################
MoveCursor ()
{
CALLED_FUNCTION="F-MoveCursor"
YCOR=$1
XCOR=$2
print  -n  " [${YCOR};${XCOR}H"
}

#####################################################################
#                                                                   #
# DisplayMessage                                                    #
#                                                                   #
# This function displays a message.                                 #
#                                                                   #
# Input   : Message type                                            #
#           Message                                                 #
#                                                                   #
#####################################################################
DisplayMessage()
{
CALLED_FUNCTION="F-DisplayMessage"

MSG_TYPE=$1
MESSAGE=$2

# Prepare function name to be displayed with error message
if   [ "${CALLING_SCRIPT}" =  "" -a "${CALLING_FUNCTION}" = "" ]
then
    FUNCTION_NAME="${CALLED_SCRIPT}:${CALLED_FUNCTION}"
elif [ "${CALLING_SCRIPT}" != "" -a "${CALLING_FUNCTION}" = "" ]
then
    FUNCTION_NAME="${CALLING_SCRIPT}:${CALLED_SCRIPT}:
    ➤  ${CALLED_FUNCTION}"
else
    FUNCTION_NAME="${CALLING_SCRIPT}:${CALLING_FUNCTION}:
    ➤  ${CALLED_SCRIPT}:${CALLED_FUNCTION}"
fi

EVALUATED_MESSAGE="`eval echo ${MESSAGE}`"

if [ "${MSG_TYPE}" = "E" ]
then
    # clear
    MoveCursor 23 1
    EM="${FUNCTION_NAME}${EM}"
    echo "${RVON}${EM}${EVALUATED_MESSAGE}${RVOFF}\c"
    sleep  ${SLEEP_DURATION}
    echo "\n"
else
    clear
    MoveCursor 23 1
    IM="${FUNCTION_NAME}${IM}"
    echo "${RVON}${IM}${EVALUATED_MESSAGE}${RVOFF}\c"
    sleep  ${SLEEP_DURATION}
    echo "\n"
fi

# Reset variables
CALLING_SCRIPT=
CALLED_SCRIPT=
CALLING_FUNCTION=
CALLED_FUNCTION=
}

#####################################################################
#                                                                   #
# PrintSpoolFile                                                    #
#                                                                   #
# This function prints the named file.                              #
#                                                                   #
# Input   : File name to be printed                                 #
#                                                                   #
# Note      This function suppresses the banner page.               #
#                                                                   #
#####################################################################
PrintSpoolFile ( )
{
CALLED_FUNCTION="F-PrintSpoolFile"

# Was a string passed?
if [ $# -eq 0 ]
then
    # no parameter was passed
    DisplayMessage E "$FILE_NAME_MISSING"
    return ${FALSE}
fi

FILE_TO_BE_PRINTED=$1
# Check parameter
if [ -z "${FILE_TO_BE_PRINTED}" ]
then
    DisplayMessage E "$EMPTY_STRING"
    return ${FALSE}
fi

while true
```

```
do
    clear
    echo "Do you wish to print the output(Y/N):\c"
    read REPLY
    case $REPLY  in
        Y|y) lp -o nb -d$PRINTER ${FILE_TO_BE_PRINTED};
             break;;

        N|n) break;;

          *) :;;
    esac
done
}

#####################################################################
#                                                                   #
# AgeFile                                                           #
#                                                                   #
# This function "ages" the specified file and keeps the specified   #
# number of generations of it.                                      #
#                                                                   #
# Input   :  File name                                              #
#            Number of generations to keep                          #
#                                                                   #
# Notes   1   The file name is in the following format:             #
#                file.extension                                     #
#                                                                   #
#         2   The "aged" file name is in the following format:      #
#                file.extension.generation                          #
#                                                                   #
#         3   The function returns an error if it doesn't find all  #
#             files between the first generation and the one        #
#             specified.                                            #
#                                                                   #
#####################################################################
AgeFile( )
{
CALLED_SCRIPT="F-AgeFile”

# Set globals
CALLED_SCRIPT="S-ulib"
CALLED_FUNCTION="F-AgeFile"

# Check the parameters
if [ $# -eq 2 ]
then
    :
else
    # Display error message
    DisplayMessage E "${AGE_USAGE}"
    return ${FALSE}
fi

FILE=$1
KEEP_GENERATION=$2
GENERATION_TO_REMOVE=
GENERATION_TO_MOVE=
LOOP=${KEEP_GENERATION}

# Check that the file exists
if [ ! -f $FILE ]
then
    # Display error message
    DisplayMessage E "${NO_FILE_TO_AGE}"
    return ${FALSE}
fi

# Check the generation parameter
if IsDigit ${KEEP_GENERATION}
then
    if [ ! ${KEEP_GENERATION} -gt  0 ]
    then
        # Display error message
        DisplayMessage E "${INVALID_GENERATION}"
        return ${FALSE}
    fi
else
      # Display error message
      DisplayMessage E "${INVALID_GENERATION}"
      return ${FALSE}
fi

# Remove the last generation to make room for new file
GENERATION_TO_REMOVE=${KEEP_GENERATION}
(rm -f ${FILE}.${GENERATION_TO_REMOVE}) 2> /dev/null

# Special treatment if only one generation to be kept
if [ "${KEEP_GENERATION}" -eq 1 ]
then
    mv -f ${FILE} ${FILE}.${KEEP_GENERATION}
else
    while [ ${LOOP} -gt 1 ]
    do
        # Age all generations of files
        GENERATION_TO_MOVE=`expr ${LOOP} - 1`
        if [ ! -f ${FILE}.${GENERATION_TO_MOVE} ]
        then
            # Display error message
            DisplayMessage E "${GENERATION_MISSING}"
            return ${FALSE}
        fi
        mv -f ${FILE}.${GENERATION_TO_MOVE} ${FILE}.${LOOP}
        LOOP=`expr ${LOOP} - 1`
    done

