51 AIX

January 2000

upgrade

In this issue

Published by

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

Distributing user passwords to AIX servers

BACKGROUND.

The *passwd* file in the */etc/security* directory contains entries for user passwords. Password entries in this file are built using three random characters generated by AIX. This means that on different sites the same password results in different entries in the */etc/security/passwd* file.

However, if you transfer a password from one AIX machine to another, you will get the same password on both machines. This is an easy way to distribute user passwords to all AIX systems.

I developed two scripts that make use of this. The benefits of using these scripts are:

- You've always got the same password on all AIX servers.
- You have to enter the password on only one machine.
- You only have to enter your password twice (including confirmation), which reduces typing errors.
- Automating the replication process saves time.
- The administrator doesn't know the user's password, which is probably the same as the network password.

DETAILS OF THE /ETC/SECURITY/PASSWD FILE

This file contains the following entries for each user:

```
user_identification:
      password = password_string
      lastupdate = update_string
      flags = flag_string
```

If the user has never logged in, *password_string* will be an asterisk ('*') and the other two entries (*lastupdate* and *flags*) will be missing.

When replicating passwords, the *lastupdate* entry has to be changed, as it contains date and time information necessary to calculate when

the password has to be changed. The entry flags also have to be changed, as AIX uses them to check whether the password has to be changed immediately on next logon (*ADMCHG*).

Whenever the password file changes, you must copy it to */etc/security/opasswd* to ensure that you conform to AIX standards.

TO CHANGE A PASSWORD ON ALL AIX SERVERS

- The user changes his or her password on the machine where the script **dist_passwd** resides.
- Someone logs on as root and starts the script **dist_passwd** ('distribute password'). This script takes two parameters: the user-id of the user whose password has changed and root's password.

The script searches the password file for an entry belonging to the nominated user. If an entry is not found, an error message is generated and execution stops. If the entry is found, the script creates a temporary file and enters a loop; during each iteration, the temporary file is sent to an AIX server using FTP. On reception of the file, the remote server starts the remote execution of the second script, **passwd_chg**. This script reads the temporary file and updates the password file (if it contains the user entry). It also creates a result file. After executing the script, it transfers the result file (using FTP) and displays the result on your monitor.

NOTES

1 If the user doesn't exist on an AIX machine, then the user has not been created on that machine.

2 The two scripts use other scripts that are published in other editions of *AIX Update*:

   – **testftp**, which checks the success of an FTP operation (see *AIX Update*, March 1999)

   – **logmsg**, a 'script logging system' (see *AIX Update*, March 1997).

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

DIST_PASSWD

```
# Name script : /home/oper/dist_passwd
# Last change : 10-06-99 pst creation script
# Description : Distribute a password on this machine to all other
#               AIX servers
#-------------------------------------------------------------------

if [ $# -ne 2 ]
then echo "\nMissing parameters: user and/or root's password\n"
     echo "Example: dist_passwd pos9909 password_of_root\n"
     exit
fi

if [ "$*" = "?" ]
then echo "\nRequired parameters : user"
     echo "                     : password of root"
     echo "Example: dist_passwd pos9909 password_of_root\n"
     exit
fi

if [ `whoami` != "root" ]
then
      echo "\nOnly root is allowed to execute this script:"
      echo "After the script terminates, please logon as root.\n"
      exit 1
fi

# The next lines prevent damage to the file /.netrc:
# check whether another script is using FTP;
# if yes, then wait until it stops.
while [ -f /home/data/netrc.busy ]
      do
      n1=`cat /home/data/netrc.busy`
      echo "FTP of project $n1 is busy, waiting one minute."
      sleep 60
      done

# we're using FTP, so lock FTP to other scripts
echo "change_passwd" > /home/data/netrc.busy

# Initialize variables
my_machine=`hostname`
user_found="no"
number=0
ftp_ok=0
```

```
ftp_error=0
rexec_ok=0
rexec_error=0
passwd_ok=0
passwd_error=0

while read variable contains var_string
   do
   if [ "$variable" = "$1:" ]
   then
        # We've got the user section, set user_found to get entries
        user_found="yes"
        user=$1
   fi
   if [ "$user_found" = "yes" ]
   then
        case "$variable" in
        password   )
            password=$var_string
            ;;
        lastupdate )
           # Last entry in the user section, fill the temp file
           echo "$user $password $var_string" >/tmp/dist_passwd
           # Get the current time (needed for statistics)
           begin_time=`date +"%H%M%S`
           echo "Time       Output commands"
           # Go to all servers exept this one
           for site in `cat /home/data/all_aix_servers | grep -v
           ➤  $my_machine`
           do
               let number=$number+1
               echo "`date +"%H:%M:%S"` - Distributing password
               ➤  entries to $site"
               # Create the FTP file /.netrc
               echo "machine $site login root password $2"
               ➤  >/tmp/passwd_ftp.netrc
               echo "macdef init"
               ➤  >>/tmp/passwd_ftp.netrc
               echo "put /tmp/dist_passwd"
               ➤  >>/tmp/passwd_ftp.netrc
               echo "quit"
               ➤  >>/tmp/passwd_ftp.netrc
               echo ""
               ➤  >>/tmp/passwd_ftp.netrc
               cp /tmp/passwd_ftp.netrc  /.netrc
               chmod 600                 /.netrc
               # Make sure you've got no output of previous FTP
               rm /tmp/passwd_ftp_outp 1>/dev/null 2>&1
               # Use FTP to sent the temporary password file
               ftp -v $site 1>/tmp/passwd_ftp_outp 2>&1
```

```
# Check that the FTP succeeded
testftp /tmp/passwd_ftp_outp $site sys
RETC=$?
if [ "$RETC" -ne 0 ]
then
     let ftp_error=$ftp_error+1
     let passwd_error=$passwd_error+1
     echo "`date +"%H:%M:%S"` - FTP to $site failed"
     echo "`date +"%H:%M:%S"` - Password has NOT changed
     ➤  at $site. \n"
     echo "FTP to $site failed"
     ➤  >>/tmp/passwd_err
     echo "Information follows:"
     ➤  >>/tmp/passwd_err
     cat  /tmp/passwd_ftp_outp
     ➤  >>/tmp/passwd_err
     echo "Remote command has not been executed"
     ➤  >>/tmp/passwd_err
     echo "Password NOT changed!\n"
     ➤  >>/tmp/passwd_err
else
     let ftp_ok=$ftp_ok+1
     echo "`date +"%H:%M:%S"` - The password will be
     ➤  changed using the remote command passwd_chg"
     # call on the remote site the script which actually
     # changes the password
     rexec $site /home/oper/passwd_chg
     ➤  2>/tmp/passwd_err2
     RETC=$?
     if [ "$RETC" -ne 0 ]
     then
          let rexec_error=$rexec_error+1
          let passwd_error=$passwd_error+1
          echo "`date +"%H:%M:%S"` - Rexec command to
          ➤  $site failed"
          echo "`date +"%H:%M:%S"` - Password NOT
          ➤  changed! \n"
          echo "Rexec command to $site failed"
          ➤  >>/tmp/passwd_err
          echo "Returncode=$RETC"
          ➤  >>/tmp/passwd_err
          echo "Information follows:"
          ➤  >>/tmp/passwd_err
          cat /tmp/passwd_err2
          ➤  >>/tmp/passwd_err
          echo "Password NOT changed!\n"
          ➤  >>/tmp/passwd_err
          rm /tmp/passwd_err2
     else
          let rexec_ok=$rexec_ok+1
```

```
                echo "`date +"%H:%M:%S"` - Getting the result
                ➤  of the remote command"
                echo "machine $site login root password $2"
                ➤  >/.netrc
                echo "macdef init"                  >>/.netrc
                echo "get /tmp/dist_passwd.return"  >>/.netrc
                echo "quit"                         >>/.netrc
                echo ""                             >>/.netrc
                # Get the result of the remote exec command
                ftp -v $site 1>/dev/null 2>&1
                # Display the result
                echo "`date +"%H:%M:%S"` - \c"
                cat /tmp/dist_passwd.return
                # statistics
                check_it=`cat /tmp/dist_passwd.return | grep
                ➤  NOT`
                rm /tmp/dist_passwd.return
                if [ -z "$check_it" ]
                then
                     let passwd_ok=$passwd_ok+1
                else
                     let passwd_error=$passwd_error+1
                fi
                echo
           fi
      fi
      rm /tmp/passwd_ftp_outp
      done
end_time=`date +"%H%M%S`
# Remove temporary files
rm /tmp/dist_passwd
rm /home/logging/sys.log 2>/dev/null
rm /.netrc               2>/dev/null
rm /home/data/netrc.busy 2>/dev/null
if [ -s /tmp/passwd_err ]
then
     # There are errors: display them
     pg /tmp/passwd_err
     rm /tmp/passwd_err
fi
# Show statistics
begin_hour=`echo $begin_time |cut -c1-2`
begin_min=`echo $begin_time |cut -c3-4`
begin_sec=`echo $begin_time |cut -c5-6`
end_hour=`echo $end_time |cut -c1-2`
end_min=`echo $end_time |cut -c3-4`
end_sec=`echo $end_time |cut -c5-6`
let number_hour=$end_hour-$begin_hour
let number_min=$end_min-$begin_min
let number_sec=$end_sec-$begin_sec
```

```
            if [ $number_sec -lt 0 ]
            then
                 let number_min=$number_min-1
                 let number_sec=60+$number_sec
            fi
            if [ $number_min -lt 0 ]
            then
                 let number_hour=$number_hour-1
                 let number_min=60+$number_min
            fi
            if [ $number_hour -lt 0 ]
            then
                 let number_hour=$number_hour+24
            fi
            let average=\($number_sec+60*$number_min+3600*number_hour\)
            ➤  /$number
            echo "Statistics:"
            echo "Number of sites processed         : $number"
            echo "Number of ftps ok                 : $ftp_ok"
            echo "Number of wrong ftps              : $ftp_error"
            echo "Number of rexecs ok               : $rexec_ok"
            echo "Number of wrong rexecs            : $rexec_error"
            echo "Number of passwords changed       : $passwd_ok"
            echo "Number of passwords not changed   : $passwd_error"
            echo "Average processing time / site : $average seconds"
            exit 0   ;;
         * ) ;;
         esac
fi
done </etc/security/passwd
# remove temporary files
rm /home/logging/sys.log 2>/dev/null
rm /.netrc               2>/dev/null

# remove lock FTP
rm /home/data/netrc.busy 2>/dev/null

echo "User \"$1\" not found in /etc/security/passwd"
echo "Please check user \"$1\"! "

exit 1
```

