53 AIX update

March 2000

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.

Monitoring remote servers

Our environment includes many distributed AIX servers, and we have to monitor their status continuously. We want to ensure that all filesystems have enough free space, all scheduled tasks run successfully, and all processes are running properly. If a problem occurs, we want to know about it before a user calls us to report it. More importantly, we want to take the necessary action before problems arise.

For example: sometimes a filesytem fills to 90% or more. If this filesystem then grows by even a small amount, users may not be able to access the server. We should connect to this server and take necessary action before the filesystem becomes that full. In addition, we may want to make sure that DB2 or CICS processes are running correctly on the servers.

Something else that invariably needs monitoring is scheduled scripts. System programmers frequently configure a large number of scripts and programs to run in *crontab*. They want to know the names of any server on which the scripts did not complete successfully and the return codes of scripts that terminated with an error. Thus, if a back-up script is set in *crontab*, we want to know which servers failed to back up successfully.

Hence we wrote some programs that allow us to keep an eye on the status of our servers. The programs use TCP/IP sockets to communicate with servers. **alert_b** runs on every server and monitors some pre-defined thresholds continuously. If a threshold is exceeded, it sends an alert to the control server. **alert_s** is the component that runs on the control server. It receives the alerts and inserts them in a DB2 table. A service named *alert* must be added to */etc/services* file.

alert_b reads two 'parameter files'. The first, *alert_parms*, specifies the control server and the interval for monitoring thresholds in seconds.

The second, *alert.ctrl*, contains pre-defined thresholds. We can define as many thresholds as we want by including the three lines shown overleaf for each one.

```
df -k | grep actlogs | awk '{ print $4 } ' | sed 's/%//'
>
/u/pmkdbd2/alert.out > 80 ACTV
```

alert_s receives the server name, error code, and return code from servers in which pre-defined thresholds are exceeded. It inserts this data, along with the timestamp when the alert is received, into a DB2 table called *alert_table*. All servers send at least two alerts to indicate that they are alive and sending alerts. If the server sends an alert between these two alerts, **alert_s** inserts the alert into the table. Any error code that is not received in this interval is deleted from the table, as the error situation is considered to be rectified. This means that *alert_table* includes only current error codes and that rows are deleted when an error is corrected.

alert_s also writes alert information to a file. While only the current status of servers is shown in the *alert_table* table, the history file shows all alerts sent.

One suggested improvement to this program is to implement a GUI interface to display the *alert_table* table. Note the use of the continuation character, '➤', in the code below to indicate a formatting line break that's not present in the original code.

ALERT_S.SQC

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tcplib.h>
#include <time.h>
#include <sqlenv.h>

int main (void) {

  int    socket_num;
  struct tm * ctime;
  long   ltime;
  char   * getenv();
  FILE   * fn;
  char   * hosttosend;

  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    char   CLIENT_CODE[9];
```

```
  char   ERROR_CODE[5];
  short  RETURN_CODE;
  char   DBNAME[10];
  char   LASTOK_TMSTMP[27];
EXEC SQL END DECLARE SECTION;

hosttosend=getenv("SENDTOHOST");

while ( TRUE ) {
  socket_num=rec_so_init(hosttosend,"alert");
  strcpy(DBNAME,"d942tmx");
  exec sql connect to :DBNAME;
  if ( SQLCODE != 0 ) {
    printf("Cannot connect to database... SQLCODE = %ld
    ➤  \n",SQLCODE);
  }

  for (;;) {
    rec_so_req(socket_num);
    if (strcmp(clnt_rec.server_name,"") != 0) {
      time(&ltime);
      ctime=localtime(&ltime);
      fn=fopen("alert_s.out","a");
      fprintf(fn," %s %s %d %s \n",clnt_rec.server_name,
      ➤  clnt_rec.error_code,clnt_rec.return_code,asctime(ctime));
      fclose(fn);
      strcpy(CLIENT_CODE,clnt_rec.server_name);
      strcpy(ERROR_CODE,clnt_rec.error_code);
      RETURN_CODE=clnt_rec.return_code;
      if ( strcmp(clnt_rec.error_code,"LAST") == 0 ) {
        exec sql select TMSTMP into :LASTOK_TMSTMP from ALERT_TABLE
        ➤  where CLIENT_CODE=:CLIENT_CODE and ERROR_CODE='OKOK';
        if ( SQLCODE != 0 ) {
          strcpy(LASTOK_TMSTMP,"1901-01-01-01.01.01.000001");
        }
        exec sql delete from ALERT_TABLE where CLIENT_CODE=
        ➤  :CLIENT_CODE and TMSTMP <= :LASTOK_TMSTMP ;
        if ( SQLCODE != 0 ) {
          printf("Cannot delete from database... SQLCODE = %ld
          ➤  \n",SQLCODE);
        }
      }
      exec sql delete from ALERT_TABLE where CLIENT_CODE=:CLIENT_CODE
      ➤  and ERROR_CODE=:ERROR_CODE;
      exec sql insert into ALERT_TABLE
           values(:CLIENT_CODE,:ERROR_CODE,:RETURN_CODE,CURRENT
           ➤  TIMESTAMP);
      if ( SQLCODE != 0 ) {
        printf("Cannot insert into database... SQLCODE = %ld
        ➤  \n",SQLCODE);
```

```
        }
        exec sql commit;
      }
    }
  }
}
```

ALERT_C.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <tcplib.h>
#include <time.h>

int main ()
{
  struct  tm * ctime;
  long    ltime;
  int     rec_sock,sleep_time;
  char    localhost[MAXHOSTNAME+1];
  char    alert_cmd[201];
  char    alert_operator[3];
  long    alert_value;
  long    alert_out;
  char    *getenv();
  char    hosttosend[10];
  FILE    *fph;
  FILE    *fp;
  FILE    *fpr;

  fph = fopen("/u/pmkdbd2/mntdir/alert_parms","r");
  fscanf(fph,"%10s",hosttosend);
  fscanf(fph,"%ld",&sleep_time);
  fclose(fph);

  gethostname(localhost,MAXHOSTNAME);
  strcpy(clnt_rec.server_name,localhost);

  for (;;) {
    fp = fopen("/u/pmkdbd2/mntdir/alert.ctrl","r");
    for (;;) {
      fpr = fopen("/u/pmkdbd2/alert.out","r");
      if ( fgets(alert_cmd,200,fp) == NULL ) { break;}
      fscanf(fp,"%s\n",&alert_operator);
      fscanf(fp,"%ld %4s \n",&alert_value,&clnt_rec.error_code);
      system(alert_cmd);
      fscanf(fpr,"%ld",&alert_out);
      time(&ltime);
```

```
      ctime=localtime(&ltime);
      clnt_rec.return_code=alert_out;
      if (strcmp(alert_operator,"=") == 0)
        if (alert_out == alert_value)
          send_so_req(hosttosend,"alert");

       if (strcmp(alert_operator,">") == 0)
         if (alert_out > alert_value)
           send_so_req(hosttosend,"alert");

       if (strcmp(alert_operator,"<") == 0)
         if (alert_out < alert_value)
           send_so_req(hosttosend,"alert");

       if (strcmp(alert_operator,">=") == 0)
         if (alert_out >= alert_value)
           send_so_req(hosttosend,"alert");

       if (strcmp(alert_operator,"<=") == 0)
         if (alert_out <= alert_value)
           send_so_req(hosttosend,"alert");

       if (strcmp(alert_operator,"!=") == 0)
         if (alert_out != alert_value)
           send_so_req(hosttosend,"alert");
      fclose(fpr);
    }
    fclose(fp);
    sleep(sleep_time);
  }
}
```

SAMPLE ALERT_PARMS

```
AN942DH
10
```

SAMPLE ALERT.CTRL

```
echo 1 > /u/pmkdbd2/alert.out
=
1 OKOK
df -k | grep actlogs | awk '{ print $4 } ' | sed 's/%//' >
➤  /u/pmkdbd2/alert.out
>
80 ACTV
df -k| grep hd1 | awk '{ print $4 } ' | sed 's/%//' >
➤  /u/pmkdbd2/alert.out
>
```

```
80 HOME
df -k| grep db2temp | awk '{ print $4 } ' | sed 's/%//' >
➤  /u/pmkdbd2/alert.out
>
80 TEMP
df -k| grep arclogs | awk '{ print $4 } ' | sed 's/%//' >
➤  /u/pmkdbd2/alert.out
>
80 ARCH
df -k| grep dbdir | awk '{ print $4 } ' | sed 's/%//' >
➤  /u/pmkdbd2/alert.out
>
70 DBDR
ls -l /tmp/dump/vm* 2> /dev/null | grep -v chk | wc -l  >
➤  /u/pmkdbd2/alert.out
>
0 DUMP
echo 1 > /u/pmkdbd2/alert.out
=
1 LAST
```

TCPLIB.H

```
#define DRIVE_SZ       50
#define BACKLOG        500
#define MAXHOSTNAME    32

struct serv_str {
  int    retcode;
  } serv_rec;

struct clnt_str {
  char server_name[8];
  char error_code[5];
  int  return_code;
} clnt_rec;

int rec_so_init(char * hosttosend,char * service_name);
int rec_so_req(int s);
int send_so_req(char * host_name,char * service_name);
```