    # Age the new file to generation 1
    mv -f ${FILE} ${FILE}.1
fi

# Now deal with generations above $KEEP_GENERATION
GENERATION_ABOVE=`expr ${KEEP_GENERATION} + 1`
while [ -f ${FILE}.${GENERATION_ABOVE} ]
do
    rm -rf  ${FILE}.${GENERATION_ABOVE}
    GENERATION_ABOVE=`expr ${GENERATION_ABOVE} + 1`
done
}

#####################################################################
#                                                                   #
# Strlen                                                            #
#                                                                   #
# This function returns the length of a variable.                  #
#                                                                   #
# Input   : Name of a variable                                      #
#                                                                   #
# Notes   1   If the parameter is missing, the function returns:    #
#                ${FALSE}                                           #
#                                                                   #
#         2   The syntax for calling this function is:              #
#                                                                   #
#                if Strlen ${STRING}                                #
#                then                                               #
#                    LEN=${RETURNED_VALUE}                          #
#                fi                                                 #
#                                                                   #
#####################################################################
Strlen ( )
{
CALLED_FUNCTION="F-Strlen"

# Was a string passed?
if [ $# -eq 0 ]
then
    # No parameter was passed
    DisplayMessage E "$STRING_MISSING"
    return ${FALSE}
fi
```

```
STRING=$1
# Check parameter
if [ -z "${STRING}" ]
then
    # DisplayMessage E "$STRING_MISSING"
    ➤  "$SCRIPT:$FUNCTION:ulib:Strlen"
    return ${FALSE}
fi

STRLEN=`echo "$STRING\c" | wc -c`

RETURNED_VALUE="${STRLEN}"
return ${TRUE}
}

#####################################################################
#                                                                   #
# Array                                                             #
#                                                                   #
# This function demonstrates the use of arrays                      #
#                                                                   #
# Input   : Array elements (NAME1, NAME2, NAME3, etc)               #
#                                                                   #
#####################################################################
Array ( )
{
CALLED_FUNCTION="F-Array"

ARRAY_LEN=$#

# Store the arguments
ELEMENT=0
while true
do
    ARRAY[ $ELEMENT ]="`eval echo \\$$ELEMENT`"

    ELEMENT=`expr $ELEMENT + 1`
    if [ $ELEMENT -gt $ARRAY_LEN ]
    then
        break
    fi
done

# Display the arguments
ELEMENT=0
while true
do
    echo ${ARRAY[ $ELEMENT ]}

    ELEMENT=`expr $ELEMENT + 1`
    if [ $ELEMENT -gt $ARRAY_LEN ]
    then
        break
    fi
done
}

# Invoke functions
DefineGlobalMessages
DefineGlobalVariables
# Array 1 2 3 4

######################################################################
#                                                                    #
# StripCR                                                            #
#                                                                    #
# This function strips trailing carriage returns from a file.        #
#                                                                    #
######################################################################
StripCR ()
{
CALLED_FUNCTION="F-StripCR"

if [ $# -eq 0 ]
then
    echo "Usage: strip_CR <Input File> <Output File>"
    echo "Where trailing ^Ms in <Input File> are striped"
    echo "and the results placed in <Output File>"
    exit 1
fi
sed 's/$//' ${1} > ${2}
}
```

SCRIPT LAYOUT

Most shell scripts I see, even complex ones, are not written in a structured way. A long and complex script can be just as difficult to follow as any other program in other programming languages. By using functions in shell scripts it is possible to make the scripts modular and structured and hence easier to read and maintain. The example below is a skeleton of a shell script that illustrates this point.

EXAMPLE OF SCRIPT LAYOUT

```
#! /bin/ksh
######################################################################
#                                                                    #
# sample( ) script.sh                                               #
#                                                                   #
# This script illustrates the structure of a shell script.          #
#                                                                   #
# Input   : None                                                    #
#                                                                   #
# Notes   1   The script contains following functions:              #
#             - main                                                #
#             - InitializeVariables                                 #
#             - ProcessBody                                         #
#             - ProcessExit                                         #
#                                                                   #
# History                                                           #
# ----------------------------------------------------------------- #
# Date       Author         Description                             #
# ----------------------------------------------------------------- #
# 01/01/99   A Zaman        Initial build                           #
#                                                                   #
#####################################################################

#####################################################################
#                                                                   #
# InitializeVariables                                               #
#                                                                   #
# This function initializes all variables.                          #
#                                                                   #
# Input   :                                                         #
#                                                                   #
# Returns :                                                         #
#                                                                   #
# Notes                                                             #
#                                                                   #
#####################################################################
InitializeVariables ( )
{
TRUE=0
FALSE=1

SEC=0             # success exit code
FEC=1             # failure exit code
}

#####################################################################
#                                                                   #
# ProcessBody                                                       #
#                                                                   #
# This function carries out the bulk of the processing.             #
#                                                                   #
# Input   :                                                         #
#                                                                   #
# Returns : TRUE or FALSE                                            #
#                                                                    #
# Notes                                                              #
#                                                                    #
######################################################################
ProcessBody ( )
{
echo ""
}

######################################################################
#                                                                    #
# ProcessExit                                                        #
#                                                                    #
# This function removes any temporary files and ensures a            #
# "graceful" exit, including an exit code.                           #
#                                                                    #
# Input   : Exit code                                                #
#                                                                    #
# Returns :                                                          #
#                                                                    #
# Notes                                                              #
#                                                                    #
######################################################################
ProcessExit ( )
{
EXIT_CODE=”$1”