PASSWD_CHG

```
# Name script : /home/oper/passwd_chg
# Last change : 22-06-94 pst creation
# Description : Change user entry in /etc/security/passwd
#-------------------------------------------------------------------
remove_ADM ()
```

```
{
# remove ADMCHG from a string of flags
case $# in
   0 ) ;;
   * ) var_string=""
       first="yes"
       while [ $# -gt 0 ]
       do
            if [ "$1" != "ADMCHG" ]
            then
                 if [ "$first" = "yes" ]
                 then
                      var_string="$1"
                      first="no"
                 else
                      var_string="$var_string,$1"
                 fi
            fi
            shift 1
       done
       ;;
esac
}

# Get user and entries that have to be replaced
read user password lastupdate </tmp/dist_passwd

# Make new password file
>/etc/security/new_password_file

user_found="no"
while read variable contains var_string
     do
     if [ "$variable" = "$user:" ]
     then
          # We've got the user, replace the entries in the next loops
          user_found="yes"
          user_processed="yes"
     fi
     if [ "$user_found" = "yes" ]
     then
          case "$variable" in
              password     )
                  if [ "$var_string" != "*" ]
                  then
                       # User is initialized, so replace password
                       var_string=$password
                  else
                       # User has never been initialized
                       # Add user entries (add password + lastupdate)
```



```
                    echo "        password = $password"
                    ➤  >>/etc/security/new_password_file
                    echo "        lastupdate = $lastupdate"
                    ➤  >>/etc/security/new_password_file
                    # Initialize next variable, so it can be written
                    # in the loop
                    variable="flags"
                    var_string=""
                    # Finish changing user, and don't alter other
                    # entries
                    user_found="no"
                fi ;;
            lastupdate )
                var_string=$lastupdate  ;;
            flags      )
                remove_ADM `echo $var_string | tr ',' ' '`
                # Normal last entry in the user section
                # Do not change other users
                user_found="no"   ;;
            * ) ;;
        esac
    fi
    # Is this a user section?
    check_user_entry=`echo $variable | grep :`
    if [ -n "$check_user_entry" ]
    then
        # Yes, so fill the file with user section
        echo "$variable" >>/etc/security/new_password_file
    else
        # No, so fill the file with user entry
        echo "        $variable $contains $var_string"
        ➤  >>/etc/security/new_password_file
    fi
done </etc/security/passwd

if [ "$user_processed" = "yes" ]
then
    # Copy the old password file to opasswd to conform with AIX
    # standards
    cp /etc/security/passwd /etc/security/opasswd
    # Replace the complete password file by the one generated to
    # minimize login errors
    mv /etc/security/new_password_file /etc/security/passwd
    logmsg $0 "Password changed for user $user"
    # Fill the result file
    echo "The password of $user is changed in `hostname`"
    ➤  >/tmp/dist_passwd.return
else
    # User not found: give message
    logmsg $0 "User-id $user doesn't exist in /etc/passwd: password
```

```
     ➤  NOT changed!"
     echo "User-id NOT in /etc/passwd: the password is NOT changed in
     ➤  `hostname`!"  >/tmp/dist_passwd.return
     rm /etc/security/new_password_file
fi

# remove input
rm /tmp/dist_passwd
```

The file */home/data/all_aix_servers* contains a list of all AIX servers. In our example, it consists of the following lines:

```
aftopt.amf.schuitema
afts1.amf.schuitema
afts2.amf.schuitema
aftvi1.amf.schuitema
wdns7.wdn.schuitema
```

EXAMPLES PASSWORD ENTRIES

Before changing the password:

```
root:
        password = hDoRLucOQfYNA
        lastupdate = 922092027
        flags =

user001:
        password = *

user002:
        password = 8H3GmteZWdtCA
        lastupdate = 810296852
        flags = NOCHECK,ADMCHG,ADMIN
```

After changing the password:

```
root:
        password = wmzE7uRQGbJPM
        lastupdate = 923549108
        flags =

user001:
        password = vWFJtM1hQPXGs
        lastupdate = 922974551
        flags =

user002:
        password = IdXtyTl2oNyMI
```

```
        lastupdate = 921592532
        flags = NOCHECK,ADMIN
```

Screen output of the script:

```
Time       Output commands
17:02:12 - Distributing password entries to aftopt.amf.schuitema
17:02:17 - The password will be changed using the remote command
           passwd_chg
17:02:22 - Rexec command to aftopt.amf.schuitema went wrong
17:02:22 - Password NOT changed!

17:02:22 - Distributing password entries to afts2.amf.schuitema
17:02:23 - FTP to afts2.aft.schuitema went wrong
17:02:23 - Password is NOT changed in afts2.amf.schuitema!

17:02:23 - Distributing password entries to aftvi1.amf.schuitema
17:02:24 - The password will be changed using the remote command
           passwd_chg
17:02:26 - Getting the result of the remote command
17:02:27 - Userid NOT in /etc/passwd: the password is NOT changed in
           aftvi1!

17:02:27 - Distributing password entries to wdns7.wdn.schuitema
17:02:28 - The password will be changed using the remote command
           passwd_chg
17:02:31 - Getting the result of the remote command
17:02:32 - The password of pos9909 is changed in wdns7

Rexec command to aftopt.amf.schuitema failed
Returncode=1
Information follows:
Login incorrect.
Password NOT changed!

Rexec command to aftopt.amf.schuitema failed
Returncode=1
Information follows:
Login incorrect.
Password NOT changed!

FTP to afts2.amf.schuitema failed
Information follows:
ftp: Unknown host afts2.amf.schuitema
?Invalid command
?Invalid command
?Invalid command
.
.
.
```

```
?Invalid command
?Invalid command
?Invalid command
?Invalid command
Remote command has not been executed
Password is NOT changed!

(EOF):

Statistics:
Number of sites processed         : 4
Number of ftps ok                 : 3
Number of wrong ftps              : 1
Number of rexecs ok               : 2
Number of wrong rexecs            : 1
Number of passwords changed       : 1
Number of passwords not changed   : 3
Average processing time / site : 5 seconds

$

Output in the logging system of 2 sites:
07/06/99 17:02:30 17552 root     /home/oper/passwd_chg ------- Password
changed for user pos9900

07/06/99 17:02:25 23578 root     /home/oper/passwd_chg ------- Userid
pos9900 doesn't exist in /etc/passwd: password NOT changed!
```

Teun Post
Unix/Oracle Specialist
Schuitema NV (The Netherlands)

Using the bc programmable calculator

The **bc** utility is a programmable calculator. It performs several types of calculation and also provides simple looping logic. It also benefits from being much easier to use than the **expr** expression evaluator used in shell scripting.