TCPLIB.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <netdb.h>
#include <tcplib.h>
#include <sqlenv.h>

int rec_so_init(char * hosttosend,char * service_name)
{
  int s,t;
  int i,rval;
  struct sockaddr_in sa;
  struct hostent *hp;
  struct servent *sp;
  char localhost[MAXHOSTNAME+1];

  if ((sp=getservbyname(service_name,"tcp")) == NULL) {
    fprintf(stderr,"%s: No service on this host\n",service_name);
    exit(1);
  }

  strcpy(localhost,hosttosend);

  if ((hp=gethostbyname(localhost)) == NULL) {
    fprintf(stderr," Cannot get local host info?\n");
    exit(1);
  }

  sa.sin_port=sp->s_port;
  bcopy((char *)hp->h_addr,(char *)&sa.sin_addr,hp->h_length);
  sa.sin_family=hp->h_addrtype;

  if((s=socket(hp->h_addrtype,SOCK_STREAM,0)) < 0) {
    perror("socket");
    exit(1);
  }

  if(bind(s,&sa,sizeof(sa)) < 0) {
    perror("bind");
    exit(1);
  }

  listen(s,BACKLOG);
  return(s);
}

int rec_so_req(int s)
{
  int t,i,rval;
  short index;
  struct sockaddr_in isa;

    i=sizeof(isa);
```

```
    if((t=accept(s,&isa,&i)) < 0) {
      perror("accept");
      exit(1);
    } else {

      strcpy(clnt_rec.server_name,"");
      strcpy(clnt_rec.error_code,"");
      clnt_rec.return_code=0;

      if ((rval=read(t,&clnt_rec,sizeof(clnt_rec))) < 0)
        perror("reading stream message");
      close(t);
    }
}

int send_so_req(char * host_name,char * service_name)
{
  int sock;
  int len;
  struct hostent *hp,*gethostbyname();
  struct sockaddr_in  server;
  struct servent *sp;

  sock=socket(AF_INET,SOCK_STREAM,0);
  if (sock < 0 ) {
    perror("Opening stream socket");
    exit(1);
  }

  server.sin_family = AF_INET;
  server.sin_len = sizeof(server);
  hp=gethostbyname(host_name);
  if (hp == 0) {
    fprintf(stderr, "%s: Unknown host \n", host_name);
    exit(2);
  }

  bcopy(hp->h_addr,&server.sin_addr,hp->h_length);

  if ((sp=getservbyname(service_name,"tcp")) == NULL) {
    fprintf(stderr,"%s: No service on this host\n",service_name);
    exit(1);
  }

  server.sin_port = sp->s_port;

  if (connect(sock,&server,sizeof(server)), 0) {
    perror("Connecting stream socket");
    exit(1);
  }
```

```
  if (write(sock,&clnt_rec,sizeof(clnt_rec)), 0) {
    perror("Writing on stream socket");
    exit(1);
  }

  close(sock);
}
```

Abdullah Ongul
DB2 DBA
Pamukbank (Turkey)

More AIX utilities

This article concludes my review of AIX utilities that began in the February issue of *AIX Update*.

NL

The **nl** utility adds line numbers to a file. Although this seems like a simple task, **nl** has a great number of options that make it a little complicated to use. To illustrate some of these options, we are going to undertake the old-fashioned task of adding line numbers to a COBOL program. I chose this example because it provides an opportunity to illustrate many of the features of **nl**. Listing 11 is *hello.txt*, a COBOL program with missing line numbers.

LISTING 11: A COBOL PROGRAM

```
$ cat hello.txt
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

PROGRAM-BEGIN.
    DISPLAY "Hello world."
```

```
PROGRAM-DONE.
    STOP RUN.
```

The first pass uses the simplest version of the **nl** command to add line numbers, as in Listing 12. The output has several problems. The numbers start with '1' and increment by one. COBOL usually operates in increments of 10 or 100. An increment of one is valid, but not usual. COBOL numbering also includes leading zeros. The **nl** utility has skipped blank lines, though they should also be numbered. Also, while not visible in print, the separator between the line number and the original line is a tab, which many COBOL compilers can't handle. This type of numbering is the default behaviour of **nl**.

LISTING 12: A SIMPLE USE OF NL

```
$ nl <hello.txt >hello.cbl
cat hello.cbl
     1    IDENTIFICATION DIVISION.
     2    PROGRAM-ID. HELLO.
     3    ENVIRONMENT DIVISION.
     4    DATA DIVISION.
     5    PROCEDURE DIVISION.

     6    PROGRAM-BEGIN.
     7        DISPLAY "Hello world."

     8    PROGRAM-DONE.
     9        STOP RUN.
```

Let's tackle these problems one at a time. The separator character can be changed from a tab to a space by using the **-s** switch. The modified version of the command would be:

```
$ nl -s" " <hello.txt >hello.cbl
```

The format of the number itself is controlled by several options. The **-w** option is used to specify the width of the number. For COBOL, this is six characters. The default for **nl** also happens to be six, but I include the option for completeness:

```
$ nl -s" " -w6 <hello.txt >hello.cbl
```

The **-v** option lets you specify the starting number and **-i** lets you specify the increment. Putting all this together, Listing 13 specifies the separator, width, starting number, and increment:

LISTING 13: USING NL WITH OPTIONS

```
$ nl -s" " -w6 -v100 -i100 <hello.txt >hello.cbl
cat hello.cbl
   100 IDENTIFICATION DIVISION.
   200 PROGRAM-ID. HELLO.
   300 ENVIRONMENT DIVISION.
   400 DATA DIVISION.
   500 PROCEDURE DIVISION.

   600 PROGRAM-BEGIN.
   700     DISPLAY "Hello world."

   800 PROGRAM-DONE.
   900     STOP RUN.
```

This is closer to our goal, but still needs some work. The number format is controlled by the **-n** option. There are three formats:

- Left-justified with leading zeros suppressed: **-nln**
- Right-justified with leading zeros suppressed: **-nrn** (this is the default)
- Right-justified with leading zeros kept: **-nrz**.

We need those leading zeros, so we use the **-nrz** option:

LISTING 14: NUMBER FORMATTING IN NL

```
$ nl -s" " -w6 -v100 -i100 -nrz <hello.txt >hello.cbl
cat hello.cbl
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLO.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.

000600 PROGRAM-BEGIN.
000700     DISPLAY "Hello world."

000800 PROGRAM-DONE.
000900     STOP RUN.
```

The default behaviour of **nl** is to skip blank lines, as in the result shown above. The treatment of blank lines can be modified by the **-b** switch. Some **-b** options are:

-ba Number all lines

-bt Number only text lines (the default behaviour)

-bp*string* Number only lines containing *string*.

The last option is interesting: a somewhat artificial example of it in use is shown in Listing 15, in which only lines containing the word 'PROGRAM' are numbered.

LISTING 15: NUMBERING ONLY SELECTED LINES

```
$ nl -s" " -w6 -v100 -i100 -nrz -bpPROGRAM <hello.txt >hello.cbl
cat hello.cbl
IDENTIFICATION DIVISION.
000100 PROGRAM-ID. HELLO.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

000200 PROGRAM-BEGIN.
    DISPLAY "Hello world."

000300 PROGRAM-DONE.
    STOP RUN.
```

In our case, we need the **-ba** option to number all lines. Listing 16 is the final version of the command and the output result.

LISTING 16: THE FINAL VERSION OF NL

```
$ nl -s" " -w6 -v100 -i100 -nrz -ba <hello.txt >hello.cbl
cat hello.cbl
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLO.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000500 PROCEDURE DIVISION.
000600
000700 PROGRAM-BEGIN.
000800     DISPLAY "Hello world."
000900
001000 PROGRAM-DONE.
001100     STOP RUN.
```

While **nl** sounds deceptively simple, its options make it possible to perform a wide range of numbering tasks. It also includes switches to

recognize the start of new pages, to number pages, and to restart numbering so that the lines on each page start at ‘1’.

BASENAME

basename strips off the directory portion of a file full name and returns just the file name itself, as in the following example:

```
$ basename /this/is/a/file.txt
file.txt
$
```

In the example above, the file and path don’t even need to exist. The **basename** command analyses the string and takes out the pieces that it knows would correspond to the path. Basically it leaves the last piece of the string that does not contain a slash, ‘/’. It also removes any trailing slash in order to identify the base portion of the name. In the following listing, the final slash is removed before evaluating the string and returning the word ‘file’ as the base for the string.

```
$ basename /this/is/a/file/
file
$
```

The **basename** utility is very useful for replicating files across multiple directories: in the example that follows, we consider a hypothetical system in which a main directory contains four subdirectories. The first subdirectory contains the current version of a document, the second contains a new version to install, and up to two earlier versions of the same document are stored in a further two subdirectories. The subdirectories are *docs/current* (current version), *docs/install* (new version to replace existing documents in *docs/current*), *docs/old* (the most recently retired version from *docs/current*), and *docs/oldest* (an older version).

The utility that implements this (shown in Figure 17) works as follows: if a document exists in more than one version in more than one directory, it has the same name in all directories. A process then checks the names of all new documents in the *docs/install* directory. The process uses a ‘bucket brigade’ to move the version in *docs/old* to *docs/oldest*, the one in *docs/current* to *docs/old*, and the one in *docs/install* to *docs/current*. **basename** is used to extract just the filename

of each document in *docs/install*. The filename is placed in the variable *fname*; from there, *$fname* is used in various tests and procedures.

The 'for name' loop in line 1 sets the variable *$name* to each filename that matches *docs/install/**. In line 3, **basename** is used to extract the filename into variable *$fname*. Line 4 tests whether a file of the same name exists in the *docs/old* directory. If it does, it is moved to *docs/oldest* in line 6. This is repeated for the *docs/current* and *docs/old* in lines 8 to 11. Finally in line 12 the file in *docs/install* is moved into *docs/current*.

LISTING 17: USING BASENAME TO CREATE A BUCKET BRIGADE

```
 1 for name in /docs/install/*
 2 do
 3     fname=`basename $name`
 4     if [ -f /docs/old/$fname ]
 5     then
 6         mv /docs/old/$fname /docs/oldest/$fname
 7     fi
 8     if [ -f /docs/current $fname ]
 9     then
10         mv /docs/current/$fname /docs/old/$fname
11     fi
12     mv /docs/install/$fname /docs/current/$fname
13 done
```

This logic assumes that the *docs/install* directory contains only files that are to be installed in *docs/current* and that it contains no subdirectories.

The **basename** utility can also be used to strip a filename from the path name, returning it without an extension. The extension to be stripped is passed to the utility on the command line after the path. In the following example, *.txt* is the extension to be stripped:

```
$ basename /this/is/a/file.txt .txt
file
$
```

DIRNAME

The **dirname** utility is the complement of **basename**. It returns the

'other half' of a path/filename string: the path. The following example illustrates this:

```
$ dirname /this/is/a/file.txt
/this/is/a
dirname /x/y/z/
/x/y
$
```

In the second use of **dirname** in the above example, we can see that **dirname** removes the trailing slash ('/') before interpreting the string. The result is that the **dirname** portion of */x/y/z/* is */x/y*.

dirname has no options and simply returns the directory portion of a file path. While **dirname** is a useful tool, I find that I don't use it as often as **basename**.

TIME

The **time** command times a process and is excellent for analysing the performance of a shell script or command. Simply type **time** followed by the command that you wish to time. Three values are printed when the program or script finishes: the length of time (real-world time) spent on processing the program, the total user time spent in the program, and the total system time spent (CPU overhead). The first figure is perhaps the most useful, though the third tells you how busy your CPU is.