# remove temporary files
rm -r $TEMP_FILE

exit $EXIT_CODE
}

######################################################################
#                                                                    #
# main                                                               #
#                                                                    #
# This is the function that invokes all other functions.             #
#                                                                    #
# Input   :                                                          #
#                                                                    #
# Returns :                                                          #
#                                                                    #
# Notes                                                              #
#                                                                    #
######################################################################
main ( )
{
InitializeVariables
ProcessBody
ProcessExit $SEC
}

# invoke main function with all command line options
main
```

Arif Zaman
DBA/System Administrator
High-Tech Software Ltd (UK)

fixdist – keeping your system up-to-date

AIX Update Issue 26 (December 1997) briefly mentioned the existence of a freely available IBM tool called **fixdist**. This tool enables the user to download AIX patches from IBM's Internet service sites. With just a small amount of setting up (pointing at the relevant IBM Internet server and stating what type of firewall you have), **fixdist** provides a self-contained program for checking and downloading patches.

However, **fixdist** has its drawbacks. The database of fixes is in binary format and is, therefore, hard to read without the front-end provided (its format is not in the public domain either, so that, for instance, no *.h* file is available). Also missing is a command line interface, so that you must use either the ASCII version or the X-Window version.

To address these issues, I've written a small script that users of **fixdist** who wish to make further use of the information it provides may find useful. The script uses the database downloaded by **fixdist** (via the *strings* command) to check the latest version of fixes available from IBM. It then compares the list with information on software and fixes installed on the host system. This means that the **fixdist** database file must be on the same machine on which the script is run. The report produced displays only file sets that are not up-to-date with fixes available from IBM. You can then download the relevant fixes, if and when required.

If your installation has multiple systems and allows the remote execution of commands from the system that hosts **fixdist**, simply change the **lslpp –qcL…** line to **rsh <*target_host*> lslpp –qcL…** to get reports for other systems.

I have written the script to handle both AIX version 3 and 4. Despite the fact that AIX version 3 is now 'functionally stabilized', IBM still supports the distribution of patches for version 3 through **fixdist**. In the sample output and code below, note the use of the continuation character, '➤', to indicate that one line of code maps to more than one line of print.

SAMPLE OUTPUT

AIX Version 3

```
U423564 for object bos.obj is not installed on your system
U423569 for object bos.obj is not installed on your system
U423650 for object bos.obj is not installed on your system
U423651 for object bos.obj is not installed on your system
U423654 for object bos.obj is not installed on your system
```

AIX Version 4

```
For fileset bos.acct you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.6
For fileset bos.adt.base you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.7
For fileset bos.adt.debug you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.9
For fileset bos.adt.graphics you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.3
For fileset bos.adt.include you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.19
For fileset bos.adt.lib you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.2
For fileset bos.adt.libm you have version 4.2.1.0
  ➤ fixdist knows about 4.2.1.1
```

FIXDIST1.KSH

```
#!/bin/ksh
#set -x
#
# Shell script to check the installed version of file sets against
# information collected by fixdist.
#
# The script relies on two files from the fixdist database:
#
#  - f32db.d01 for AIX version 3
#  - f41db.d01 for AIX version 4
#
# Both files are in binary format.
#
# ARGUMENTS:
#
# The script takes an optional argument. If a file set name is
# supplied on the command line, then the script reports only
# on that file set. If no arguments are supplied, then it
# reports on all file sets.
#
LPPS=$1
AIX_TEST=`lslpp -ql | grep "bos.rte " | egrep "APPL|COMM" | wc -l`
if [[ $AIX_TEST -gt 1 ]]
  then
    AIX_VER=4
  else
    AIX_VER=3
  fi
#
# This portion deals with AIX version 4
#
if [[ $AIX_VER -eq 4 ]]
  then
    find / -name f41db.d01 -print | read DBLOC
    if [[ $DBLOC = "" ]]
      then
        echo "ERROR - no AIX version 4 fixdist database file found ...
              ➤  cannot continue"
        exit 1
      fi
    strings $DBLOC | awk -F"." '
    NF > 2 {
             print $0
           }' | sort -u > /tmp/$$.fixdist.lst
    lslpp -qcL $LPPS | awk -F":" '{ print $2 " " $3 }' | sort -u |
    ➤  while read a b
      do
        c=${b%.*}
        grep $a /tmp/$$.fixdist.lst| grep $a.[0-9] | grep $c
        ➤  > /tmp/$$.tmp
        WC=`wc -l /tmp/$$.tmp | awk '{ print $1 }'`
        if [[ $WC -gt 0 ]]
          then
            NOF=`head -1 /tmp/$$.tmp | awk -F"." '{ print NF }'`
            VER=`cat /tmp/$$.tmp | sort -t"." -krn$NOF | head -1 |
            ➤  awk -F"." '{
                             X=NF - 3
                             Y=NF - 2
                             Z=NF - 1
                             print $X"."$Y"."$Z"."$NF }'`
            if [[ $b != $VER ]]
              then
                echo "For file set " $a " you have version " $b "
                      ➤  fixdist knows about " $VER
              fi
            fi
      done
    else
    #
    # This portion deals with AIX version 3
    #
    find / -name f32db.d01 -print 2> /dev/null | read DBLOC
    if [[ $DBLOC = "" ]]
      then
        echo "ERROR - no AIX version 3 fixdist database file
              ➤  found...cannot continue"
        exit 1
      fi
    strings $DBLOC | grep U[0-9] | grep obj > /tmp/$$.fixdist.lst
    mkdir /tmp/$$.dir
    chmod 777 /tmp/$$.dir
    if [[ $LPPS = "" ]]
      then
        lslpp -lqc | awk -F":" '{ print $2 }' | sort -u | awk '{
        ➤  print $1 }' | while read fname
          do
            lslpp -lacq $fname | grep -v AVAILABLE | awk -F":" '{
            ➤  print $2":"$3":"$4}' | sort -u -o /tmp/$$.dir/$fname
          done
      else
        lslpp -lacq $LPPS | grep -v AVAILABLE | awk -F":" '{
        ➤  print $2":"$3":"$4}' | sort -u -o /tmp/$$.dir/$LPPS
      fi
    for i in `ls /tmp/$$.dir`
      do
        grep $i /tmp/$$.fixdist.lst | awk -F"." '{ print $NF }' |
        ➤  while read PTFNO
          do
            INSTALLED=`grep $PTFNO /tmp/$$.dir/$i | wc -l`
            if [[ $INSTALLED -eq 0 ]]
              then
                echo "$PTFNO for object $i is not installed on your
                ➤  system"
            fi
```