Start **bc** by typing the command **bc** from a command line. Once **bc** starts, you're in the context of the **bc** calculator until you type *quit*. Below is a simple example **bc** session. In the listing, the expression

‘4+2’ is entered by the user, to which **bc** responds by outputting ‘6’. Finally the user types *quit* and presses *Return* to exit **bc**.

```
$ bc
4+2
6
quit
$
```

Variables in **bc** can be used to hold values. Some versions of **bc** allow only single-character variables, though later versions allow multi-character variable names. In all the examples in this article, I use single-character variables so as to maintain compatibility with earlier versions. In the example below, the user makes the following assignments: *a=4*, *b=5*, and *c=b-a*. Notice that the assignments do not result in any output from **bc**. Finally the user types *c* on a line by itself and **bc** outputs ‘1’, the result stored in *c* after the assignment *c=b-a*. The **bc** calculator displays the result of any calculation that is not assigned to a variable. Typing an expression or a variable without an assignment causes the result to be output to the screen.

```
$ bc
a=4
b=5
c=b-a
c
1
quit
$
```

A special temporary variable in **bc** holds the last number output. This variable is ‘dot’ (‘.’). In the next example, the user enters *4+2* and **bc** outputs *6*. The user then enters ‘.’ and **bc** again outputs *6*, the value held in the temporary variable. The variable itself can be used in an expression, as in *.+1*, to which **bc** outputs *7*. Typing ‘.’ will again output *7*, as this is now the last value output and thus has been assigned to ‘.’. Finally the user enters an assignment, *c=4*, and then enters ‘.’ again. The output from **bc** is, again, *7*. It is important to remember that ‘.’ represents the last number output, not the last variable assignment.

```
$ bc
4+2
6
.
.+1
```

```
7
.
7
c=4
.
7
quit
$
```

The number of decimal places printed by **bc** is controlled by the variable *scale*. Predefined internal **bc** variables, such as *scale*, violate the one-character variable name rule even in early versions of **bc**. The default value of *scale* is 0. As shown below, the result of dividing 7 by 16 is displayed using different values of *scale*, starting with *scale=5* and ending with *scale=0*. As can be seen, **bc** truncates values rather than rounding them.

```
$ bc
scale=5
7/16
0.43750
scale=3
7/16
0.437
scale=2
7/16
0.43
scale=0
7/16
0
quit
$
```

Some standard **bc** operators are demonstrated in the next listing. Addition (‘+’), subtraction (‘-’), multiplication (‘*’), and division (‘/’) are fairly standard. Further down the list are remainder division (‘%’) and exponentiation (‘^’).

In remainder division, the result of the calculation is the remainder of the division rather than the quotient, hence:

- 5%2 is 1
- 15%13 is 2
- 6%3 is 0

- 27%5 is 2.

The exponentiation operator raises the left-hand operand to the power of the right-hand one, so:

- 3^2 is 9
- 2^3 is 8.

The **bc** assignment operator assigns a value to a variable. They are similar to the assignment and compound assignment operators in C. The first two examples in the listing below are simple assignment (‘=’). The compound plus-equal operator(‘+=’) adds the value on the right-hand side of the assignment to the variable on the left-hand side and stores the result in the variable on the left-hand side of the assignment. Thus *a+=2* is shorthand for *a=a+2*. The remaining compound assignment operators (‘-=’, ‘*=’, ‘/=’, ‘^=’, ‘%=’) all behave in analogous ways. When reading the listing below remember that the effect of the compound operator on the variable *a* is to alter its value.

```
$ bc
a = 7
b = 3
a += 2
a
9
a -= 4
a
5
a *= 2
a
10
a /= b
a
3
a ^= 3
a
27
a %= 5
a
2
quit
$
```

The **bc** increment and decrement operators are also similar to C,

though they have a side-effect in **bc**. Normally an increment or decrement operator simply adds or subtracts '1' from a variable, as in *x*++, or --*y*. Using an increment or decrement operator in **bc** also causes the value of the variable to be echoed to the screen. In the case of pre-increment or pre-decrement (++*x* and --*x*), the variable is changed and then its value echoed to the screen. In the case of post increment and decrement operators (*x*++ and *x*--), the value of the variable is echoed to the screen and then changed. Examples of these effects are shown below.

```
$ bc
x=1
x++
1
x
2
++x
3
x
3
x--
3
x
2
--x
1
x
1
quit
$
```

Output to the screen for increment or decrement operators can be suppressed by using the *void* keyword, as in:

```
$ bc
x=1
void x++
x
2
void ++x
x
3
quit
$
```

It seems to me that an assignment operator is more efficient than an increment or decrement operator in cases where you don't want to

display the result. Typing *x+=1* or *x-=1* requires fewer keystrokes than *void x++* or *void x--*.

Comparison operators are used to construct *if* structures. The available operators are: 'equal to' ('=='), 'less than' ('<'), 'greater than' ('>'), 'greater than or equal to' ('>='), 'less than or equal to' ('<='), and 'not equal' ('!='). The entire test should be enclosed in parentheses, and parentheses can be used to group tests. Comparison tests can be combined using 'and' ('&&') and 'or' ('||'), as in:

```
if ( ( ( x < 5 ) && ( y != 7 ) ) || ( z=4 ) )
```

More examples of *if* tests are provided later in this article.

There are two types of flow control loop in **bc**: *for* loops and *while* loops. A *for* loop takes the form:

```
for(initialization; test ; end of loop expression){
    statements;
}
```

In the next example, *x=0* sets *x*'s initial value to zero, *x<5* controls how long the *for* loop executes, and *++x* indicates what to do at the end of each loop. Everything within the curly braces is executed. In this case, the value of *x* is displayed on screen, resulting in a display of '0' to '4'. Note that a statement may be terminated either with a semicolon (';') or a newline character (the example uses a semicolon). Also note that the increment operator within a *for* loop does not cause the variable to be displayed. The behaviour of increment and decrement operators is slightly inconsistent in **bc**, and it is necessary to display the value of *x* explicitly below.

```
$ bc
for(x=0;x<5;++x){
        x;
}
0
1
2
3
4
quit
$
```

A *while* loop is another control loop available in **bc**. Its format is:

```
while(condition){
    statements;
}
```

The loop in the example below produces the same result as the preceding *for* loop. This time the loop takes advantage of the display and increment behaviour of *x*++ to display the value and increment it.

```
$ bc
x=0
while(x<5){
        x++;
}
0
1
2
3
4
quit
$
```

Just a quick warning for experienced C programmers: some versions of **bc** insist that the opening curly brace that starts the body of a *for*, *while*, or *if* block appears on the same line as the command. This is also true of function definitions, which I will cover in a moment. Hence the next example won't work on some versions of **bc** because the opening curly brace starts on a new line.

```
$ bc
x=0
while(x<10)
{
    x++;
}
quit
$
```

Text can be output to the screen by surrounding it with quotes, as in the following example. When you are using **bc** interactively at a terminal this is not much use, though it will become useful in writing **bc** programs. Output text is not appended with a new line, making it possible to display information as in:

```
$ bc
x = 7
"The value of x is "; x;
```

```
The value of x is 7
quit
$
```

The variable '.' holds only the last *number* that was output, so printing text on the screen does not affect its value.

A function in **bc** is a block of named code that can be executed as a subroutine within **bc**. The block can be passed values on which it operates and, given that **bc** is a calculator program, it can return a numeric value. In the next example, the function *s()* is defined as taking a single value. The function performs a calculation and returns the square of the value. When the function is typed in to **bc**, as shown below, **bc** defines the function but does nothing with it. If your version of **bc** supports identifiers longer than one character, it would be better to name this function *square()*. The keyword *return* causes *x***x* to be returned by *s()*.

```
$ bc
define s(x){
    return(x * x);
}
```

To call the function, you must pass it a value. The listing below defines the function and then uses it to calculate the square of *3* and place it in *a*. Variable *a* is then printed. Next, the square of *8* is calculated and printed directly to the screen.

```
$ bc
define s(x){
    return(x * x);
}
a = s(3)
a
9
s(8)
64
quit
```

Once you type *quit* and exit **bc**, the function definition is lost.