```
$ time bigjob.sh
real 10m 10.55s
user  4m 08.47s
sys   1m 12.14s
```

Some older versions of **time** report the results in seconds only, as in the following example:

```
$ time bigjob.sh
real  610.55
user  248.47
sys    72.14
```

TEE

The **tee** utility is one of my personal favourites and is very simple. The

command is intended to be used in a pipe to capture the standard output of another command, display it on the screen, and then copy it to a file. In Listing 18, the directory listing is displayed on the screen, and is also copied to the file *dir.txt*. Using **cat** to type *dir.txt* shows that it contains the same information that was displayed on the screen:

LISTING 18: USING TEE

```
$ ls -l|tee dir.txt
total 141
-rwxrwxrwx   1 mjb   group    16850 Apr 12 16:13 SMALL01.DOC
-rwxrwxrwx   1 mjb   group    14881 Apr 12 20:51 SMALL02.DOC
-rwxrwxrwx   1 mjb   group    17758 Jun 13 01:29 Small03.doc
-rwxrwxrwa   1 mjb   group    12791 Jul 12 22:44 Small04.doc
-rwxrwxrwx   1 mjb   group     4232 Jun 12 00:03 Smallxx.doc
drwxrwxrwx   1 mjb   group        0 Jul 12 21:25 docs
-rwxrwxrwx   1 mjb   group      261 Jun 13 01:08 hello.cbl
-rwxrwxrwx   1 mjb   group      184 Jun 13 00:59 hello.txt
$ cat dir.txt
total 141
-rwxrwxrwx   1 mjb   group    16850 Apr 12 16:13 SMALL01.DOC
-rwxrwxrwx   1 mjb   group    14881 Apr 12 20:51 SMALL02.DOC
-rwxrwxrwx   1 mjb   group    17758 Jun 13 01:29 Small03.doc
-rwxrwxrwa   1 mjb   group    12791 Jul 12 22:44 Small04.doc
-rwxrwxrwx   1 mjb   group     4232 Jun 12 00:03 Smallxx.doc
drwxrwxrwx   1 mjb   group        0 Jul 12 21:25 docs
-rwxrwxrwx   1 mjb   group      261 Jun 13 01:08 hello.cbl
-rwxrwxrwx   1 mjb   group      184 Jun 13 00:59 hello.txt
$
```

tee normally creates the named file, overwriting an existing one, though the **-a** option causes the new information to be appended to an existing file. Listing 19 extends the *dir.txt* file just created.

LISTING 19: APPENDING TO A TEE FILE

```
$ echo "Those are all the files"|tee -a dir.txt
Those are all the files
$ cat dir.txt
total 141
-rwxrwxrwx   1 mjb   group   16850 Apr 12 16:13 SMALL01.DOC
-rwxrwxrwx   1 mjb   group   14881 Apr 12 20:51 SMALL02.DOC
-rwxrwxrwx   1 mjb   group   17758 Jun 13 01:29 Small03.doc
-rwxrwxrwa   1 mjb   group   12791 Jul 12 22:44 Small04.doc
-rwxrwxrwx   1 mjb   group    4232 Jun 12 00:03 Smallxx.doc
drwxrwxrwx   1 mjb   group       0 Jul 12 21:25 docs
```

```
-rwxrwxrwx   1 mjb   group     261 Jun 13 01:08 hello.cbl
-rwxrwxrwx   1 mjb   group     184 Jun 13 00:59 hello.txt
Those are all the files
$
```

tee is very useful for logging messages at the same time as displaying them on the screen. Assuming that **job.sh** displays information on the screen while it is processing, Listing 20 copies everything that goes to the screen into the file *job.log*. In this example, the log file is set up with date and time information and then the job is run, appending the information to the log file.

LISTING 20: USING TEE IN LOGGING

```
$ echo "Starting job.sh"> job.log
$ date>> job.log
$ job.sh|tee -a job.log
Processing Check File
Check Entries Valid - No Errors
Printing Checks
End Of Job
$ cat job.log
Starting job.sh
Mon Jul 13 13:36:22 PDT 1998
Processing Check File
Check Entries Valid - No Errors
Printing Checks
End Of Job
$
```

The **tee** utility can cut down a lot of double logging, where a message is sent to the screen and either another message or the same one is also captured in a log file. It can also be used to debug long pipes by putting a **tee** between critical points in the pipe to capture output and see what is happening at that stage, as in the following example:

```
$ls -l|tee step1.txt|sort|tee step2.txt|awk -f awk.script
```

That wraps up this article on some of the ‘smaller’ Unix commands. I hope I’ve given you some ideas on how to use them to help you in various tasks.

Mo Budlong
President
KCSI (USA)

Removing files by timestamp (part 2)

This month's instalment concludes this article on removing files by timestamp, the first instalment of which appeared in last month's issue. Note that the code below follows directly on from that published last month.

```
##############################################################################
# Name     : InitializeLogFile
#
# Overview : Initialize the log file.
#
# Returns  : $TRUE or $FALSE
##############################################################################
InitializeLogFile ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP

if [ ! -f ${LOG_FILE} ]
   then
   # initialize the log file
   #
   ( > ${LOG_FILE} ) > /dev/null 2>&1
   if [ $? -ne 0 ]
   then
        DisplayMessage E "${LOG_NOT_INITIALIZED}"
        return $FALSE
   fi
   echo " Log File for File Removal on ${DATETIME}" > ${LOG_FILE}
   echo " Directory=$DIR_NAME" >> ${LOG_FILE}
   echo " Userid =${USERID}" >> ${LOG_FILE}
   echo "============================================================"\
      >> ${LOG_FILE}
   return $TRUE
fi

# Log file exists: copy it to ${LOG_FILE}.old file
cp ${LOG_FILE} ${LOG_FILE}.old

# Initialize the log file
( > ${LOG_FILE} ) > /dev/null 2>&1
if [ $? -ne 0 ]
then
   DisplayMessage E "${LOG_NOT_INITIALIZED}"
   return $FALSE
fi
```

```
echo " Log file for file removal on ${DATETIME}" > ${LOG_FILE}
echo " Directory=$DIR_NAME" >> ${LOG_FILE}
echo " Userid =${USERID}" >> ${LOG_FILE}
echo "============================================================" >>\
   ${LOG_FILE}
return $TRUE
}

#############################################################################
# Name     : ParseCommandLine
#
# Overview : Parse command line parameters.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 The following command line parameters are expected:
#            D=<directory name>
#            d=<date>
#            i=<y|n>
#            l=<logfile name>
#############################################################################
ParseCommandLine ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP

DIR_NAME=""
DATE=""
INTERACTIVE=""
LOG_FILE=""

# Establish argument count
if [ ${ARGC} -lt 1 -o ${ARGC} -gt 4 ]
then
   DisplayMessage E "${INVALID_ARGC}"
   DisplayMessage E "${USAGE}"
   return $FALSE
fi

# Process arguments
INDEX=1
while [ ! $INDEX -gt $ARGC ]
do
   # Extract next argument line
   ARG_LINE=`echo "${ARGV}" | cut -d' ' -f${INDEX}`

   # Extract argument type
   ARG_TYPE=`echo ${ARG_LINE} | cut -c1-2`
   case "${ARG_TYPE}" in
      D=|d=|i=|l=)
```

```
            # Check for duplicate argument type
            if [ "${ARG_TYPE}" = "D=" -a "${DIR_NAME}" != "" ]
            then
               DisplayMessage E "${DUP_ARG}" ;

            elif [ "${ARG_TYPE}" = "D=" -a "${DIR_NAME}" = "" ]
            then
               # Store this argument value
               DIR_NAME=`echo "${ARG_LINE}" | cut -d'=' -f2` ;

            elif [ "${ARG_TYPE}" = "d=" -a "${DATE}" != "" ]
            then
               DisplayMessage E "${DUP_ARG}" ;

            elif [ "${ARG_TYPE}" = "d=" -a "${DATE}" = "" ]
            then
               # Store this argument value
               DATE=`echo "${ARG_LINE}" | cut -d'=' -f2`;

            elif [ "${ARG_TYPE}" = "i=" -a "${INTERACTIVE}" != "" ]
            then
               DisplayMessage E "${DUP_ARG}" ;

            elif [ "${ARG_TYPE}" = "i=" -a "${INTERACTIVE}" = "" ]
            then
               # Store this argument value
               INTERACTIVE=`echo "${ARG_LINE}" | cut -d'=' -f2` ;

            elif [ "${ARG_TYPE}" = "l=" -a "${LOG_FILE}" != "" ]
            then
               DisplayMessage E "${DUP_ARG}" ;

            elif [ "${ARG_TYPE}" = "l=" -a "${LOG_FILE}" = "" ]
            then
               # Store this argument value
               LOG_FILE=`echo "${ARG_LINE}" | cut -d'=' -f2` ;
            fi ;;

         * ) DisplayMessage E "${INVALID_ARG_TYPE}" ;
            DisplayMessage I "${USAGE}" ;
            return $FALSE ;;
      esac
      INDEX=`expr $INDEX + 1`
   done

   return $TRUE
   }

   ##############################################################################
   # Name     : FileQualifyForRemoval
```