```
          done
        done
      rm -rf /tmp/$$.dir
    fi
rm -f /tmp/$$.tmp /tmp/$$.fixdist.lst > /dev/null 2>&1
```

Phil Pollard
Unix Systems Administrator (UK)

sysdoc – Web-based system documentation

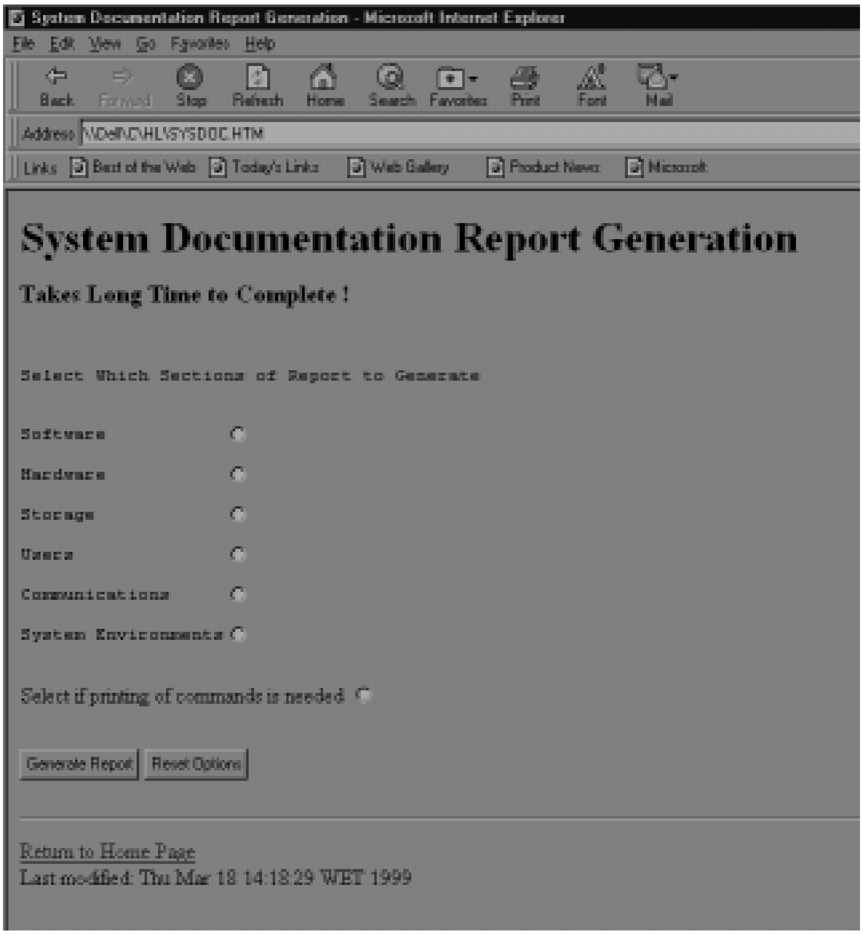
INTRODUCTION

System documentation scripts are useful at various stages during the development and maintenance of complex computer installations. To this end I have implemented a documentation system that is based on the Web. It consists of two parts: a CGI script, written for the Korn shell, and an HTML file that invokes the script. The system gathers a large amount of system configuration data, which is intended to be printed and kept off line or saved as text in order to allow comparisons of different configurations using standard Unix utilities, such as **diff**. The HTML file displays a form that allows the user to select particular types of system information for display.

I have tested the system using AIX 4.1, AIX 4.2, and AIX 4.3. The CGI script is easily extensible to include additional data, such as information on additional installed software (ADSM, HACMP) and hardware (tape libraries, SSA disks).

I used Lotus's Go Web server, which is supplied as a part of standard AIX distribution, to host the Web page. The CGI script is called **sysdoc** and should be installed in the *cgi-bin* subdirectory of the Web server that runs on your computer (in my case, the directory is */usr/lpp/internet/server_root/cgi-bin*). *sysdoc.html* may be located in any directory to which access is provided via your Web server (in my case */usr/lpp/internet/server_root/pub*). You should set **sysdoc**'s permissions to '755' and *sysdoc.html*'s to 555.

Figure 1 shows *sysdoc.html* viewed using Internet Explorer.


System Documentation Report Generation - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail
Address \\Dell\C\HL\SYSDOC.HTM
Links Best of the Web Today's Links Web Gallery Product News Microsoft
Generate Report
Reset Options

System Documentation Report Generation

Takes Long Time to Complete !

Select Which Sections of Report to Generate

Software ○

Hardware ○

Storage ○

Users ○

Communications ○

System Environments ○

Select if printing of commands is needed ○

Return to Home Page
Last modified: Thu Mar 18 14:18:29 WET 1999

Figure 1: sysdoc.html viewed using Internet Explorer

Note the use of the continuation character, '➤', in the listings below to indicate that one line of code maps to more than one line of print.

SYSDOC.CGI