The variable *x* in the function definition of *s(x)* is a temporary variable or place holder. What happens to this *x* does not affect any other variable named *x* in the **bc** program. In the next code example, the instance of *x* that's set to *1* outside the function *s()* is not affected by

what happens to the other instance of *x* inside the function definition, even though *s()* changes the value of its instance of *x* inside the function.

```
$ bc
define s(x){
    x*=x;
    return(x);
}
x=1
a = s(3)
a
9
s(8)
64
x
1
quit
```

Now you have all the parts that you need to create a **bc** program. The first example program uses the function *s()* developed in the earlier listing. Using **vi**, create the following script (excluding the line numbers, which are just for explanation). The first thing you'll notice is that comments (lines 1 and 6) use the same convention as C, and comprise any text between an opening slash-asterisk ('/*') and a closing asterisk-slash ('*/'). Comments can span multiple lines, and some versions of **bc** support the character '#' to indicate the start of a one-line comment. The semicolon at the end of line 3 is optional as **bc** treats either a semicolon or a newline as the end of a statement, though the ones in lines 8 and 10 are not. Aside from the fancy text formatting, there are no surprises in the script. Save this script with a name such as *sqr*.

```
 1  /* s() returns the square of a number */
 2  define s(x){
 3      return(x*x);
 4  }
 5
 6  /* find the square of 2 and 8 */
 7  x=2
 8  "The area of a square with ";x;" ft sides is ";s(x);" sq ft";
 9  x=8
10  "The area of a square with ";x;" ft sides is ";s(x);" sq ft";
11
12  quit
```

Execute the script by typing **bc sqr**, as shown below. If **bc** is followed by a file name, the file is used as the input to **bc**.

```
$ bc sqr
The area of a square with 2
 ft sides is 4
 sq ft
The area of a square with 8
 ft sides is 64
 sq ft
```

This example is fine if you want to know the squares of two and eight, though what is needed is something more flexible, something that takes an argument and calculates a square from it.

The first step is to use something called a 'here' document. In the example below, *sqr* is converted into a here document. A line has been added at the top of the program that reads *bc <<END-OF-INPUT* and one at the bottom that reads *END-OF-INPUT*. **sqr** is no longer a **bc** program, it's now a shell script, and you must change its mode to allow execution using **chmod a+x sqr**. The instruction in line 1 reads: 'run **bc** and use the rest of this file as input to **bc** until you encounter a line containing *END-OF-INPUT*.' When **sqr** is run, the shell first starts **bc**, then reads line 2 and subsequent lines of the script, feeding them to **bc**. When line 15 is fed to **bc**, **bc** stops running as the line contains *quit*. The shell reads line 16 and recognizes the *END-OF-INPUT* tag that it is expecting and stops feeding lines to **bc** (which is fortunate, as **bc** has stopped running). So far, all we have is another method of showing the squares of two and eight, so not much has changed.

```
 1  bc <<END-OF-INPUT
 2  /* s() returns the square of a number */
 3  define s(x){
 4      return(x*x);
 5  }
 6
 7  /* use s on 2 and 8 */
 8  x=2
 9  "The area of a square with ";x;" ft sides is ";s(x);" sq ft";
10  x=8
11  "The area of a square with ";x;" ft sides is ";s(x);" sq ft";
12
15  quit
16  END-OF-INPUT
```

In the next example, **sqr** has been turned into a script that accepts input arguments and processes them correctly. In lines 3 to 7, the number of arguments on the command line is tested. The program must be started using the command **sqr** followed by a number. If there is no argument or more than one argument, an error results. Note that the error checking does not test whether the argument passed really is a number, it merely tests that a single argument exists, though you could improve its error handling by implementing code that validates that the argument is a number. If you invoke **sqr** with a non-numeric argument, as in:

```
sqr hello
```

bc either treats the argument as zero and returns 0 or displays an error.

The call to start **bc** with a here document begins at line 9, and is the same as discussed previously. Line 16 assigns the argument from the command line to *x*, and then the function *s()* is called.

```
 1  # sqr - takes a numeric argument and outputs its square
 2
 3  if [ $# != 1 ]
 4  then
 5      echo "A number argument is required"
 6      exit
 7  fi
 8
 9  bc <<END-OF-INPUT
10  /* s() returns the square of a number */
11  define s(x){
12      return(x*x);
13  }
14
15  /* use s on the argument from the command line */
16  x=$1
17  "The area of a square with ";x;" ft sides is ";s(x);" sq ft";
18
19  quit
20  END-OF-INPUT
```

The listing below illustrates the use of **sqr**.

```
$ sqr 9
The area of a square with 9
 ft sides is 81
$ sqr 15
The area of a square with 15
```

```
 ft sides is 225
 sq ft
$
```

addgrg gives **bc** a real workout. It is a procedure to add a number of days to a date and is invoked with two arguments, a date in *yyyymmdd* format and a number of days (positive or negative) to be added to the date. The listing of **addgrg** below follows the old **bc** convention of using only single-character names for variables and functions, so functions have cryptic names like as *y()* and *j()*. If your version of **bc** supports longer names, use ones like *days_in_year()* or *cvt_grg_to_jul()*. The script starts with a test to ensure that two arguments exist on the command line. Again, I check only for the existence of two arguments, and don't verify their correctness. Most of the **bc** script defines functions, and the main part of the program, where the arguments are used, doesn't begin until line 101. There are some additional features of a **bc** program that you will see here. Consider the function *y()* defined in lines 12 to 27. The first oddity is the existence of *y* as the name of a function and a variable. The designer of **bc** realized the limitation of one-character names and made sure that, although two functions cannot have the same name, a function can have the same name as a variable. Without this feature, **bc** would run out of names quickly.

Another new feature appears in line 13. The variable *x* is declared as *auto*. By default, all variables in **bc** except placeholders (described earlier) are global and available to all parts of the program. The *auto* declaration causes a temporary variable named *x* to be created that is valid only in the context of the function in which it is declared. In *y()*, *x* is created, used, then thrown away when *y()* returns. If a **bc** program encounters a global variable and an *auto* variable with the same name, the *auto* variable is used, if available. In this example, the existence of a global variable named *x* does not affect the use of a local version of *x* inside the function *y()*.

The third new feature appears in lines 15 to 17. While I mentioned *if* tests earlier, this is the first example of their syntax. Everything within the curly braces is executed if *x==0* is true.

Briefly, *y()* in lines 11 to 27 checks whether a year is a leap year. The function returns the number of days in the given year (365 or 366).

The rules for a leap year are (both must be satisfied):

1 A leap year is evenly divisible by 4.

2 A leap year is not evenly divisible by 100 unless it is also evenly divisible by 400.

Testing these rules is simplified by turning them upside down and testing for evenly divisible by 400, then by 100, then by 4, and taking the result as soon as you have a match.

The function *m()* in lines 28 to 41 receives a year and a month as input and uses the '30 days hath September…' rule to return 30 or 31. If the month is February, *y()* is called to find the number of days in the year. The return value of *y()* is tested to allow *m()* to return 29 or 28.

The function *g()* (lines 42 to 55) converts a date in Julian format to Gregorian format. Julian format is *yyyyddd* where *ddd* is the ordinal day of the year. For example Jan 1, 1999 is *1999001*, February 2, 1998 is *1998033* and December 31, 2000 is *2000366* (yes, 2000 is a leap year). The date in Julian format is converted to a Gregorian format, *yyyymmdd*, by calculating the days in each month and reducing the ordinal number of days of the date until the month is found.

The function *j()* (lines 56 to 61) is the inverse of *g()*. It converts a date in Gregorian format to Julian format by adding the days in all preceding months in the year to the day of the month to obtain the *ddd* portion of the date.

The function *b()* (lines 70 to 90) should have had a better name. It adds positive or negative days to a Julian format date. The reason that *yyyymmdd* dates are converted to *yyyyddd* format is that it's much easier to perform arithmetic using the latter format, which just requires adding the days and then checking whether you need to wrap forward or backward to another year.

The logic in *b()* probably requires some explanation. First, the days are added. After addition, the number of days value will be in one of three states. The number could be greater than the number of days in the year in question, indicating that the addition has thrown the date forward to a later year. Alternatively, it could be a value less than 1, indicating that a negative value was added and has thrown the date

back to an earlier year. Finally the days could fall within the year. If the number of days falls within the year, then no further processing is needed and the logic in lines 77 to 81 and 82 to 86 is skipped. These two pieces of logic handle the first two states respectively.

Working out the new year is just a matter of repeated subtraction (or addition) of the number of days in future (or past) years; this process is repeated until the number of calculated days is reduced to a value between 1 and 365 (or 366).

Function *a()* (lines 91 to 98) is the main function of the program. *a()* is passed a date in Gregorian format and a number of days (in fact, it's just passed the same arguments that are on the command line for the **addgrg** script). *a()* calls *j()* to convert the date to Julian format, then calls *b()* to add the days to the date in Julian format, and calls *g()* to convert the result back to Gregorian format.

Line 101, the main logic of the program, calls function *a()*, passing it the two arguments it was passed on the command line.

The listing below is available for download from *AIX Update*'s Web site (*http://www.xephon.com/aixupdate.html*). However, note that you need to strip out line numbers for the utility to work.