```
#
# Overview : Check whether the input file qualifies for removal.
#
# Input    : File Name
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 The last modification date of file is compared with
#            the argument $DATE. The file qualifies for removal if
#            its last modification date is earlier than $DATE.
################################################################################
FileQualifyForRemoval ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP

# Assign parameter
P_FILE_NAME="$1"

# Work out last modified date for the input file
LAST_MODIFIED_DAY=`istat ${P_FILE_NAME} | grep "Last modified" | \
                  cut -c15- | awk {'print $3'}`
LAST_MODIFIED_MON=`istat ${P_FILE_NAME} | grep "Last modified" | \
                  cut -c15- | awk {'print $2'}`
LAST_MODIFIED_YEAR=`istat ${P_FILE_NAME} | grep "Last modified" | \
                  cut -c15- | awk {'print $5'}`
LAST_MODIFIED_TIME=`istat ${P_FILE_NAME} | grep "Last modified" | \
                  cut -c15- | awk {'print $4'} | sed s/':'//g`

# Rework month
if   [ "${LAST_MODIFIED_MON}" = "Jan" ]
then
      LAST_MODIFIED_MON="01"
elif [ "${LAST_MODIFIED_MON}" = "Feb" ]
then
      LAST_MODIFIED_MON="02"
elif [ "${LAST_MODIFIED_MON}" = "Mar" ]
then
      LAST_MODIFIED_MON="03"
elif [ "${LAST_MODIFIED_MON}" = "Apr" ]
then
      LAST_MODIFIED_MON="04"
elif [ "${LAST_MODIFIED_MON}" = "May" ]
then
      LAST_MODIFIED_MON="05"
elif [ "${LAST_MODIFIED_MON}" = "Jun" ]
then
      LAST_MODIFIED_MON="06"
elif [ "${LAST_MODIFIED_MON}" = "Jul" ]
then
      LAST_MODIFIED_MON="07"
```

```
elif [ "${LAST_MODIFIED_MON}" = "Aug" ]
then
      LAST_MODIFIED_MON="08"
elif [ "${LAST_MODIFIED_MON}" = "Sep" ]
then
      LAST_MODIFIED_MON="09"
elif [ "${LAST_MODIFIED_MON}" = "Oct" ]
then
      LAST_MODIFIED_MON="10"
elif [ "${LAST_MODIFIED_MON}" = "Nov" ]
then
      LAST_MODIFIED_MON="11"
elif [ "${LAST_MODIFIED_MON}" = "Dec" ]
then
      LAST_MODIFIED_MON="12"
fi

# Build the $LAST_MODIFIED_DATE
LAST_MODIFIED_DATE="${LAST_MODIFIED_DAY}${LAST_MODIFIED_MON}\
   ${LAST_MODIFIED_YEAR}${LAST_MODIFIED_TIME}"

if [ "${LAST_MODIFIED_DATE}" = "${DATE}" ]
then
   # File does not qualify for removal
   return $FALSE
fi

# Sort $LAST_MODIFIED_DATE against $DATE
echo "$LAST_MODIFIED_DATE" > ${TEMP_FILE_1}
echo "$DATE" >> ${TEMP_FILE_1}

# The date is held in format <ddmmyyyyhhmiss> eg '01012000121060'
# The sort order is ascending
cat ${TEMP_FILE_1} | sort -t: -k 1.5,1.8 -k 1.3,1.4 -k 1.1,1.2 \
    -k 1.9,1.10 -k 1.11,1.12 -k 1.13,1.14 > ${TEMP_FILE_2}

# Read the top line from file ${TEMP_FILE_2}
LINE=`head -1 ${TEMP_FILE_2}`

if [ "${LINE}" = "${LAST_MODIFIED_DATE}" ]
then
   # The file was modified before the input date, therefore
   # it qualifies for removal
   return $TRUE
else
   # The file does not qualify for removal
   return $FALSE
fi
}
```

```
##############################################################################
# Name     : FileToBeRemoved
#
# Overview : Confirm the removal of a specified file.
#
# Input    : File Name
#
# Returns  : $TRUE or $FALSE
##############################################################################
FileToBeRemoved ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP

P_FILE="$1"
while true
do
   clear
   if [ "${FILE_TYPE}" != "d" ]
   then
      echo "Remove file ${DIR_NAME}/$P_FILE(Y/N):\c"
   else
      echo "Remove directory ${DIR_NAME}/$P_FILE(Y/N):\c"
   fi
   read REPLY
   case $REPLY in
      Y|y ) return $TRUE ;;
      N|n ) return $FALSE ;;
        * ) DisplayMessage E "${INVALID_ENTRY}" ;;
   esac
done
}

##############################################################################
# Name     : RemoveFiles
#
# Overview : Remove all files that qualify for removal.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 If the user-id is not the same as the file owner and
#            is not root, the function does not remove the file,
#            writes a log entry to that effect.
#
#          2 If a directory is to be removed, then the function
#            removes the directory and all its contents.
##############################################################################
RemoveFiles ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
```

```
DisplayMessage I "${WORKING}" N

# Store current directory
CURDIR=`pwd`

# Switch to required directory
cd $DIR_NAME 2> ${TEMP_FILE_1}
if [ $? -ne 0 ]
then
   SYSERROR=`cat ${TEMP_FILE_1}`
   DIR="${DIR_NAME}"
   DisplayMessage E "${DIR_NOT_ACCESSABLE}"
   DisplayMessage E "${OS_ERROR}"
   return $FALSE
fi

# Prepare a list of files from the directory
# $1=File type $9=File name
ls -al | awk {'print $1" "$9'} > ${FILE_LIST}
if [ ! -s ${FILE_LIST} ]
then
   DIR="${DIR_NAME}"
   DisplayMessage I "${DIR_EMPTY}" Y
   echo "refbot.sh:INFO:Directory is empty" >> ${LOG_FILE}
   return $TRUE
fi

# Prepare a list of files from that qualify for removal
cat ${FILE_LIST} | while read FILE_TYPE FILE_NAME
do
   # Ignore directories . and ..
   if [ "${FILE_NAME}" = "." -o "${FILE_NAME}" = ".." -o \
      "${FILE_NAME}" = "" ]
   then
      continue
   fi
   if FileQualifyForRemoval ${FILE_NAME}
   then
      FILE_TYPE=`echo $FILE_TYPE | cut -c1-1`
      echo "${FILE_NAME} ${FILE_TYPE}" >> ${QUALIFIED_FILE_LIST}
   fi
done
if [ ! -s ${QUALIFIED_FILE_LIST} ]
then
   DIR="${DIR_NAME}"
   DisplayMessage I "${NO_FILE_REMOVED}" Y
   echo "refbot.sh:INFO:No files qualifie for removal" >> ${LOG_FILE}
   return $TRUE
fi
```

```
unalias rm

# Populate an array with qualified file names
INDEX=1
cat ${QUALIFIED_FILE_LIST} | while read FILE_NAME FILE_TYPE
do
   QUALIFIED_FILE_ARRAY[$INDEX]="${FILE_NAME}"
   QUALIFIED_FILE_TYPE[$INDEX]="${FILE_TYPE}"
   INDEX=`expr $INDEX + 1`
done

# Process file names from the array
INDEX=1
while [ "${QUALIFIED_FILE_ARRAY[$INDEX]}" != "" ]
do
   FILE_NAME="${QUALIFIED_FILE_ARRAY[$INDEX]}"
   FILE_TYPE="${QUALIFIED_FILE_TYPE[$INDEX]}"

   # Establish file owner
   FILE_OWNER=`ls -l ${FILE_NAME} | awk {'print $3'}`
   if [ "${FILE_OWNER}" != "${USERID}" ]
   then
      if [ "${USERID}" != "root" ]
      then
      if [ "${FILE_TYPE}" = "d" ]
      then
         echo "refbot.sh:INFO:Cannot remove directory $FILE_NAME;
              ➤  not owner" >> ${LOG_FILE}
      else
         echo "refbot.sh:INFO:Cannot remove file $FILE_NAME;
              ➤  not owner" >> ${LOG_FILE}
      fi
      INDEX=`expr $INDEX + 1`
      continue
      fi
   fi
   if [ "${INTERACTIVE}" = "Y" -o "${INTERACTIVE}" = "y" ]
   then
      if FileToBeRemoved "${FILE_NAME}"
      then
         if [ "${FILE_TYPE}" != "d" ]
         then
         echo "refbot.sh:INFO:Removing file $FILE_NAME" >>
              ➤  ${LOG_FILE}
         else
         echo "refbot.sh:INFO:Removing directory $FILE_NAME" >>
              ➤  ${LOG_FILE}
         fi
         if [ "${FILE_TYPE}" = "d" ]
         then
```

```
         ( rm -r ${FILE_NAME} ) >> ${TEMP_FILE_1} 2>&1
      else
         ( rm ${FILE_NAME} ) >> ${TEMP_FILE_1} 2>&1
      fi
      if [ $? -eq 0 ]
      then
         if [ "${FILE_TYPE}" != "d" ]
         then
         echo "refbot.sh:INFO:Removed file" >> ${LOG_FILE}
         else
         echo "refbot.sh:INFO:Removed directory" >> ${LOG_FILE}
         fi
      else
         if [ "${FILE_TYPE}" != "d" ]
         then
         echo "refbot.sh:ERROR:Failed to remove file" >>
              ➤  ${LOG_FILE}
         cat ${TEMP_FILE_1} >> ${LOG_FILE}
         else
         echo "refbot.sh:ERROR:Failed to remove directory" >>
              ➤  ${LOG_FILE}
         cat ${TEMP_FILE_1} >> ${LOG_FILE}
         fi
      fi
   else
      if [ "${FILE_TYPE}" != "d" ]
      then
      echo "refbot.sh:INFO:Not removing file $FILE_NAME" >>
           ➤  ${LOG_FILE}
      else
      echo "refbot.sh:INFO:Not removing directory $FILE_NAME">>
           ➤  ${LOG_FILE}
      fi
   fi
else
   if [ "${FILE_TYPE}" != "d" ]
   then
      echo "refbot.sh:INFO:Removing file $FILE_NAME" >>
           ➤  ${LOG_FILE}
   else
      echo "refbot.sh:INFO:Removing directory $FILE_NAME" >>
           ➤  ${LOG_FILE}
   fi
   if [ "${FILE_TYPE}" != "d" ]
   then
      ( rm ${FILE_NAME} ) >> ${TEMP_FILE_1} 2>&1
   else
      ( rm -r ${FILE_NAME} ) >> ${TEMP_FILE_1} 2>&1
   fi
   if [ $? -eq 0 ]
```