```
#!/usr/bin/ksh
#
# Print system documentation for RS/6000 running AIX 4
#
#
function printtitle
{
echo "<P><TITLE>"
echo $*
echo "</TITLE>"
}

function printhead
{
echo "<P><H1>"
echo $*
echo "</H1>"
}

function printhead2
{
echo "<P><H2>"
echo $*
echo "</H2>"
}

function printhead3
{
echo "<P><H3>"
echo $*
echo "</H3>"
}

function printtext
{
echo "<P><PRE>"
echo $* | awk '{print "<BR>" $0}'
echo "</PRE>"
}

function printcom
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "Following output has been produced by command: "
  echo "<I>"
  echo $*
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $* | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printcom2
{
if [[ "$VERBOSE" = "TRUE" ]]
```

```
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo $1 '|' $2
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $1 | $2 | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printcom_trim
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo $*
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $* | sed '/^#.*/d' | tr ':' '\011' | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printfile
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo "cat $1 | sed '/^[#:\*].*/d'"
  echo "</B></I>"
fi
  echo "<P><PRE>"
  cat $1|  sed '/^[#:\*].*/d' | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

#
# Print 2 lines of mandatory output from CGI script
#
echo 'Content-type: text/html'
echo
#
# Start of the main program
#
```

```
#############################################################
HOSTNAME=$(hostname)
DATE=$(date)

IT="<I>"
E_IT="</I>"
PDISKS=`lspv| awk '{print $1}'`
VGS=`lsvg -o`  # List VG infor for available VG's only !
#
# Options Setting
#
cc=`echo $QUERY_STRING|grep VERBOSE|wc -c`
if [ $cc -gt 0 ]
then
 VERBOSE="TRUE"
fi

cc=`echo $QUERY_STRING|grep SOFTWARE|wc -c`
if [ $cc -gt 0 ]
then
 SOFTWARE="TRUE"
fi

cc=`echo $QUERY_STRING|grep HARDWARE|wc -c`
if [ $cc -gt 0 ]
then
 HARDWARE="TRUE"
fi

cc=`echo $QUERY_STRING|grep STORAGE|wc -c`
if [ $cc -gt 0 ]
then
 STORAGE="TRUE"
fi

cc=`echo $QUERY_STRING|grep USERS|wc -c`
if [ $cc -gt 0 ]
then
 USERS="TRUE"
fi

cc=`echo $QUERY_STRING|grep COMMS|wc -c`
if [ $cc -gt 0 ]
then
 COMMS="TRUE"
fi

cc=`echo $QUERY_STRING|grep SYSENV|wc -c`
if [ $cc -gt 0 ]
then
```

```
 SYSENV="TRUE"
fi

if [[  "$SOFTWARE" = "TRUE" || "$HARDWARE" = "TRUE" ||  "$STORAGE" =
➤  "TRUE" ||  "$USERS" = "TRUE" || "$COMMS" = "TRUE" ||  "$SYSENV" =
➤  "TRUE" ]]
then
  printtitle Configuration information for host $HOSTNAME on $DATE
  printhead Configuration information for host $IT$HOSTNAME$E_IT on
  ➤  $IT$DATE$E_IT
else
  printhead2 "No Sections Selected"
fi

if [[ "$SOFTWARE" = "TRUE" ]]
then
  printhead2 Software

  printhead3 Level of AIX Operating System
  printcom oslevel

  printhead3 Installed Software
  printcom lslpp -l
fi

if [[ "$HARDWARE" = "TRUE" ]]
then
  printhead2 Hardware

  printhead3 System Parameters
  printcom lsattr -E -H -l sys0

  printhead3 Detailed Hardware Configuration
  printcom lscfg -v

  printhead3 Installed Adapters
  printcom lsdev -C
fi

if [[ "$STORAGE" = "TRUE" ]]
then
  printhead2 Storage

  printhead3 Installed Physical Disks
  printcom  lspv

  printhead3 Logical Volumes Distribution per Physical Disk
  for i in $PDISKS
  do
   printcom lspv -l $i
```

```
done

printhead3 Physical Partitions Distribution per Physical Disk
for i in $PDISKS
do
 printcom lspv -p $i
done

printhead3 Physical Partitions Distribution per Logical Volume
for i in $PDISKS
do
 printcom lspv -M $i
done

printhead3 Volume Groups
printcom lsvg

printhead3 Online Volume Groups
printcom lsvg -o

printhead3 Volume Groups Characteristics
for i in $VGS
do
 printcom lsvg $i
done

printhead3 Physical Disks Distribution per Volume Group
for i in $VGS
do
 printcom lsvg -p $i
done

printhead3 Logical Volumes Distribution per Volume Group
for i in $VGS
do
 printcom lsvg -l $i
done

printhead3 Logical Volumes Characteristics
for j in $VGS
do
 LVS=`lsvg -l $j|grep '/'|awk '{print $1}'`
 for i in $LVS
 do
  printcom lslv $i
 done
done

printhead3 Logical Volumes Distribution on Physical Disk
for j in $VGS
```

```
  do
   LVS=`lsvg -l $j|grep '/'|awk '{print $1}'`
   for i in $LVS
   do
    printcom lslv -l $i
   done
  done

  printhead3 Logical Volumes Allocation Map

  for j in $VGS
  do
   LVS=`lsvg -l $j|grep '/'|awk '{print $1}'`
   for i in $LVS
   do
    printcom lslv -m $i
   done
  done

  printhead3 Paging Space Layout and Utilization
  printcom lsps -a

  printhead3 File Systems
  printcom lsfs -a -q

  printhead3 Mounted File Systems
  printcom mount

  printhead3 /etc/filesystems
  printfile /etc/filesystems
fi

if [[ "$USERS" = "TRUE" ]]
then
  printhead2 Users Information

  printhead3 Users
  printtext "Name    Id   Group(s) Home Directory   Shell"
  printcom_trim lsuser -c  ALL

  printhead3 Groups
  printtext "Name  Id Admin Members"
  printcom_trim lsgroup -c  ALL
fi