ADDGRG

```
 1    # addgrg yyyymmdd days - adds 'days' to yyyymmdd
 2
 3    if [ $# != 2 ]
 4    then
 5      echo "Not enough arguments"
 6      echo "Syntax addgrg yyyymmdd [-]days"
 7      exit
 8    fi
 9
10    bc <<END-OF-INPUT
11    /* return days in a four digit year */
12    define y(y){
13      auto x;
14      x = y % 400;
15      if (x == 0) {
16        return(366);
17        }
18      x = y % 100;
19      if ( x == 0){
```

```
20        return(365);
21        }
22      x = y % 4;
23      if ( x == 0){
24        return(366);
25        }
26      return (365);
27    }
28    /* return days in a month when yyyy and month are passed */
29    define m(y,m){
30      auto x,d;
31      if (m==1||m==3||m==5||m==7||m==8||m==10||m==12){
32        return(31);                                    /* 31 days */
33        }
34      if (m==4||m==6||m==9||m==11) {                  /* 30 days */
35        return(30);
36        }
37      if(y(y)==366){                          /* except February */
38        return(29);
39        }
40      return(28);
41    }
42    /* convert julian (yyyyddd) to gregorian (yyyymmdd)*/
43    define g(j){
44      auto y,d,m,n,g;
45      y = j/1000;                             /* extract the year */
46      d = j % 1000;                    /* extract the ddd portion */
47      for (m=1 ; d > 0 ; ++m){                 /* for each month */
48        n = m(y,m);            /* subtract the days of the month */
49        d -= n;                   /* until days are zero or less */
50        }
51      void --m;                              /* back up one month */
52      d+=n;                  /* add back the last days subtracted */
53      g= (y*10000)+(m*100)+d;    /* build a yyyymmdd format date */
54      return(g);
55    }
56    /* convert gregorian (yyyymmdd) to julian(yyyyddd)*/
57    define j(g){
58      auto y,m,d,x,i,j;
59      y=g/10000;                                   /* extract year */
60      m=(g/100) % 100;                                /* month */
61      d=g%100;                                       /* and day */
62      for(x=1;x<m;++x){               /* for each previous month */
63        i=m(y,x);                       /* add days of the month */
64        d+=i;                        /* add to days in this month */
65        }
66      j= (y*1000)+d;                     /* build yyyyddd format */
67      return(j);
68
69    }
```

```
70    /* add positive or negative days to julian date */
71    define b(j,a) {
72      auto y,d,r;
73      y=j/1000;
74      d=j%1000;
75      d+=a;
76      i=y(y);
77      while(d>i){    /* while calculated days > days in the year */
78        y+=1;                              /* increment the year */
79        d-=i;                        /* subtract days in the year */
80        i=y(y);                      /* and try days in this year */
81        }
82      while(d<1){                    /* while calculated days < 1 */
83        y-=1;                                 /* go back 1 year */
84        i=y(y);                       /* get the days in the year */
85        d+=i;        /* add to the calculated days and try again */
86        }
87      r= (y*1000)+d;
88      return(r);
89
90    }
91    /* add positive or negative days to a greg date */
92    define a(g,a){
93      auto j,r;
94      j=j(g);                         /* convert to julian format */
95      j=b(j,a);                      /* add days to julian format */
96      r=g(j)                 /* convert back to gregorian format */
97      return(r);
98    }
99
100    /* call add to greg using the two command line arguments */
101    a($1,$2)
102
103    quit
104    END-OF-INPUT
```

The **bc** script is very legible compared with an equivalent shell script using **expr** and all the extra characters that **expr** requires. Below is a sample output from a series of test runs on **addgrg** that focus on leap years and boundary conditions.

```
$ addgrg 19980101 -1
19971231
addgrg 20000101 60
20000301
addgrg 20000101 59
20000229
addgrg 19000101 365
19010101
```

```
addgrg 20001231 -365
20000101
$
```

If you do a lot of numeric work that's programmable, learning to program **bc** scripts should make calculations much simpler.

A final note on dates: both the date formats used in **addgrg** are in the Gregorian calendar. The *yyyyddd* format should really be called the ordinal day date format, though the convention of calling it the 'Julian date' or 'Julian format date' has stuck. It is, however, neither a date in the Julian calendar nor one in the period of the Julian calendar.

Mo Budlong (USA)

SSCCARS (part 3)

This month's instalment continues this series of articles on utilities that comprise a source code management system.

CHKOUT.SH

chkout.sh is the check out module. It has an interface to an Oracle database called 'SMART' and uses the *UpdateSmart* library function to store the checkout details of the updatable copy of the source.