```
      then
         if [ "${FILE_TYPE}" != "d" ]
         then
            echo "refbot.sh:INFO:Removed file" >> ${LOG_FILE}
         else
            echo "refbot.sh:INFO:Removed directory" >> ${LOG_FILE}
         fi
      else
         if [ "${FILE_TYPE}" != "d" ]
         then
            echo "refbot.sh:ERROR:Failed to remove file" >>
                 ➤  ${LOG_FILE}
         else
            echo "refbot.sh:ERROR:Failed to remove directory" >>
                 ➤  ${LOG_FILE}
         fi
         cat ${TEMP_FILE_1} >> ${LOG_FILE}
      fi
   fi
   INDEX=`expr $INDEX + 1`
done

# Switch to current directory
( cd $CURDIR ) 2> $TEMP_FILE_1
if [ $? -ne 0 ]
then
   SYSERROR=`cat ${TEMP_FILE_1}`
   DIR="${CURDIR}"
   DisplayMessage E "${DIR_NOT_ACCESSABLE}"
   DisplayMessage E "${OS_ERROR}"
   return $FALSE
fi
}

##############################################################################
# Name     : main
#
# Overview : The function invokes all others.
#
# Notes    1 The function calls the following functions:
#            o InitializeVariables
#            o ParseCommandLine
#            o RemoveFiles
#            o ProcessExit
##############################################################################
main ()
{
InitializeVariables
if ! ParseCommandLine
then
```

```
   ProcessExit $FEC
fi

if ! ValidateArguments
then
   ProcessExit $FEC
fi
if ! InitializeLogFile
then
   ProcessExit $FEC
fi
RemoveFiles

# View the log file
view ${LOG_FILE}

ProcessExit $SEC
}

# Store command line arguments and argument count
ARGV="$@"
ARGC="$#"

# Invoke main
main
```

SAMPLE LOG FILE

```
Log File for File Removal on 26/08/1999 at 13:30:36
Directory=/tmp
Userid   =ecatmgr
=====================================
refbot.sh:INFO:Can not remove file 24674t2; not owner
refbot.sh:INFO:Can not remove file 29988t1; not owner
refbot.sh:INFO:Can not remove file llp16.dat; not owner
refbot.sh:INFO:Not removing file refbot.log
refbot.sh:INFO:Not removing file refbot.log.old
refbot.sh:INFO:Not removing file refbot_19570_1.dat
refbot.sh:INFO:Not removing file refbot_19570_1.tmp
refbot.sh:INFO:Can not remove directory tmp; not owner
refbot.sh:INFO:Removing file xxxx
refbot.sh:INFO:Removed file
```

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

SSCCARS (part 5)

This is the final part of this article on a group of utilities that together comprise a source code management system. The first part of this article appeared in the November 1999 issue of *AIX Update* (Issue 49).

BR.SH

br.sh is the module that builds either a full or patch release that contains source files to be installed. A full release contains all the latest source files while a patch release contains only those source files that have been modified after a specified date.

When building a patch release, the system maintains a small file in directory $ROOT_RELEASE_DIR that contains the date on which the last patch release was built. Any source files that were modified after this date are included in the new patch release.

BR.SH

```
#! /bin/ksh
###############################################################################
# Name     : br.sh
#
# Overview : Prepares either a full release with the latest
#            sources or a patch release with the sources
#            that were modified since the last patch build
#            date, stored in a specified release directory.
#
# Input    : Release type: F = Full P = Patch
#
# Notes    1 The script uses the following global variables
#            to locate all the required source directories:
#            o $SOURCE_DIR
#            o $FILE_EXTS (file extensions)
#
#          2 The script contains following functions:
#            o DefineModuleVariables
#            o InitializeReleaseLogFile
#            o GetReleaseDirectoryName
#            o ProcessExit
#            o CreateReleaseDirectory
```

```
#           o ProcessRelease
#           o ProcessLastPatchBuiltDate
#           o ProcessLastPatchBuiltDate
#           o FileToBeIncluedInPatch
#           o PrepareReleaseDocument
#           o UpdateLastPatchBuiltDate
#           o UpdateSmart
#           o GetReleaseDescription
#
#         3 The script is called from ssccars.sh
##############################################################################
##############################################################################
# Name     : DefineModuleVariables
#
# Overview : Defines all module variables.
##############################################################################
DefineModuleVariables ()
{
DATETIME=`date "+%d/%m/%y at %H:%M:%S"`
TODAY=`date +%d%m%y`
PROG="br.sh"
TEMP_FILE="/tmp/br_$$.tmp"
# used in copying the file
FILE_INDEX=0
FILE_NAME_ARRAY[$FILE_INDEX]=""
RELEASE_DOCUMENT=""                # readme.doc file for the release
}
##############################################################################
# Name     : GetReleaseDirectoryName
#
# Overview : Gets a release directory name from the user.
##############################################################################
GetReleaseDirectoryName ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
echo "Getting the release directory name" >> ${RELEASE_LOG_FILE}
while true
do
  clear
  echo "Enter a release subdirectory name(q to quit):\c"
  read RELEASE_NAME
  case $RELEASE_NAME in
    "") if [ "${FUNCTION_INTERRUPTED}" = "Y" ]
        then
          FUNCTION_INTERRUPTED=N ;
        else
          Display E "${INVALID_ENTRY}" ;
        fi ;;
   q|Q) FUNCTION_ABORTED=Y;
```

```
        DisplayMessage I "${FUNCTION_ABORTING}" ;
        echo "Release aborted" >> ${RELEASE_LOG_FILE} ;
        return $FALSE ;;
    *) if [ -d ${ROOT_RELEASE_DIR}/${RELEASE_NAME} ]
        then
          RELEASE_DIR="${ROOT_RELEASE_DIR}/${RELEASE_NAME}"
          DisplayMessage E "${REL_DIR_EXISTS}" ;
        else
          break ;
        fi ;;
  esac
done
# log a message
echo "Release directory name is ${RELEASE_NAME} " >>
➤  ${RELEASE_LOG_FILE}
return $TRUE
}
##############################################################################
# Name     : CreateReleaseDirectory
#
# Overview : Creates the required release directory.
##############################################################################
CreateReleaseDirectory ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
RELEASE_DIR="${ROOT_RELEASE_DIR}/${RELEASE_NAME}"
echo "Creating release directory ${RELEASE_DIR}" >> ${RELEASE_LOG_FILE}
(mkdir $RELEASE_DIR) > ${TEMP_FILE} 2>&1
if [ $? -ne 0 ]
then
  ERROR_MSG=`cat $TEMP_FILE |  head -1`
  DisplayMessage E "${OS_ERROR}"
  echo "Failed to create release directory" >> ${RELEASE_LOG_FILE}
  echo "${ERROR_MSG}" >> ${RELEASE_LOG_FILE}
  rm ${TEMP_FILE}
  return $FALSE
else
  DisplayMessage I "${REL_DIR_CREATED}"
  echo "Successfully created release directory" >> ${RELEASE_LOG_FILE}
  return $TRUE
fi
}
##############################################################################
# Name     : PrepareReleaseDocument
#
# Overview : Prepares the release document.
#
# Returns  : $TRUE  or  $FALSE
##############################################################################
```

```
PrepareReleaseDocument ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
echo "Updating release document" >> ${RELEASE_LOG_FILE}
# prepare file name
RELEASE_DOCUMENT="${RELEASE_DIR}/README.doc"
# write header
HEADER="Release details for release ${RELEASE_NAME} on ${DATETIME}"
FormatUnderscores   "$HEADER"
echo "  $HEADER   "          >  ${RELEASE_DOCUMENT}
echo "  $UNDERSCORE\n"       >> ${RELEASE_DOCUMENT}
#  put all the source names in the document
echo "Following sources have been released"  >> ${RELEASE_DOCUMENT}
echo "====================================="  >> ${RELEASE_DOCUMENT}
# remove header information from the list before appending
cat ${RELEASED_SOURCE_LIST} | sed 1,5d       >> ${RELEASE_DOCUMENT}
# allow user to modify the files
DisplayMessage  I "${EDIT_RELEASE_DOC}"
vi ${RELEASE_DOCUMENT}
}
##############################################################################
# Name     : GetReleaseDescription
#
# Overview : Processes the last patch build date which is held in
#            $LAST_PATCH_BUILT_DATE_FILE.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 The function calls the following function:
#            o  ValidateLastPatchBuiltDate
##############################################################################
GetReleaseDescription ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
echo "Getting the release description" >> ${RELEASE_LOG_FILE}
while true
do
  clear
  echo "Enter a release description (max 60 characters, or q to
  ➤  quit):\c"
  read  RELEASE_DESC
  case  $RELEASE_DESC  in
    "") if  [ "${FUNCTION_INTERRUPTED}"  = "Y" ]
        then
          FUNCTION_INTERRUPTED=N ;
        else
          Display E "${INVALID_ENTRY}" ;
        fi ;;
   q|Q) FUNCTION_ABORTED=Y;
```