if [[ "$COMMS" = "TRUE" ]]
then
  printhead2 Communications

  printhead3 TCP/IP
```

```
printhead3 Hostname
printcom hostname

printhead3 Arp Table
printcom arp -a

printhead3 Routing Table
printcom netstat -rn

printhead3 Network Interfaces
printcom lsdev -C -c if

printhead3 Name Resolution /etc/hosts
printfile /etc/hosts

if [[ -f /etc/resolv.conf ]]
then
  printhead3 Name Resolution /etc/resolv.conf
  printfile /etc/resolv.conf
fi

printhead3 Client Network Services
printhead3 /etc/services
printfile /etc/services

printhead3 /etc/protocols
printfile /etc/protocols

printhead3 /etc/syslog.conf
printfile  /etc/syslog.conf

printhead3 Server Network Services
if [[ -f /etc/hosts.equiv ]]
then
  printhead3 Remote Host Access Control /etc/hosts.equiv
  printfile /etc/hosts.equiv
fi

if [[ -f /etc/ftpusers ]]
then
  printhead3 Local User Names NOT To Be Used by Remote FTP clients
  ➤  /etc/ftpusers
  printfile /etc/ftpusers
fi

if [ `mount|grep nfs|wc -l` -gt 0 ]
then
  printhead3 Directories Mounted thru NFS
  printcom mount|grep nfs
fi
```

```
  if [[ -f /etc/exports ]]
  then
    printhead3 Local Directories Exported by NFS
    printfile /etc/exports
  fi
fi

if [[ "$SOFTWARE" = "SYSENV" ]]
then
  printhead2 System Environments

  printhead3 /etc/inittab
  printfile /etc/inittab

  printhead3 Subsystems
  printcom lssrc -a

  printhead3 TimeZone
  printtext $TZ
fi
```

SYSDOC.HTML

```
<html>
  <head>
    <title>System Documentation Report Generation</title>
  </head>

  <body>
    <H1>System Documentation Report Generation</H1>
    <H3><BLINK>Takes Long Time to Complete !</BLINK></H3>

<FORM action="cgi-bin/sysdoc" method="GET">

<BR>
<PRE>
Select Which Sections of Report to Generate

<BR>Software           <INPUT TYPE="radio" NAME="SOFTWARE">
<BR>Hardware           <INPUT TYPE="radio" NAME="HARDWARE">
<BR>Storage            <INPUT TYPE="radio" NAME="STORAGE">
<BR>Users              <INPUT TYPE="radio" NAME="USERS">
<BR>Communications     <INPUT TYPE="radio" NAME="COMMS">
<BR>System Environments<INPUT TYPE="radio" NAME="SYSENV">
</PRE>
<BR>
Select if printing of commands is needed
<INPUT TYPE="radio" NAME="VERBOSE">
<HR>
```

```
<INPUT TYPE="submit" VALUE="Generate Report">
<INPUT TYPE="reset" VALUE="Reset Options">
</FORM>
<HR>
<!-- Created: Thu Mar 18 10:44:44 WET 1999 -->
<A HREF="/home.html">Return to Home Page</A>
<BR>
<!-- hhmts start -->
Last modified: Thu Mar 18 14:18:29 WET 1999
<!-- hhmts end -->
</body>
</html>
```

A Polak
System Engineer
APS (Israel)

A utility to implement a ‘highlighted calendar’

today.sh is a shell script that displays the calendar for the current month with highlighted blinking focus on today’s date.

TODAY.SH

```
######################################################################
#
#  Name     : today.sh
#
#  Overview : The script displays the calendar for the current
#             month, highlighting the current day.
#
#  History  :
#  Date       Name       Description
#  ------------------------------------------------------------------
#  02/02/99   A Zaman    Initial Build
#
######################################################################

######################################################################
#
#  Name     : InitializeVariables
```

```
#
#  Overview : The function initialises all variables.
#
######################################################################
InitializeVariables ( )
{
BON=[[5m
BOFF=[[25m
RVON=[[7m                  # reverse video on
RVOFF=[[27m                # reverse video off
TODAY=`date +%d`
}

######################################################################
#
#  Name     : ChangeDayFormat
#
#  Overview : The function changes the format of two-digit days
#             (with leading zeros).
#
######################################################################
ChangeDayFormat ( )
{
if [ "${TODAY}" = "01" -o  "${TODAY}" = "02" -o "${TODAY}" = "03" -o \
     "${TODAY}" = "04" -o  "${TODAY}" = "05" -o "${TODAY}" = "06" -o \
     "${TODAY}" = "07" -o  "${TODAY}" = "08" -o "${TODAY}" = "09" ]
then
    TODAY=`echo $TODAY | cut -c2-2`
fi
}

######################################################################
#
#  Name     : DisplayCalendar
#
#  Overview : This function displays the calendar formatted using the
#             sed command.
#
######################################################################
DisplayCalendar ( )
{
if [ "${TODAY}" = "1" -o "${TODAY}" = "2" -o "${TODAY}" = "3" -o \
     "${TODAY}" = "4" -o "${TODAY}" = "5" -o "${TODAY}" = "6" -o \
     "${TODAY}" = "7" -o "${TODAY}" = "8" -o "${TODAY}" = "9" ]
then
   #
   # format sed command for one-digit day
   #
   cal | sed s/^/'                   '/ | \
                   sed s/" $TODAY  "/" $RVON$BON$TODAY$RVOFF$BOFF  "/
else
   #
   # format sed command for two-digit day
   #
   cal | sed s/^/'                   '/ | \
                   sed s/$TODAY/$RVON$BON$TODAY$RVOFF$BOFF/
fi
}

######################################################################
#
#  Name     : main
#
#  Overview : The function main invokes all other functions.
#
######################################################################
main ( )
{
InitializeVariables
ChangeDayFormat
DisplayCalendar
}