CHKOUT.SH

```
#! /bin/ksh
################################################################################
# Name     : chkout.sh
#
# Overview : Allows the user to check out:
#            o A read-only copy of the latest version of a source
#            o A read-only copy of a specific version of a source
#            o An updateable copy of a source.
#
#            It also allows the authorized user to cancel
#            any booking of the modifiable copy of the source.
#
```



```
# Notes     1 Requests for read-only copies of a source are
#               always met.
#
#           2 Requests for updateable copies of a source are
#               met only if the source is not already checked out.
#
#           3 If a copy of the required source exists in the
#               target directory, the program displays an error
#               message.
#
#           4 If the target directory is not writeable by
#               user, the program displays an error message.
#
#           5 The required source must have a pre-defined file
#               extension.
#
#           6 The program always copies the source with the
#               highest version in response to a request for a
#               read-only copy; otherwise the copy of the
#               source has the highest version plus one.
#
#           7 A pre-requisite of this program is that all
#               existing source code has a version number.
#
#           8 Only the authorized user (root) may cancel
#               a booking of a modifiable copy of a source.
#
#           9 When a copy of a source is made for update,
#               it locks the source by calling execsu with the
#               ACTION variable set to 'CR'. This creates and
#               maintains a zero-byte file of the same name in
#               directory $LOCK_DIR. When a request to cancel a
#               booking is made, the zero-byte file is removed,
#               thus freeing the source. This is done by calling
#               execsu with the ACTION variable set to 'RM'.
#
#          10 Users are not allowed to copy source files to any
#               of the restricted directories listed in
#               $RESTRICTED_DIR_LIST.
#
#          11 All the allowed file extensions are held in
#               the variable FILE_EXTS.
#
#          12 If a new file extension is added, a subdirectory
#               with the same name must be created in the
#               $SOURCE_DIR directory, otherwise the source file
#               cannot be found.
#
#          13 A log entry is written whenever a copy of the
#               source is made for update or an existing booking
```

```
#             for an update is cancelled.
#
#          14 The script contains following functions:
#             o main
#             o DisplayMenu
#             o ProcessExit
#             o ProcessMenuOption
#             o DefineModuleVariables
#             o ProcessReadOnlyLatestCopy
#             o ProcessReadOnlySpecificCopy
#             o ProcessUpdateableCopy
#             o CheckLocation
#             o GetSourceFileNameFromLov
#             o ProcessFileExtension.
################################################################################
################################################################################
# Name     : DisplayMenu
#
# Overview : The function displays menu
################################################################################
DisplayMenu ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
OPTION=
clear
echo ""
echo ""
echo "          ##############################################"
echo "          #        ${CHKOUT}                           #"
echo "          #                                            #"
echo "          #   5  Get latest copy (read-only)           #"
echo "          #  10  Get specific version (read-only)      #"
echo "          #  15  Get modifiable copy                   #"
echo "          #  25  Main menu                             #"
echo "          #  99  Exit                                  #"
echo "          #                                            #"
echo "          #  Please acknowledge message displayed      #"
echo "          ##############################################"
echo "                    Enter option--->\c"
read OPTION
}
################################################################################
# Name     : ProcessMenuOption
#
# Overview : Process a menu option
#
# Notes    1 The function calls the following functions:
#          o DisplayMenu
#          o ProcessReadOnlyLatestCopy
#          o ProcessReadOnlySpecificCopy
```

```
#          o ProcessUpdateableCopy
#          o CancelCheckedOutBooking
#############################################################################
ProcessMenuOption ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
while true
do
  DisplayMenu
  case "${OPTION}" in
     5) if  ProcessReadOnlyLatestCopy
        then
          DisplayMessage I "${READ_LATEST_CHECKOUT_DONE}" ;
        else
          if [ "${FUNCTION_ABORTED}" = "Y" ]
          then
            FUNCTION_ABORTED=N ;
            DisplayMessage I "${FUNCTION_ABORTING}" ;
          else
            DisplayMessage E "${READ_LATEST_CHECKOUT_FAILED}" ;
          fi ;
        fi;;
    10) if  ProcessReadOnlySpecificCopy
        then
          DisplayMessage I "${READ_SPECIFIC_CHECKOUT_DONE}" ;
        else
          if [ "${FUNCTION_ABORTED}" = "Y" ]
          then
            FUNCTION_ABORTED=N ;
            DisplayMessage I "${FUNCTION_ABORTING}" ;
          else
            DisplayMessage E "${READ_SPECIFIC_CHECKOUT_FAILED}" ;
          fi ;
        fi;;
    15) if  ProcessUpdateableCopy
        then
          DisplayMessage I "${UPDATE_CHECKOUT_DONE}" ;
        else
          if [ "${FUNCTION_ABORTED}" = "Y" ]
          then
            FUNCTION_ABORTED=N ;
            DisplayMessage I "${FUNCTION_ABORTING}" ;
          else
            DisplayMessage E "${UPDATE_CHECKOUT_FAILED}" ;
          fi ;
        fi ;;
    25) return $TRUE ;;
    99) ProcessExit $SEC ;;
    "") if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
        then
```

```
          FUNCTION_INTERRUPTED=N;
        else
          DisplayMessage E "${INVALID_ENTRY}" ;
        fi ;;
     *) DisplayMessage E "${INVALID_ENTRY}"
  esac
done
}
##############################################################################
# Name     : ProcessReadOnlyLatestCopy
#
# Overview : Makes a read-only copy of the latest version of the
#            requested source file.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 No log entry is written for this action.
#
#          2 The function calls following functions:
#            o GetSourceName
#            o CheckLocation
#            o ProcessFileExtension
#            o GetDirectoryName
#            o GetLatestVersion  READ
#            o CopySource.
##############################################################################
ProcessReadOnlyLatestCopy ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get source name
if ! GetSourceName "CO"
then
  return $FALSE
fi
# check current location
if ! CheckLocation
then
  return $FALSE
fi
# process file extension
if ! ProcessFileExtension "E"
then
  return $FALSE
fi
# get target directory name
if ! GetDirectoryName "CO"
then
  return $FALSE
fi
# get latest version number for the source
```

```
if ! GetLatestVersion READ
then
  return $FALSE
fi
# establish file details
REQ_DIR="${SOURCE_DIR}/${SOURCE_EXT}"
REQ_FILE="${REQ_DIR}/${REQ_SOURCE}"
TARGET_FILE="${TARGET_DIR}/${REQ_SOURCE}"
# check that the required file exists
if [ ! -f "${REQ_FILE}" ]
then
  FILE_NAME=${REQ_SOURCE}
  DIR_NAME=${REQ_DIR}
  DisplayMessage E "${NO_SOURCE_FOUND}"
  return $FALSE
fi
# check for required file in target directory
if [ -f "${TARGET_FILE}" ]
then
  SLEEP_DURATION=5
  FILE_NAME=${REQ_SOURCE}
  DIR_NAME=${TARGET_DIR}
  DisplayMessage E "${SOURCE_EXISTS}"
  return $FALSE
fi
# copy the source
if ! CopySource ${REQ_FILE} ${TARGET_FILE}
then
  return $FALSE
fi
}
##############################################################################
# Name     : ProcessReadOnlySpecificCopy
#
# Overview : This function makes a read-only copy of a specific
#            version of a given source.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 No log entry is written for this action.
#
#          2 The function calls following functions:
#            o GetSourceName
#            o CheckLocation
#            o ProcessFileExtension
#            o GetDirectoryName
#            o CopySource.
##############################################################################
ProcessReadOnlySpecificCopy ()
{
```

```
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get source name
if ! GetSourceName "CO"
then
  return $FALSE
fi
# check for user current location
if ! CheckLocation
then
  return $FALSE
fi
# process file extension
if ! ProcessFileExtension "E"
then
  return $FALSE
fi
# get source version number
if ! GetSourceVersionNumber
then
  return $FALSE
fi
# get target directory name
if ! GetDirectoryName "CO"
then
  return $FALSE
fi
# establish file details
REQ_SOURCE="${SOURCE_NAME_WITHOUT_EXT}_${VERSION_NUMBER}.${SOURCE_EXT}"
REQ_DIR="${SOURCE_DIR}/${SOURCE_EXT}"
REQ_FILE="${REQ_DIR}/${REQ_SOURCE}"
TARGET_FILE="${TARGET_DIR}/${REQ_SOURCE}"
# check for file existence for required file
if [ ! -f "${REQ_FILE}" ]
then
  FILE_NAME=${REQ_SOURCE}
  DIR_NAME=${REQ_DIR}
  DisplayMessage E "${NO_SOURCE_FOUND}"
  return $FALSE
fi
# check for required file in target directory
if [ -f "${TARGET_FILE}" ]
then
  FILE_NAME=${REQ_SOURCE}
  DIR_NAME=${REQ_DIR}
  DisplayMessage E "${SOURCE_EXISTS}"
  return $FALSE
fi
# copy the source
if ! CopySource "$REQ_FILE" "${TARGET_FILE}"
then
```



```
  return $FALSE
fi
return $TRUE
}
##############################################################################
# Name     : ProcessUpdateableCopy
#
# Overview : This function makes an updateable copy of the
#            requested source file.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 The function checks $LOCK_DIR to establish whether
#            the source is already checked out. If it is, an
#            error message is displayed.
#
#          2 A log entry is written when the source is successfully
#            copied.
#
#          3 The function calls following functions:
#            o GetSourceName
#            o CheckLocation
#            o ProcessFileExtension
#            o GetDirectoryName
#            o GetLatestVersion UPDATE
#            o CheckLock
#            o LockSource
#            o CopySource
#            o FreeLock.
##############################################################################
ProcessUpdateableCopy ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get source file name
if ! GetSourceName "CO"
then
  return $FALSE
fi
# check for user current location
if ! CheckLocation
then
  return $FALSE
fi
# process source file extension
if ! ProcessFileExtension "E"
then
  return $FALSE
fi
# get target directory
if ! GetDirectoryName "CO"
```

```
then
  return $FALSE
fi
# get latest version of the source
if ! GetLatestVersion UPDATE
then
  return $FALSE
fi
# check for lock on the requested source
if CheckLock "${TARGET_SOURCE}"
then
  # source is already locked
  DisplayMessage E "${SOURCE_ALREADY_LOCKED}"
  return $FALSE
fi
# establish file details
REQ_DIR="${SOURCE_DIR}/${SOURCE_EXT}"
REQ_FILE="${REQ_DIR}/${REQ_SOURCE}"
TARGET_FILE="${TARGET_DIR}/${TARGET_SOURCE}"
SOURCE_VERSION=`echo $TARGET_SOURCE |  sed  's/.*\_//' | cut -d'.' -f1`
# check for file existence for required file
if [ ! -f "${REQ_FILE}" ]
then
  DisplayMessage E "${NO_SOURCE_FOUND}"
  return $FALSE
fi
# check for required file in target directory
if [ -f "${TARGET_FILE}" ]
then
  DisplayMessage E "${SOURCE_EXISTS}"
  return $FALSE
fi
# check database status
if ! SmartDatabaseStatus
then
  DisplayMessage E "${DB_NOT_OK}"
  return $FALSE
fi
# lock the source
if ! LockSource ${TARGET_SOURCE}
then
  return $FALSE
fi
# copy the source
if ! CopySource "${REQ_FILE}" "${TARGET_FILE}"
then
  # free the lock
  FreeLock ${TARGET_SOURCE}
  return $FALSE
fi
```

```
# write log
${SSCCARS_BIN}/execsu "WL" ${LOG_FILE} ${LOG_DIR} ${LOG_DAY} \
  ${LOG_TIME} ${USERID} ${TARGET_SOURCE} "CHECKED OUT" > \
  ${TEMP_FILE2} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE2} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  # undo everything; remove copied source file
  rm -f "${TARGET_FILE}"
  # remove the lock
  FreeLock "${TARGET_SOURCE}"
  return $FALSE
fi
# update SMART database
if UpdateSmart "CO"
then
  DisplayMessage I "${SMART_UPDATED}"
else
  DisplayMessage E "${SMART_NOT_UPDATED}"
fi
return $TRUE
}
##############################################################################
# Name     : DefineModuleVariables
#
# Overview : Defines all the module's variables
##############################################################################
DefineModuleVariables ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# define error handling variables
PROG="chkout.sh"
# define ordinary variables
OPTION=      # selected menu option
REPLY=
MESSAGE=
ACTION=      # used to specify the type of action to be carried out
             # by execsu (ACTION = R to remove a file,
             #            ACTION = C to copy a file, etc)
# define variables required to obtain the latest version of the source
FILE_MATCH=
REQ_FILE_LIST=
FILE_COUNT=
FILE_NAME=
# define temporary files
TEMP_FILE1="${TEMP_DIR}/chkout_1_$$.dat"
TEMP_FILE2="${TEMP_DIR}/chkout_2_$$.dat"
}
##############################################################################
```

```
# Name     : ProcessExit
# Overview : Removes all temporary files and makes a graceful exit.
#
# Input    : exit code ($SEC or $FEC)
################################################################################
ProcessExit ()
{
rm -f ${TEMP_DIR}/*chkout*
EXIT_CODE="$1"
GLOBAL_EXIT=Y
exit $EXIT_CODE
}
################################################################################
# Name     : GetSourceVersionNumber
#
# Overview : Gets the required source version number from the user.
################################################################################
GetSourceVersionNumber ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
VERSION_NUMBER=""
while true
do
  clear
  echo "Enter required source version number (q to quit):\c"
  read VERSION_NUMBER
  case $VERSION_NUMBER in
    "") if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
        then
          FUNCTION_INTERRUPTED=N:
        else
          DisplayMessage E "${INVALID_ENTRY}" ;
        fi ;;
   q|Q) FUNCTION_ABORTED=Y ; return $FALSE ;;
     *) # check for numeric
        if ([ `expr $VERSION_NUMBER + 0` -eq $VERSION_NUMBER ]) \
             > /dev/null 2>&1
        then
          break ;
        else
          DisplayMessage E "${NOT_NUMERIC}" ;
        fi ;;
  esac
done
}
################################################################################
# Name     : main
#
# Overview : Calls all other functions.
################################################################################
```

```
main ()
{
DefineModuleVariables
ProcessMenuOption
}
# invoke main
main
```

CHKIN.SH

chkin.sh is the ‘check-in’ module. The module has an interface to an Oracle database called ‘SMART’. It uses the *UpdateSmart* library function to store the relevant check-in details in the database.