```
        DisplayMessage I "${FUNCTION_ABORTING}" ;
        echo "Release aborted" >> ${RELEASE_LOG_FILE} ;
        return $FALSE ;;
     *) RELEASE_DESC=`echo $RELEASE_DESC | cut -c1-60`;
        break ;;
  esac
done
# log a message
echo "Release description:${RELEASE_DESC} " >> ${RELEASE_LOG_FILE}
return $TRUE
}
##############################################################################
# Name     : UpdateLastPatchBuiltDate
#
# Overview : Updates the last patch built date that is held in file
#            $LAST_PATCH_BUILT_DATE_FILE.
#
# Returns  : $TRUE or $FALSE
##############################################################################
UpdateLastPatchBuiltDate ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# assemble current date and time
LAST_PATCH_BUILT_DATE=`date +%d%m%Y%H%M%S`
(echo "${LAST_PATCH_BUILT_DATE}" 1>  ${LAST_PATCH_BUILT_DATE_FILE}) \
                                2>> ${RELEASE_LOG_FILE}
if [ $? -eq 0 ]
then
  return $TRUE
else
  return $FALSE
fi
}
##############################################################################
# Name     : ProcessLastPatchBuiltDate
#
# Overview : Processes the last patch built date that is held in
#            file $LAST_PATCH_BUILT_DATE_FILE.
#
# Returns  : $TRUE or $FALSE
#
# Notes    : 1 The function calls the following function:
#              o ValidateLastPatchBuiltDate
##############################################################################
ProcessLastPatchBuiltDate ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# check for file existence
if [ ! -f ${LAST_PATCH_BUILT_DATE_FILE} ]
```

```
then
   DisplayMessage E "${NO_LAST_PATCH_BUILT_DATE_FILE}"
   return $FALSE
fi
# read the date from the file
LAST_PATCH_BUILT_DATE=`cat  ${LAST_PATCH_BUILT_DATE_FILE}`
# verify the date format which should be <ddmmyyyyhhmiss>
if ! ValidateLastPatchBuiltDate
then
  return $FALSE
else
  return $TRUE
fi
}
##############################################################################
# Name     : FileToBeIncludedInPatch
#
# Overview : Establishes whether a specific source file is to be
#            included in the current patch release.
#
# Input    : Source Name
#
# Notes    1 The function uses the command istat to retrieve the
#            last modified date and time for the file. It compares
#            this with the date and time held in the variable
#            $LAST_PATCH_BUILD_DATE. The file is only included
#            in the current patch release if it was modified after
#            the date and time in $LAST_PATCH_BUILT_DATE.
##############################################################################
FileToBeIncludedInPatch ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# assign parameter
P_FILE_NAME="$1"
LAST_MODIFIED_DAY=`istat ${P_FILE_NAME} | grep "Last modified" | \
            cut -c15- | awk {'print $3'}`
LAST_MODIFIED_MON=`istat ${P_FILE_NAME} | grep "Last modified" | \
            cut -c15- | awk {'print $2'}`
LAST_MODIFIED_YEAR=`istat ${P_FILE_NAME} | grep "Last modified" | \
            cut -c15- | awk {'print $5'}`
LAST_MODIFIED_TIME=`istat ${P_FILE_NAME} | grep "Last modified" | \
            cut -c15- | awk {'print $4'} | sed s/':'//g`
# rework month
if [ "${LAST_MODIFIED_MON}" = "Jan" ]
then
  LAST_MODIFIED_MON="01"
elif [ "${LAST_MODIFIED_MON}" = "Feb" ]
then
  LAST_MODIFIED_MON="02"
```

```
elif [ "${LAST_MODIFIED_MON}" = "Mar" ]
then
  LAST_MODIFIED_MON="03"
elif [ "${LAST_MODIFIED_MON}" = "Apr" ]
then
  LAST_MODIFIED_MON="04"
elif [ "${LAST_MODIFIED_MON}" = "May" ]
then
  LAST_MODIFIED_MON="05"
elif [ "${LAST_MODIFIED_MON}" = "Jun" ]
then
  LAST_MODIFIED_MON="06"
elif [ "${LAST_MODIFIED_MON}" = "Jul" ]
then
  LAST_MODIFIED_MON="07"
elif [ "${LAST_MODIFIED_MON}" = "Aug" ]
then
  LAST_MODIFIED_MON="08"
elif [ "${LAST_MODIFIED_MON}" = "Sep" ]
then
  LAST_MODIFIED_MON="09"
elif [ "${LAST_MODIFIED_MON}" = "Oct" ]
then
  LAST_MODIFIED_MON="10"
elif [ "${LAST_MODIFIED_MON}" = "Nov" ]
then
  LAST_MODIFIED_MON="11"
elif [ "${LAST_MODIFIED_MON}" = "Dec" ]
then
  LAST_MODIFIED_MON="12"
fi
# build the $LAST_MODIFIED_DATE
LAST_MODIFIED_DATE="${LAST_MODIFIED_DAY}${LAST_MODIFIED_MON}$
➤  {LAST_MODIFIED_YEAR}${LAST_MODIFIED_TIME}"
# sort $LAST_MODIFIED_DATE against $LAST_PATCH_BUILT date
echo "$LAST_MODIFIED_DATE"    >  ${TEMP_FILE1}
echo "$LAST_PATCH_BUILT_DATE" >> ${TEMP_FILE1}
# The date is held in format <ddmmyyyyhhmiss> 01012000121060
# reversing the sort should put $LAST_MODIFIED_DATE at the top if
# the file was indeed modified after $LAST_PATCH_BUILT_DATE.
cat ${TEMP_FILE1} | sort -r -t: -k 1.5,1.8 -k 1.3,1.4 \
      -k 1.1,1.2 -k 1.9,1.10 -k 1.11,1.12 -k 1.13,1.14 > ${TEMP_FILE2}
# read the top line from file ${TEMP_FILE2}
LINE=`head -1 ${TEMP_FILE2}`
if [ "${LINE}" = "${LAST_MODIFIED_DATE}" ]
then
  # the file was modified after the last patch build
  return $TRUE
else
```

```
  # the file was not  modified after the last patch build
  return $FALSE
fi
}
##############################################################################
# Name     : ProcessRelease
#
# Overview : The function obtains the latest sources from each
#            of the source subdirectories and puts these in the
#            corresponding subdirectory under $RELEASE_DIR.
#
# Returns  : $TRUE or $FALSE
#
# Notes    1 The function calls the following functions:
#            - ProcessFileExtension
#            - ExtractSourceName
##############################################################################
ProcessRelease ()
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
echo "Processing  release" >> ${RELEASE_LOG_FILE}
FILE_NAME=""
PREV_FILE_NAME=""
CUR_SOURCE_NAME=""
PREV_SOURCE_NAME=""
DIR_NAME=""
RELEASE_SUB_DIR=""
CUR_FILE_NAME_SEG=""
PREV_FILE_NAME_SEG=""
SOURCE_NAME_LEN=0
FILE_NAME_LEN=0
# initialize released source list file
HEADER="List of Sources for Release ${RELEASE_NAME}"
FormatUnderscores   "$HEADER"
echo "       $HEADER   "          >  ${RELEASED_SOURCE_LIST}
echo "       $UNDERSCORE\n"       >> ${RELEASED_SOURCE_LIST}
echo "   Delete Unwanted Sources from the list"  >> \
      ${RELEASED_SOURCE_LIST}
echo "   by deleting the corresponding line\n"   >> \
      ${RELEASED_SOURCE_LIST}
# for patch release process  last patch built date
if [ "${RELEASE_TYPE}"  = "P" ]
then
  if ! ProcessLastPatchBuiltDate
  then
  return $FALSE
  else
  echo "**** Last Patch Built Date = ${LAST_PATCH_BUILT_DATE} ****\n"
➤  >> ${RELEASE_LOG_FILE}
  fi
```

```
fi
# store current directory
CURDIR=`pwd`
# process all source subdirectories, name of which are the same as
# those file extensions held in $FILE_EXTS
for SUB_DIR in $FILE_EXTS
do
  DIR_NAME="${SOURCE_DIR}/${SUB_DIR}"
  DIR_TO_PROCESS="${SOURCE_DIR}/${SUB_DIR}"
  DisplayMessage I  "${PROCESSING_DIR}"
  DisplayMessage I "${WORKING}" N
  RELEASE_SUB_DIR="${RELEASE_DIR}/${SUB_DIR}"
  HEADER="Processing Source Directory = ${DIR_TO_PROCESS}"
  FormatUnderscores   "$HEADER"
  echo "\n\n$HEADER"         >> ${RELEASE_LOG_FILE}
  echo "$UNDERSCORE\n"       >> ${RELEASE_LOG_FILE}
  #  make  releass subdirectory
  echo "Making release subdirectory ${RELEASE_SUB_DIR}" >> \
    ${RELEASE_LOG_FILE}
  (mkdir $RELEASE_SUB_DIR)  >  ${TEMP_FILE}  2>&1
  if [  $? -ne 0 ]
  then
    ERROR_MSG=`cat $TEMP_FILE |  head -1`
    DisplayMessage E "${OS_ERROR}"
    echo "Failed to make release subdirectory" >> ${RELEASE_LOG_FILE}
    echo "${ERROR_MSG}" >> ${RELEASE_LOG_FILE}
    rm ${TEMP_FILE}
    return $FALSE
  fi
  echo "Successfully made release subdirectory" >> ${RELEASE_LOG_FILE}
  # switch the appropriate source subdirectory
  cd ${DIR_TO_PROCESS} > /dev/null  2>&1
  if [ $? -ne 0 ]
  then
    echo "Source directory, ${DIR_TO_PROCESS} does not exist" >> \
         ${RELEASE_LOG_FILE}
    echo "Continuing with next source directory" >> ${RELEASE_LOG_FILE}
    continue
  fi
  # create a sorted list of files in a temporary file
  ls | sort -r > ${TEMP_FILE}
  # populate the array $FILE_NAME_ARRAY[] with the source names from
  # the temporary file.
  FILE_INDEX=0
  cat ${TEMP_FILE} | while read FILE_NAME
  do
    FILE_NAME_ARRAY[$FILE_INDEX]="${FILE_NAME}"
    FILE_INDEX=`expr $FILE_INDEX + 1`
  done
  FILE_NAME_ARRAY[$FILE_INDEX]="end_1.${SUB_DIR}"
```