#
# invoke main
#
main
```

SAMPLE OUTPUT

```
     February 1999
     Sun   Mon   Tue   Wed   Thu   Fri   Sat
             1     2     3     4     5     6
       7     8     9    10    11    12    13
      14    15    16    17    18    19    20
      21    22    23    24    25    26    27
      28
```

Arif Zaman
High-Tech Software Ltd (UK)

Freeware for AIX

Many programmers and system administrators are aware that high-quality open source software can be downloaded for a wide variety of applications. Some applications, such as compilers, word processors, and database management systems, are dominated by commercial products. Others, such as Web servers, scripting languages, and e-mail packages, tend to be dominated by open source software.

So how do you learn about useful free software? Mostly in the same way as you would learn about any other useful software or information: talk to co-workers, ask experts, read magazines, search and participate in FAQs and the USENET, attend conferences, and surf the Web. To make things easier, several Web sites have organized collections of free software of potential interest.

One of these sites is *http://www.bull.de/pub/*. Their freeware and shareware archive proudly announces itself as the world's first archive of **smit**-installable freeware for AIX 4. The packages available are self-extracting, which basically means that you download them, execute the downloadable to allow it to extract itself, and then install the extracted package using AIX's infamous **smit**.

Below is a list of the utilities that I think are most useful and the sort of user that would be most likely to use them.

WORKGROUP SERVER PACKAGES

This category includes software packages that would typically be used to provide services to a group of users or to implement a small (departmental) application.

- *Samba v2.0.0 SMB client and server for Unix*
  Samba is a freeware utility that allows PC users to access Unix disk and printer resources without having to install NFS on the PC.

- *Apache v1.3.4 HTTP server*
  Apache is a freeware Web server that's well supported in the

freeware world. It can integrate closely with Perl to create stable, high-performance applications.

- *MySQL v3.22.14 SQL database server*
  MySQL is a multi-thread SQL database engine. It can be accessed directly from Perl using the Perl database interface driver.

- *Perl v5.5.2 scripting language*
  The Perl scripting language is widely used for writing CGI programs for Web servers. It's also used as a regular scripting language for automating administrative tasks. Perl is rapidly becoming the preferred scripting language for all applications, as it's available on a wide range of platforms (including Unix, Mac, and Windows).

- *Squid v1.1.20 Web proxy server*
  Squid is a high-performance caching Web proxy server.

- *FTPWeblog v1.0.2 Web and FTP server statistics package*
  FTPWeblog generates graphical statistics of Web server usage. It's easy to use and provides good, intelligent analysis of who's accessing which resources on your Web server.

- *Wget v1.5.3 Web file retrieval*
  Wget is a powerful tool for downloading individual Web pages or entire Web sites. It's typically used to 'mirror' a server or part of a server. Wget understands both HTTP and FTP URLs, and can work through Web or Socks proxy servers.

- *Weblint v1.20 'Lint' program for HTML*
  Weblint can be used to check HTML documents for syntax errors. It should be used when HTML documents are modified manually.

ADMINISTRATIVE TOOLS

This category comprises tools that are of specific use to system administrators.

- *Lsof v4.38 list open files*
  Lsof is an essential utility on any modern Unix system. It is particularly useful in answering two difficult questions:

  - Which users are using a filesystem? To unmount a filesystem under Unix, all users must first stop using it. However, it's often difficult to identify which users have left processes running in a particular filesystem. Lsof allows processes that are still using a disk resource to be identified easily.

  - Which processes are using the TCP/IP stack? When debugging a network problem, it's often vital to be able to analyse which processes are actively using the TCP/IP stack. Lsof can identify them quickly and easily.

- *Monitor v2.1.5 performance monitor*
  Monitor is another 'life-saving' utility. It allows all the major performance indicators to be displayed simultaneously on a simple ASCII screen (or xterm). Monitor combines the information available from many standard tools (**vmstat**, **iostat**, **netstat**, and others) to provide an immediate view of system performance. Monitor also benefits from the fact that it can be executed by non-root users.

- *Tidysys v2.2.1 system maintenance tool*
  On a normal Unix system, there are files that accumulate under */tmp*, including log files that must occasionally be reduced in size, etc. Tidysys allows all the standard AIX files that need to be maintained in this way to be kept to a reasonable size. In addition, log files from add-on products can also be maintained using Tidysys.

- *AIX Tools v1.5.1 command-line tools*
  The LPP *freeware.aix.tools.rte* contains a range of small utilities from different sources. Among the tools available, **whichlpp** shows which LPP delivered a file, **pstree** displays processes in tree format, **ll** is equivalent to **ls -l**, **ldd** lists the shared library dependencies of a program better than **dump -H**, **xd** is a hexadecimal dump utility that's better than **od -x**, and **chpass** is a batch password modification program.

- *Satan v2.0.1 security analyst*
  Satan (Security Administrator Tool for Analysing Networks) allows Unix systems on a network or subnet to be probed for

externally visible security problems. It's a powerful and easy-to-use tool, recently updated to conform with ITCS201 recommendations. All externally-visible Unix systems should be tested with Satan several times a year.

- *COPS v1.0.4 security checker*
  COPS looks for security configuration problems on the machine where it is installed and executed. This allows an administrator to verify that no errors have been made that could allow a local user to become the superuser. All externally-visible Unix machines should be tested with COPS once a month.

- *Tiger v2.2.3.0 security checker*
  Tiger performs essentially the same job as COPS. Both can be installed and used to double-check results.

- *Crack v5.0 and Jonn v1.5 password cracking tools*