CHKIN.SH

```
#! /bin/ksh
###############################################################################
# Name     : chkin.sh
#
# Overview : Checks in a modified copy of the source code that
#            was checked out earlier or a copy of the new
#            source.
#
# Notes    1 The modified source code is copied from the current
#            directory if a target directory is not specified.
#
#          2 The required source code must have a pre-defined file
#            extension.
#
#          3 The program checks in the copy of the modified
#            source that was checked out earlier.
#
#          4 New source code that's checked in is given a
#            version number of 1, so abc.sql becomes abc_1.sql.
#
#          5 All the allowed file extensions are held in the
#            variable FILE_EXTS.
#
#          6 If a new file extension is added, a subdirectory
#            of that name must also exist in the directory
#            $SOURCE_DIR, else the source file won't be found.
#
#          7 A log entry is written whenever a copy of the
#            modified source or a new source is checked in.
#
#          8 Source to be checked in is moved and not copied.
#
```

```
#          9 The script contains following functions:
#            o DisplayCheckinMenuOptions
#            o ProcessCheckinMenuOptions
#            o CheckinNewSource
#            o GetSourceDetails
#            o CheckinExistingSource.
################################################################################
################################################################################
# Name     : SeekConfirmation
#
# Overview : Gets confirmation for a message in MESSAGE
################################################################################
SeekConfirmation ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
while true
do
  echo "$MESSAGE"
  read REPLY
  case $REPLY in
    y|Y) break ;;
    n|N) break;;
      *) echo "Invaid option";
         clear;;
  esac
done
}
################################################################################
# Name     : DefineModuleVariables
#
# Overview : Define all the module variables.
################################################################################
DefineModuleVariables ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# define error handling variables
PROG="chkin.sh"
TEMP_FILE="${TEMP_DIR}/chkin_$$.dat"
}
################################################################################
# Name     : DisplayCheckinMenuOptions
#
# Overview : Displays menu options.
################################################################################
DisplayCheckinMenuOptions ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
clear
echo ""
echo ""
```

```
echo "        ##############################################"
echo "        #        ${CHKIN}                            #"
echo "        #                                            #"
echo "        #    5. Check in existing source             #"
echo "        #   10. Check in new source                  #"
echo "        #   15. Main menu                            #"
echo "        #   99. Exit                                 #"
echo "        #                                            #"
echo "        # Please acknowledge the displayed message   #"
echo "        ##############################################"
echo "           Enter option --->\c"
read OPTION
}
#############################################################################
# Name     : ProcessCheckinMenuOption
#
# Overview : Processes selected menu option.
#############################################################################
ProcessCheckinMenuOption ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
while true
do
  clear
  OPTION=""
  DisplayCheckinMenuOptions
  case $OPTION in
    5) if CheckinExistingSource
       then
         DisplayMessage I "${CHECKED_IN}" ;
       else
         if [ "${FUNCTION_ABORTED}" = "Y" ]
         then
           FUNCTION_ABORTED=N ;
           DisplayMessage I "${FUNCTION_ABORTING}" ;
         else
           DisplayMessage E "${NOT_CHECKED_IN}" ;
         fi;
       fi;;
   10) if CheckinNewSource
       then
         DisplayMessage I "${CHECKED_IN}" ;
       else
         if [ "${FUNCTION_ABORTED}" = "Y" ]
         then
           FUNCTION_ABORTED=N ;
           DisplayMessage I "${FUNCTION_ABORTING}" ;
         else
           DisplayMessage E "${NOT_CHECKED_IN}" ;
         fi;
```

```
        fi;;
    15) rm -f ${TEMP_DIR}/*chkin* ;
        return $TRUE ;;
    99) GLOBAL_EXIT=Y;
        return $TRUE ;;
     *) if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
        then
          FUNCTION_INTERRUPTED=N ;
        else
          DisplayMessage E "${INVALID_ENTRY}";
        fi ;;
  esac
done
}
##############################################################################
# Name     : CheckinExistingSource
#
# Overview : Checks in an existing checked out source.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 Requires full source name (with the checked out
#            version number and file extension).
##############################################################################
CheckinExistingSource ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get source name
if ! GetSourceName "CI"
then
  return $FALSE
fi
# check user location
if ! CheckLocation
then
  return $FALSE
fi
# process file extension
if ! ProcessFileExtension "E"
then
  return $FALSE
fi
# get directory name
if ! GetDirectoryName "CI"
then
  return $FALSE
fi
DisplayMessage I "${WORKING}" N
# check whether checked out
if ! CheckLock "${SOURCE_NAME}"
```

```
then
  DisplayMessage E "${SOURCE_NOT_CHECKED_OUT}"
  return $FALSE
fi
# establish file details
TARGET_DIR="${SOURCE_DIR}/${SOURCE_EXT}"
TARGET_SOURCE="${SOURCE_NAME}"
TARGET_FILE="${TARGET_DIR}/${SOURCE_NAME}"
REQ_FILE="${REQ_DIR}/${SOURCE_NAME}"
SOURCE_VERSION=`echo ${SOURCE_NAME} | sed 's/.*\_//' | cut -d'.' -f1`
ExtractSourceName ${SOURCE_NAME}
# perform sanity check; check for target directory existence
if [ ! -d ${TARGET_DIR} ]
then
  DIR_NAME="${TARGET_DIR}"
  SLEEP_DURATION=5
  DisplayMessage E "${DIR_NOT_EXIST}"
  return $FALSE
fi
# check target file exists
if [ -f ${TARGET_FILE} ]
then
  FILE_NAME="${SOURCE_NAME}"
  DIR_NAME="${TARGET_DIR}"
  DisplayMessage E "${SOURCE_EXISTS}"
  return $FALSE
fi
# check required file exists
if [ ! -f ${REQ_FILE} ]
then
  FILE_NAME="${SOURCE_NAME}"
  DIR_NAME="${REQ_DIR}"
  DisplayMessage E "${NO_SOURCE_FOUND}"
  return $FALSE
fi
# check database status
if ! SmartDatabaseStatus
then
  DisplayMessage E "${DB_NOT_OK}"
  return $FALSE
fi
# check in the source
${SSCCARS_BIN}/execsu "CI" "${REQ_FILE}" "${TARGET_FILE}" > \
  ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  return $FALSE
fi
```

```
# free the lock
if ! FreeLock "${SOURCE_NAME}"
then
  DisplayMessage E "${LOCK_NOT_REMOVED}"
  RemoveCopiedSourceFile "${TARGET_SOURCE}" "${TARGET_DIR}"
  LockSource ${SOURCE_NAME}
  return $FALSE
fi
# update the log
${SSCCARS_BIN}/execsu "UL" "${LOG_FILE}" "${LOG_DIR}" "${LOG_DAY}" \
  "${LOG_TIME}" "${USERID}" "${SOURCE_NAME}" "CHECKED IN" > \
  ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  LockSource "${SOURCE_NAME}"
  RemoveCopiedSourceFile "${TARGET_SOURCE}" "${TARGET_DIR}"
  return $FALSE
fi
# update SMART database
if UpdateSmart "CIE"
then
  DisplayMessage I "${SMART_UPDATED}"
else
  DisplayMessage E "${SMART_NOT_UPDATED}"
fi
return $TRUE
}
##############################################################################
# Name     : CheckinNewSource
#
# Overview : The function checks in a new source.
#
# Notes    1 Requires full source name with the checked out version
#            number and file extension.
##############################################################################
CheckinNewSource ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get source name
if ! GetSourceName "CIN"
then
  return $FALSE
fi
# check user location
if ! CheckLocation
then
  return $FALSE
fi
```

```
# process file extension
if ! ProcessFileExtension "E"
then
  return $FALSE
fi
# get directory name
if ! GetDirectoryName "CI"
then
  return $FALSE
fi
# get source details
if ! GetSourceDetails
then
  return $FALSE
fi
DisplayMessage I "${WORKING}" N
# establish file details
TARGET_DIR="${SOURCE_DIR}/${SOURCE_EXT}"
TARGET_SOURCE="${SOURCE_NAME_WITHOUT_EXT}_1.${SOURCE_EXT}"
TARGET_FILE="${TARGET_DIR}/${TARGET_SOURCE}"
REQ_FILE="${REQ_DIR}/${SOURCE_NAME}"
# perform sanity check; check target directory exists
if [ ! -d ${TARGET_DIR} ]
then
  DIR_NAME="${TARGET_DIR}"
  SLEEP_DURATION=5
  DisplayMessage E "${DIR_NOT_EXIST}"
  return $FALSE
fi
# check required file exists
if [ ! -f ${REQ_FILE} ]
then
  SLEEP_DURATION=5
  FILE_NAME="${SOURCE_NAME}"
  DIR_NAME="${REQ_DIR}"
  DisplayMessage E "${NO_SOURCE_FOUND}"
  return $FALSE
fi
# check target file exists
if [ -f ${TARGET_FILE} ]
then
  SLEEP_DURATION=5
  FILE_NAME="${TARGET_SOURCE}"
  DIR_NAME="${TARGET_DIR}"
  DisplayMessage E "${SOURCE_EXISTS}"
  return $FALSE
fi
# check database status
if ! SmartDatabaseStatus
then
```

```
  DisplayMessage E "${DB_NOT_OK}"
  return $FALSE
fi
# check in the new source
${SSCCARS_BIN}/execsu "CI" "${REQ_FILE}" "${TARGET_FILE}" > \
  ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  return $FALSE
fi
# write log
${SSCCARS_BIN}/execsu "WL" ${LOG_FILE} ${LOG_DIR} ${LOG_DAY} \
  ${LOG_TIME} ${USERID} ${SOURCE_NAME} "CHECKED IN NEW SOURCE" > \
  ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  RemoveCopiedSourceFile "${TARGET_SOURCE}" "${TARGET_DIR}"
  return $FALSE
fi
# register source
${SSCCARS_BIN}/execsu "RS" "${DATA_FILE}" "${LOG_DIR}" \
  "${SOURCE_NAME}" "${SOURCE_FULL_NAME}" "${SOURCE_DESCRIPTION}" > \
  ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat ${TEMP_FILE} | head -1`
  DisplayMessage E "${EXECSU_ERROR}"
  RemoveCopiedSourceFile "${TARGET_SOURCE}" "${TARGET_DIR}"
  return $FALSE
fi
# update SMART database
if UpdateSmart "CIN"
then
  DisplayMessage I "${SMART_UPDATED}"
else
  DisplayMessage E "${SMART_NOT_UPDATED}"
fi
return $TRUE
}
##############################################################################
# Name     : GetSourceDetails
#
# Overview : Gets the full source name and description of the
#            new source being checked in by the user.
#
# Notes    1 The full source name may have no more than 30
```

```
#            characters.
#
#          2 The source description may have no more than
#            2000 characters.
################################################################################
GetSourceDetails ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get full name of source
while true
do
  clear
  echo "Enter the full source name (trigger or procedure name),"
  echo "which may have up to 30 charters (q to quit):\c"
  read SOURCE_FULL_NAME
  case ${SOURCE_FULL_NAME} in
   "") if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
       then
         FUNCTION_INTERRUPTED=N ;
       else
         DisplayMessage E "${INVALID_ENTRY}" ;
       fi ;;
  q|Q) FUNCTION_ABORTED=Y ; return $FALSE ;;
    *) SOURCE_FULL_NAME="`echo $SOURCE_FULL_NAME | cut -c1-30`" ;
       break ;;
  esac
done
# get source description
while true
do
  clear
  echo "Enter a short (up to 2000 characters) description"
  echo "for the source (q to quit):\c"
  read SOURCE_DESCRIPTION
  case "$SOURCE_DESCRIPTION" in
   "") if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
       then
         FUNCTION_INTERRUPTED=N ;
       else
         DisplayMessage E "${INVALID_ENTRY}" ;
       fi ;;
  q|Q) FUNCTION_INTERRUPTED="Y" ; return $FALSE ;;
    *) # accept only first 2000 characters
       SOURCE_DESCRIPTION="`echo $SOURCE_DESCRIPTION | \
         cut -c 1-2000`" ;
       break ;;
  esac
done
}
################################################################################
```

```
# Name     : main
#
# Overview : Calls all other functions.
##############################################################################
main ()
{
DefineModuleVariables
ProcessCheckinMenuOption
}
# invoke main
main
```