```
FILE_INDEX=`expr $FILE_INDEX + 1`
FILE_NAME_ARRAY[$FILE_INDEX]=""
FILE_INDEX=0
PREV_SOURCE_NAME=""
PREV_VERSION_NO=0
while [ "${FILE_NAME_ARRAY[$FILE_INDEX]}" != "" ]
do
# perform sanity check
FILE_NAME="${FILE_NAME_ARRAY[$FILE_INDEX]}"
SOURCE_NAME="${FILE_NAME}"
if ! ProcessFileExtension "I"
then
  echo "\nBad file $FILE_NAME not copied to ${RELEASE_SUB_DIR}" >> \
       ${RELEASE_LOG_FILE}
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
fi
# extract file name without the version number
if ! ExtractSourceName "${FILE_NAME}"
then
  # this is not a sensible file - ignore it
  echo "\nBad file $FILE_NAME not copied to ${RELEASE_SUB_DIR}" >> \
       ${RELEASE_LOG_FILE}
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
fi
# check the numeric version number
CUR_VERSION_NO=`echo $FILE_NAME | sed 's/.*\_//' | cut -d'.' -f1`
if [ "${CUR_VERSION_NO}" = "" ]
then
  # invalid file - the version number is missing
  echo "\nBad file $FILE_NAME not copied to ${RELEASE_SUB_DIR}" >> \
       ${RELEASE_LOG_FILE}
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
elif ([ `expr $CUR_VERSION_NO + 0` -eq $CUR_VERSION_NO ]) \
     > /dev/null 2>&1
then
  # the version number is ok
  :
else
  # the version number is not a number
  echo "\nBad file $FILE_NAME not copied to ${RELEASE_SUB_DIR}" >> \
       ${RELEASE_LOG_FILE}
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
fi
CUR_SOURCE_NAME="${SOURCE_NAME_WITHOUT_VERSION}"
# check for first file for processing
if [ "${PREV_SOURCE_NAME}" = "" ]
```

```
then
  # processing the first file
  PREV_SOURCE_NAME="${CUR_SOURCE_NAME}"
  PREV_VERSION_NO=$CUR_VERSION_NO
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
fi
# compare the extracted source name
if [ "${CUR_SOURCE_NAME}" = "${PREV_SOURCE_NAME}" ]
then
  # processing the same file
  if [ $CUR_VERSION_NO -gt $PREV_VERSION_NO ]
  then
  PREV_VERSION_NO=$CUR_VERSION_NO
  fi
  FILE_INDEX=`expr $FILE_INDEX + 1`
  continue
fi
# found the latest source
# save it for later comparison
REQ_FILE="${PREV_SOURCE_NAME}_${PREV_VERSION_NO}.${SUB_DIR}"
PREV_SOURCE_NAME="${CUR_SOURCE_NAME}"
PREV_VERSION_NO=$CUR_VERSION_NO
# copy the latest source to the relevant release subdirectory,
# which depends on the release type
if [ "${RELEASE_TYPE}" = "P" ]
then
  if ! FileToBeIncludedInPatch "${REQ_FILE}"
  then
    FILE_INDEX=`expr $FILE_INDEX + 1`
    continue
  fi
fi
echo "\nCopying file $REQ_FILE to ${RELEASE_SUB_DIR}" >> \
     ${RELEASE_LOG_FILE}
DisplayMessage I "${COPYING_FILE}" N
sleep 2
cp $REQ_FILE ${RELEASE_SUB_DIR}/${REQ_FILE} 2> /dev/null
if [ $? -ne 0 ]
then
  echo "Failed to copy file $REQ_FILE to ${RELEASE_SUB_DIR}" >> \
       ${RELEASE_LOG_FILE}
  DisplayMessage E "${FILE_NOT_COPIED}"
else
  DisplayMessage I "${COPIED_FILE}" N
  sleep 2
  echo "Successfully copied file $REQ_FILE to ${RELEASE_SUB_DIR}" \
        >> ${RELEASE_LOG_FILE}
fi
# store the source name in a file for editing
```

```
  echo "${REQ_FILE}" >> ${RELEASED_SOURCE_LIST}
  FILE_INDEX=`expr $FILE_INDEX + 1`
  done
done
cd $CURDIR
DATETIME=`date "+%d/%m/%y at %H:%M:%S"`
# edit the released source list for this release
echo "Editing source list for patch release" >> ${RELEASE_LOG_FILE}
vi ${RELEASED_SOURCE_LIST}
echo "Edited source list for patch release" >> ${RELEASE_LOG_FILE}
# update last patch build date for patch release
if [ "${RELEASE_TYPE}" = "P" ]
then
  echo "Updating last patch build date" >> ${RELEASE_LOG_FILE}
  if UpdateLastPatchBuiltDate
  then
  echo "Updated last patch build date" >> ${RELEASE_LOG_FILE}
  else
  echo "Failed to update last patch build date" >> ${RELEASE_LOG_FILE}
  echo "Please update last patch build date manually" >> \
       ${RELEASE_LOG_FILE}
  fi
fi
# prepare release document for this this release
PrepareReleaseDocument
# store released source list in the database
if UpdateSmart "SRS"
then
  echo "\n***** Release completed on $DATETIME *****" >> \
       ${RELEASE_LOG_FILE}
   return $TRUE
else
  echo "\n***** Release not completed on $DATETIME *****" >> \
       ${RELEASE_LOG_FILE}
  return $FALSE
fi
}
##############################################################################
# Name     : InitializeReleaseLogFile
#
# Overview : Initializes the release log file.
#
# Notes    1. The release log file name is as follows:
#             fr_<$TODAY_<sequence>.log for full release
#             pr_<$TODAY_<sequence>.log for patch release
#
#          2. If more than one release falls on the same day,
#             the sequence number is incremented by 1.
##############################################################################
InitializeReleaseLogFile ()
```

```
{
trap "HandleInterrupt" $SIGINT $SIGTERM $SIGHUP $SIGTSTP
# get log file name
SEQUENCE=1
if [ "${RELEASE_TYPE}" = "F" ]
then
  FILE_NAME_COMPARATOR="${RELEASE_LOG_DIR}/fr_`echo $TODAY`_"
else
  FILE_NAME_COMPARATOR="${RELEASE_LOG_DIR}/pr_`echo $TODAY`_"
fi
# start comparing the file name
while true
do
  FILE_NAME=$FILE_NAME_COMPARATOR`echo $SEQUENCE`.log
  if [ -f $FILE_NAME ]
  then
    SEQUENCE=`expr $SEQUENCE + 1`
  else
    RELEASE_LOG_FILE=$FILE_NAME
    break
  fi
done
# initialize the log file
if [ "${RELEASE_TYPE}" = "F" ]
then
  HEADER="Full release on $DATETIME"
  FormatUnderscores "$HEADER"
  echo "       $HEADER"         >  ${RELEASE_LOG_FILE}
  echo "       $UNDERSCORE\n"   >> ${RELEASE_LOG_FILE}
else
  HEADER="Patch release on $DATETIME"
  FormatUnderscores "$HEADER"
  echo "       $HEADER"         >  ${RELEASE_LOG_FILE}
  echo "       $UNDERSCORE\n"   >> ${RELEASE_LOG_FILE}
fi
}
##############################################################################
# Name        : main
#
# Decsription : Invokes all other functions
#
# Input       : Release Type  F = Full  P = Patch
##############################################################################
main ()
{
RELEASE_TYPE="$1"
DefineModuleVariables
InitializeReleaseLogFile
if ! GetReleaseDirectoryName
then
```

```
  if ! ([ "${FUNCTION_ABORTED}" = "Y" ])
  then
    DisplayMessage E "${RELEASE_FAILED}"
  fi
  ProcessExit
  return $FALSE
fi
if ! CreateReleaseDirectory
then
  DisplayMessage E "${RELEASE_FAILED}"
  ProcessExit
  return $FALSE
fi
if ! GetReleaseDescription
then
  DisplayMessage E "${RELEASE_FAILED}"
  ProcessExit
  return $FALSE
fi
if ! ProcessRelease
then
  DisplayMessage E "${RELEASE_FAILED}"
  ProcessExit
  return $FALSE
fi
DisplayMessage I "${RELEASE_SUCCEEDED}"
ProcessExit
return $TRUE
}
# execute main
# invoke main with command line argument
main $1
```

SAMPLE CHECKIN/CHECKOUT LOG FILE

```
                        CHECKIN/CHECKOUT LOG
                       ======================

DAY       TIME      DEVELOPER  MODULE             COMMENT
-----------------------------------------------------------------
20/05/96  17:09:48  azaman     execsu.c           CHECKED IN NEW SOURCE
24/05/96  16:16:42  lnsmgr     debug_deamon.pc    CHECKED IN NEW SOURCE
21/04/97  11:12:34  azaman     execsu_2.c         CHECKED OUT
21/04/97  11:28:56  azaman     execsu_2.c         CHECKED IN

21/06/96  17:41:58  root       debug_deamon_2.pc  CHECKED OUT
21/06/96  17:44:35  root       debug_deamon_2.pc  LOCKED SOURCE RELEASE
```

SAMPLE SOURCE-FILE-TO-SOURCE-NAME MAPPING FILE

```
                SOURCE FILE TO SOURCE NAME MAPPING
                ==================================

File name     Source Name                  Description
-------------------------------------------------------------------
execsu.c      execsu.c                     Superuser execution, used
                                           by the utility ssccars.sh.
gem.proc      GEN_GET_ERROR_MESSAGE        Database procedure to get
                                           error message from a table.
pllrc.trig    PROCESS_LNS_LLP_ROW_CHANGE   Database tigger on LNS_LLP
                                           table.
pmrc.trig     PROCESS_MI_ROW_CHANGE        Database trigger on MI table.
mdd.view      MI_DERIVED_DATA              Database view.
```

SAMPLE LIST OF CHECKOUT SOURCES

```
FOLLOWING SOURCES ARE CURRENTLY CHECKED OUT FOR MODIFICATION
===========================================================

MONTH     DAY      TIME     SOURCE
=====     ===      ====     ======
Jul        09      15:29    ssccars_lib_3.sh
Jul        09      15:28    br_3.sh
Jul        09      11:23    iron_10.sh
Jul        09      10:15    ssccars_2.sh
Jul        01      14:44    pmdca_10.trig
Jul        01      14:44    aot_6.proc
```

CORRECTION

One of the modules of this utility, **iron.sh** (‘Install Release Over Network’), has been omitted from this article. On reflection it was decided that this module is too installation-specific and does not lend itself easily to being customized to other installations’ needs. Consequently it’s necessary to modify the menu options available in the main module (**ssccars.sh**) to remove option ‘70’ (Install release) and ‘75’ (View installed release log file).

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

Checking for Unix users without a password

Security is an important issue for any company, whether there are only a handful of employees or hundreds. While access to the system *can* be controlled by means of unique user login names, unless users protect their logins by setting a password, access to the system is compromised.

The simple script in this article checks whether a user has a password in the */etc/shadow* file, prompting you to enter one if they don't. For this reason, the script can only be run by the *root* user.

You can enter either the user's login name (eg 'npinney') or 'all' to check every user in the */etc/passwd* file.

The script should be run periodically – say once a week – to ensure that all users have set a password for their login.

A SAMPLE OUTPUT OF THE SCRIPT

```
This script checks whether a given user has a password or not
and allows you to set one if they haven't.