  Crack verifies that passwords for user logins (including *root*) are difficult to guess. Crack is mostly used on machines that host many user accounts to verify that users choose passwords that don't contain their name, commonly used passwords (such as *password*), etc.

END-USER TOOLS

These are tools that would be directly useful to real live users with access to the system from a shell.

- *Gzip v1.2.4 file compression tool*
  Gzip is the default compression tool on the Internet.

- *Screen v3.7.4 ASCII multi-screen utility*
  Screen is a great tool if you use dumb ASCII terminals. It allows you to have several applications running on the same terminal, each of which thinks it controls a real terminal of its own. Users can switch from one application to another, allowing them, for instance, to use **vi** on one screen, perform a back-up on another, 'telnet' to another machine on a third, etc.

- *Mtools v3.9.1 utilities to access DOS disks from Unix*
  Mtools is an absolute necessity if you exchange diskettes with PC

users. Mtools provides commands with the same name and syntax as standard DOS commands, bar the fact that the names of Mtools commands are prefixed with an ‘m’ (**mcopy** instead of **copy**, **mdir** instead of **dir**, etc).

- *Xpdf v0.8 PDF viewer for X11*
  Xpdf is a smaller and faster version of the Acrobat Reader.

- *Pine v4.05 e-mail utility*
  Pine is a powerful yet easy-to-use e-mail client. While Pine is an ASCII application, it’s very quick and uses its own easy-to-use custom text editor to prepare mail messages. Pine is fully MIME-compliant, and allows text attachments to be displayed directly.

- *Unzip v5.32 and Zip v2.2.0 file compression and packing tools*
  Zip and Unzip allow ‘zip files’ (widely used on PCs) to be created (‘zipped’) and unzipped under Unix. The zip files created are compatible with the PKZIP utility on a PC.

- *Xpaint v2.5.5 image editing tool*
  Xpaint is a good tool for generating and editing images, though it’s not up to the standard of something like PaintShop Pro on a PC. Xpaint also allows images to be converted from one format to another.

- *xv v3.10.1 XV image viewer*
  XView is a shareware utility that provides powerful image viewing tools. XView can be used to capture screen images, which can then be edited and saved to disk in a number of formats.

DEVELOPMENT TOOLS

Packages in this category include all those used for software development.

- *EGCS v1.1.1 GNU C compiler*
  GCC, the GNU C and C++ freeware compilers, are probably the most widely used compilers in the world, and set the standard when it comes to portability. The GNU C compiler is a fully ANSI-C compliant compiler. It’s customary for every Unix machine to have a C compiler, but not everybody needs a

commercial product. The GNU C compiler provides basic functionality for users who are not developing commercial software applications.

Note that a commercial equivalent to GCC, such as IBM's XLC compiler, typically produces smaller, faster binaries and is also more likely to stay current with the operating system and hardware.

- *TCL/TK v8.0 scripting language tools*
  This is an implementation of the TCL scripting language.

- *Mklpp v1.2.3 LPP generation tool*
  Mklpp is the tool used to generate freeware LPPs found on CD-ROMs. Mklpp can be used to quickly and easily generate professional installation images for any AIX 4 machine. It can be used for in-house deployment of applications or to package a commercial application for easy installation by your customers.

INTERNET COLLECTION CD-ROM

Bull is to make available an 'Internet Collection CD-ROM' that contains almost all the utilities from the AIX 4.1.5 and later sections of their freeware and shareware archive. This will provide over 350 MBs of free software that is installable with **smit**. The large majority of it will also work on AIX 4.2 and AIX 4.3 thanks to AIX's strong cross-version compatibility. Source code will be included, where possible.

Werner Klauser
Klauser Informatik (Switzerland)

AIX news

IBM has announced Licence Use Management, a toolkit that gives software vendors and end-users the ability to manage the use of their applications. It supports various software licensing models, providing run-time monitoring of software assets, a mechanism to control compliance with software contracts, an ability to migrate software assets to alternative pricing models, and protection of software assets available on CD-ROM or via electronic distribution for a trial period.

Available now, the product costs US$6,000 for the AIX version, US$1000 for OS/2 and Windows NT, and US$12,500 for HP-UX, Solaris, and SGI IRIX.

For further information contact your local IBM representative.

* * *

Software AG has announced a new version of its DCOM-based EntireX middleware, which now has a security system that allows co-operation between NT and mainframe security. The authentication procedure uses Microsoft security standards, while the authorization procedure for remote services uses the host system's standards. Also new is cross-platform interoperability between different security systems without the need to use proprietary APIs.

It's out now, with versions for AIX, OS/390, HP-UX, Digital Unix, OpenVMS, Solaris, NT, and Windows 95. No prices were announced.

For further information contact:
Software AG, 11,190 Sunrise Valley Drive,
Reston, VA 22091, USA
Tel: +1 703 860 5050
Fax: +1 703 391 6975
Web: http://www.softwareag.com

Software AG (UK) Ltd, Charter Court, 74-78 Victoria Street, St Albans, AL1 3XH, UK
Tel: +44 1727 844455
Fax: +44 1727 840092

* * *

Innosoft has announced the Innosoft Distributed Directory Server (IDDS) Version 5.0 and the Innosoft LDAP Proxy Server (ILPS) Version 2.0. IDDS V5 provides high availability for LDAP 3 servers by keeping primary and secondary servers in sync, resynchronizing them should one fail then be brought back on-line. ILSP provides such features as load balancing and failover for high availability LDAP servers. Both products are expected shortly and will run on AIX 4.3 (PowerPC), NT (Intel only), HP-UX, and Digital Unix. UK prices for IDDS V5 range from £1,475 for 1000 entries to £78,000 for one million, and prices for the ILPS range from £4,000 for 15 concurrent connections to £10,000 for an unlimited number.

For further information contact:
Innosoft International, 1050 Lakes Drive, West Covina, CA 91790, USA
Tel: +1 626 919 3600
Fax: +1 626 919 3614
Web: http://www.innosoft.com

Essential Computing Limited, PO Box 49, Clevedon, Bristol BS21 7NB, UK
Tel: +44 1275 343199
Fax: +44 1275 340974

xephon