There are two more instalments of this article, which concludes in the March issue.

Arif Zama
DBA/Administrator
High-Tech Software (UK)

Script locking mechanism

Two or more instances of a script can execute at the same time if more than one instance of **cron** is running, or if the interval at which the script is being executed is too small, or simply as a result of user errors. The side-effects of running a script multiple times can, of course, be disastrous.

To prevent this happening, one must create a locking mechanism. Most locking mechanisms rely on a temporary file, called a lockfile. If the lockfile exists, the program acts accordingly.

The code below is a little more sophisticated – it creates a lockfile containing the process id. The next time your script is started, the locking code checks whether the lockfile exists. If it does, the code then checks whether the process id in the lockfile is still valid. If a valid process id is detected, the code exits. On the other hand, if the process id isn't valid, the code allows your script to continue execution.

As the lockfile is recreated (when necessary) by the script, your script can be restarted (say, after a **kill** command, etc), without the need to remove the lockfile manually.

To use this code, simply copy it and paste it in the start section of your favourite scripts.

SCRIPT LOCKING CODE

```
# START OF LOCKING MECHANISM

myname=$(basename $0)
lockfile=/tmp/.lock.${myname}

# prevent script from running multiple times simultaneously.
if [[ -f ${lockfile} ]]; then
   pid=$(cat ${lockfile})
   # Check if pid still there
   npid=$(ps -opid= -p ${pid})
   if [[ ${npid} != "" ]]; then
      # Check arguments of command running
      ps -oargs -p ${npid} | grep -i ${myname} > /dev/null
      rc=$?
      if (( ${rc} == 0 )); then
         echo $0" : script is still running"
         exit 1
      fi
   fi
fi
# create new lock file
echo $$ > ${lockfile}

# END OF LOCKING MECHANISM
```

AIX news

IBM has announced C for AIX Version 5.0, which offers improved portability through support for the OpenMP industry specification in 32-bit and 64-bit versions, easier debugging with a new graphical source debugger, enhanced SMP support through automatic and explicit parallelism, support for ANSI C and Unicode, and profile-directed feedback and inter-procedural analysis to optimize performance of C applications.

The new compiler supports run-time dynamic linking provided by AIX Version 4.2.1 or later, and also provides improved prototyping and cross-platform compatibility with VisualAge products. Other benefits include a 64-bit integer data type, a 128-bit floating-point data type, and run-time address checking.

Available now, prices start at US$780 for new users and US$340 for upgrades.

The company also launched the Power GXT300P Graphics Accelerator for IBM RS/6000 systems. It's a short PCI adapter card requiring one PCI slot. Available now, prices are available on request from IBM.

For further information contact your local IBM representative.

* * *

Computer Associates has released eTrust Access Control 5.0 for Unix, which is an uprated version of CA's ACX/ACWNT (SeOS). The software enables Unix users to protect corporate data and applications with security policies that prohibit unauthorized system access. Version 5.0 enables complete access control from a single point via a graphical user interface. It has profile groups, which allow security to be based on role or group membership.

The product now supports AIX 4.3.2 and other 64-bit Unixes, including HP-UX 11.0, Solaris 7, and IRIX 6.5 (a version is also available for NT). Out now, prices are available on request from the vendor.

For further information contact:
Computer Associates International, 1 Computer Associates Plaza, Islandia, NY 11788, USA
Tel: +1 516 342 5224
Fax: +1 516 342 5734
Web: http://www.cai.com

Computer Associates Plc, Computer Associates House, 183-187 Bath Road, Slough, Berks SL1 4AA, UK
Tel: +44 1753 577733
Fax: +1 1753 825464

* * *

Following the US Government's decision to make the export of cryptographic products and solutions using encryption stronger than 56-bit DES outside the United States and Canada easier, IBM has announced that Host On-Demand, Personal Communications, and Communications Server for AIX can now include 128-bit and/or 168-bit Triple DES encryption support for Secure Sockets Layer (SSL). Customers in the finance, insurance, and healthcare sectores along with United States subsidiaries and on-line merchants are the main beneficiaries of this.