Enter user login name or 'all' [q, ?]:  all

It's ok, root HAS got a password.
It's ok, bin HAS got a password.
It's ok, sys HAS got a password.
It's ok, jbevilac HAS got a password.
It's ok, agreenland HAS got a password.

WARNING: User jtiller does NOT have a password!

Do you want to set a password now ? (y/n/q): n

Ok, but remember to set a password soon!

WARNING: User npinney does NOT have a password!

Do you want to set a password now ? (y/n/q): y

OK, Enter the new password now:
New password: test
Re-enter new password: test

OK, New password has been set.
```

Please note the use of the continuation character, '➤', in the listing below to indicate a formatting line break that's not present in the original source code.

CHECK PASSWORD

```
#!/bin/ksh
###############################################################################
#
#  Script to check whether a user has a password or not by checking
#  for an entry in the /etc/shadow file.
#
#  You must enter the user's login name or 'all' to check every
#  user in the /etc/passwd file.
#
#  If the user does not have a password, the program prompts you
#  to enter one.
#
#  Written by Nicola Pinney - January 2000
#
###############################################################################

#*****************************************************************
#       Function to check the User Id (must be run as 'root')
#
#*****************************************************************

checkid()
{
if [ -x /usr/bin/id ]
then
   eval `id | sed 's/[^a-z0-9=].*//'`
   if [ "${uid:=0}" -ne 0 ]
   then
      echo "\n$0: Only root can run this script.\n"
      exit 2
   fi
fi
}

#*****************************************************************
#
#                     MAIN PROGRAM STARTS HERE
#
#*****************************************************************

# Call function to check User Id - you must be 'root' to run the script
checkid

echo "\nThis script checks whether a given user has a password "
```

```
echo "and allows you to set one if they haven't."

while true
do
   echo "\nEnter user login name or 'all' [q, ?]: \c"

   read ID

   if [ "$ID" = "q" ]
   then
      echo "\n"
      exit 2
   fi

   # If the user needs help then show them a list of valid users
   if [ "$ID" = "?" ]
   then
      echo "\nHere is a list of valid users:\n"
      echo `cut -f1 -d: /etc/passwd`
      continue
   fi

   # If <CR> is pressed, re-prompt to enter details
   if [ -z "$ID" ]
   then
      continue
   fi

   if [ "$ID" = "all"  -o "$ID" = "ALL" ]
   then
      cat /etc/passwd | cut -f1 -d ":" > /tmp/usernames.$$
      break
   fi

   # Search for the User name from the first character position
   # of the passwd file.
   grep "^$ID:" /etc/passwd > /tmp/usernames.$$

   if [ $? -ne 0 ]
   then
      echo "\nThat's not a valid user ... try again.\n"
      continue
   else
      break
   fi

done

# Check that the second field of the shadow file contains something
# as this is the password. If it is blank then we can assume that
```

```
# the user does not have a password set so we should prompt them to
# set one.

for ID in `cat /tmp/usernames.$$ | cut -f1 -d ":"`
do
   user_line=`cat /etc/shadow | grep $ID | cut -f1 -d ";"`
   password=`echo $user_line | cut -f2 -d ":"`
   if [ "$password" = "" ]
   then
      echo $ID >> /tmp/nopass.$$
   else
      echo "\nIt's ok, $ID HAS got a password."
   fi
done

# If there are entries in /tmp/nopass.$$ then some users do not have
# passwords so we should prompt to set one now.

if [ -f "/tmp/nopass.$$" ]
then
   for ID in `cat /tmp/nopass.$$`
   do
      echo "\nWARNING: User $ID does NOT have a password!"
      while true
      do
         echo "\nDo you want to set a password now ? (y/n/q): \c"
         read ans
         case $ans in
         n*|N*)  echo "\nOk, but remember to set a password soon!\n"
            break
            ;;
         y*|Y*)  echo "\nOK, Enter the new password now:"
            passwd $ID
            if [ "$?" != "0" ]
            then
               echo "\nThere was a problem setting the password -
                      ➤  please try again!"
               continue
            else
               echo "\nOK, New password has been set."
            fi
            break
            ;;
         q*|Q*)  echo "\n"
            exit 2
            ;;
         *)      echo "\nPlease answer 'y', 'n' or 'q'\n"
            continue
            ;;
         esac
```

```
      done
   done
fi

# Clear up temporary files we have been using
rm /tmp/usernames.$$ 2> /dev/null
rm /tmp/nopass.$$ 2> /dev/null
```

Nicola Pinney
Unix System Administrator (UK)

Viewing the contents of a tar file

This article presents a script (**lookinside.sh**) that shows the contents of a compressed **tar** file. It's a small, but very useful, script that I'd like to share with *AIX Update* readers.

LOOKINSIDE.SH

```
#!/bin/ksh
# lookinside.sh - Shows the contents of a compressed tar file.
#
# By Adnan Akbas Feb'97.
#
usage='lookinside.sh filename'

if (( $# != 1 )) ; then
  print "Usage: $usage" >&2
  exit 2
fi
fz=${1:?"Filename not given!"}
if [[ ! -r $fz ]] ; then
  print "Cannot read file!" >&2
  exit 3
fi
lfz=~/tmp/$$.Z
lf=${lfz%.Z}
/bin/cp -f $fz $lfz
let ret=$?
if (( ret != 0 )) ; then
  print "copy error ($ret)" >&2
  exit 1
fi
```



```
/bin/uncompress -f $lf
let ret=$?
if (( ret != 0 )) ; then
  print "uncompress error ($ret)" >&2
  exit 1
fi
/bin/tar -tvf $lf
let ret=$?
if (( ret != 0 )) ; then
  print "tar error ($ret)" >&2
  exit 1
fi
/bin/rm -f $lf
exit 0
```

Adnan Akbas
System Programmer
Pamukbank (Turkey)

Defining variables in /etc/environment

When defining variables in */etc/environment*, it's important not to leave any trailing spaces at the end of entries, as this will corrupt Customized Device object file ('CustDev'). Consider the following two entries:

Example 1:

```
ODMDIR=/etc/objrepos··
```

('·' represents a space)

Example 2:

```
ODMDIR=/etc/objrepos
```

Example1 corrupts the CustDev file, but example 2 is OK.

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

Check Point Software has announced that its FireWall-1 Internet security product is to be integrated with IBM HACMP (High-Availability Cluster Multi Processing). The integrated system has achieved OPSEC certification, which basically means that the two have been proven to work together. (OPSEC is Check Point's Open Platform for SECurity, an integration and interoperability platform for third-party products and services.) In return, FireWall-1 has been designated an IBM 'ClusterProven solution'. HACMP runs on current RS/6000 servers, automatically engaging alternative hardware and software in case of system hardware or software failures.

For further information contact:
Check Point Software Technologies, Three Lagoon Drive, Suite 400, Redwood City, CA 94065, USA
Tel: +1 650 628 2000
Fax: +1 650 654 4233
Web: http://www.checkpoint.com

Check Point Software Technologies, Suite 5b, Enterprise House, Vision Park, Chivers Way, Histon, Cambridge CB4 9ZR, UK
Tel: +44 1223 713600
Fax: +44 1223 236847

* * *

EMC has announced the Fibre Channel Interface Kit for AIX, which enables PCI-based RS/6000s to connect via a switched fibre channel to EMC's Connectrix Fibre Channel director. The Interface Kit for AIX is compatible with Brocade's SilkWorm 2000 family of Fibre Channel switches. Prices and availability weren't announced.

For further information contact:
EMC Corp, 171 South Street, Hopkinton, MA 01748, USA
Tel: +1 508 435 1000
Fax: +1 508 497 6915
Web: http://www.emc.com

EMC, EMC House, 7 The Parks, Newton le Willows, Merseyside, Haydock, WA12 0JQ, UK
Tel : +44 1942 275511

* * *

The Sun-Netscape Alliance is to expand the number of iPlanet Internet infrastructure and e-commerce applications that are ported to AIX. Products covered will be marketed by both firms and sold by the Alliance.

Currently, iPlanet Web Server, Messaging Server, and Directory Server products are available on AIX. In the first half of this year, AIX versions of iPlanet Application Server, Calendar Server, and Certificate Management System will roll out, as well as applications for corporate procurement, billing, EDI, and advanced merchandizing.

For further information contact:
Netscape Communications, 501 E Middlefield Road, Mountain View, CA 94043, USA
Tel: +1 650 254 1900
Web: http://www.netscape.com/ecxpert

Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA
Tel: +1 650 960 1300
Web: http://www.sun.com