88

AIX update

February 2003

In this issue

AIX Update

Published by

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

North American office

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

Subscriptions and back-issues

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

AIX Update on-line

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

Editors

Trevor Eddolls

Disclaimer

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

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Go to the end of the line

Given that documents and text files can move so easily from one system to another, you would think that some common format could be devised for text data. Unfortunately, moving a Word document or a spreadsheet from a Windows environment to a Unix environment leaves you with a document or spreadsheet that is very hard to read in the Unix world.

How about the humble text file? Surely a file containing nothing but ASCII characters must be portable across multiple systems? At least, you would think so!

The ‘gotcha’ in moving text files between systems is most apparent in moving text from an MS-DOS or Windows environment to a Unix environment. This problem comes up most often in these systems because these operating systems are so common. Of course, DOS is dead. Just don’t tell the people running thousands of vertical market MS-DOS applications that were never ported to Windows because the software vendor went out of business trying to port to Windows.

There are text files to move between these systems, but you also have code for many programs (especially those written in C and C++) that are frequently portable across these systems.

“Hey, Bloggs, didn’t you write a utility to unscramble the gormly-googles? I bet you could move that code to Unix and it would compile and run there.”

Yes, it probably will run, but the text files that contain the source code for the ‘gormly-google’ unscrambler will use a different end-of-line marker in an MS-DOS or Windows world from what it will need in a Unix world.

And the code might not compile on Unix because it can’t handle the end-of-line markers that came over from the Windows version. Nothing to do with the code itself, just the text format that the code is in.

Differences in text formats between Unix and the Windows world become very apparent when you are moving large quantities of source code around.

Terms like carriage return, line feed, and new line are frequently bandied about in relationship to text files and printing formats, but you might not know exactly what they are, and how they relate to text files.

Back when dinosaurs ruled the earth, the primary method to output information from a computer was a printer or teletypewriter. (As an aside here, the Unix tty designation for a terminal device is an abbreviation for teletypewriter.)

One of the important factors in controlling output to the printer or teletype was the subject of carriage control. Printers and teletypes had a platen or cylinder, which you can still see today on dot matrix and other impact printers (or on typewriters in museums). Paper was fed through the printer by rolling it around the platen or, in the case of pin-feed paper, by feeding it through a tractor feeder. The tractor feeder and the platen were both carriage devices and their primary function was to carry (carriage/carry – get it?) the paper in a precision manner and position it in front of the print head.

The mechanism that moved the print head back and forth was also part of the ‘carriage’ and the carriage was responsible for the amount of space between each printed line and where the print head was positioned before each line was printed.

Early printers attached to IBM’s big iron expected to receive two (sometimes three) bytes of information at the start of each line that contained information on where to print the following line of characters. These were called carriage control commands. Carriage control commands went from the very simple }‘PRINT AFTER ADVANCING 1 LINE’ to the slightly more complex ‘PRINT BEFORE ADVANCING 3 VERTICAL TABS’.

Because carriage control allowed control over both vertical and horizontal movement on the printed page, two, now well known, font decorations could be created. By printing a line and then

printing it again ‘AFTER ADVANCING 0 LINES’ you could print the same line twice creating bold type. By printing a line and then printing a line of underscores and spaces ‘AFTER ADVANCING 0 LINES’ you could print a line containing underlining.

You could also print a line or a character, followed by backspace, and then print a hyphen or a slash for a strike-through, although it was used much less often.

IBM printers were frequently fed by large bundles of wires that carried both carriage control and printing signals. Then along came smaller printers that were connected by parallel printer or serial port connections. There were no separate wires to handle carriage control information, and it became necessary to send bytes of ‘printer control’ information as part of the data stream.

Character sets that were designed for this type of device, such as IBM’s EBCDIC and the ASCII set, included additional characters that could instruct the printer on how to behave – in other words, control characters. The ASCII character set includes ‘control’ characters that can be used to control the behaviour of peripheral devices.

ASCII (American Standard Code for Information Interchange) uses 128 numbers to represent the characters of the alphabet, the digits, punctuation characters, and some special characters that are used to control printers, terminals, and other computer devices. The 128 values are numbered beginning with zero, so the numbers used range from 0 to 127.

The ASCII character set includes all the digits, the upper and lower case letters of the alphabet, and punctuation characters. All the printable characters (letters, digits, and punctuation) have values between 32 and 126. The values 0 to 31 and 127 are used for control characters.

Figure 1 is a brief ASCII chart with the decimal value of each entry, and its ASCII name or character. The non-printing characters are given with their names. You might be familiar with some of these.

0	NUL	32	SP	64	@	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	[af]	126	~
31	US	63	?	95	_	127	DEL

Figure 1: ASCII chart with decimal values

Many of the non-printable characters are used for communications protocols.

A control character is a single character that can be sent to a computer device, such as a printer or monitor, that controls the behaviour of the device (rather than printing an actual character). For example, the value 84 is an upper case ‘T’ and if sent to a

printer or terminal would cause the device to print or display a ‘T’. But a 13 (CR) is a carriage return. This value sent to a printer causes the print head to return to column 1. A carriage return (CR) also is sometimes sent by the *Return* or *Enter* key on the keyboard. Internally Unix usually translates this to a 10, a line feed (LF). The line feed value (LF) is used to move a printer or terminal up one line. Value 7 (BEL), when sent by the computer to the terminal, usually causes a beep or rings an alarm.

Some of the other interesting control characters are:

- HT (Horizontal Tab, or just plain tab in every day parlance), value 9. This character, sent to a printer or a screen, causes the cursor or print head to advance to the next defined print column.
- SO (Shift Out) and SI (Shift In), values 14 and 15, were frequently used in printer control. Many printers are set up with two built-in fonts. Sending an SI causes the printer to shift to the second font. Sending an SO causes the printer to shift back to the first font.

The values from 32 through 126 are printable characters. Value 32 (SP) is a space. Whether a space is actually a ‘printable’ character is a debatable issue. A space does not usually put ink on the paper. Instead it places a character containing no image. Most printers create this by simply advancing the print head one position.

Several of the other characters in the range below 32 are used extensively in telecommunications. For example 02 and 03, STX (start of transmission) and ETX (end of transmission), are often used at the start and end of a block of transmitted information. 06 and 21, ACK and NAK, are often used by a receiving computer to signal acknowledgement (ACK for well received) or a negative acknowledgement (NAK for not well received, please re-transmit).

Finally, we come to the use of control characters in text files. Control characters are used inside text files to indicate the end of a line. Unix uses a single LF, line feed, (10) character. MS-DOS

and Windows use a combination of CR, carriage return, (13), and LF (10). Moving a text file back and forth between these two systems without translating the end of line marker causes some unusual results. For example, the MS-DOS Edit utility is smart enough to recognize a file that has only a line feed for an end-of-line marker and displays it correctly, but the Windows Notepad utility is not. Notepad displays an 'untranslatable' control character as a thick black vertical bar. In Notepad this looks like a black box. In the following listings the black box is show as a left and right square bracket, [].

Start with a simple text file in Unix:

```
These are the times
that try men's souls.
The Metropolitan Transit Authority,
better known as the MTA
etc.
```

The same file in Notepad cannot figure out where the lines end:

```
These are the times[]that try men's souls.[]The Metropolitan Transit
Authority,[]better known as the MTA[]etc.
```

In the reverse case, a Windows text file has too many control characters for vi. The extra carriage return shows up as a control-M (^M) in the vi display.

If the following had been created in Notepad and transferred to AIX, it might look like this in vi:

```
These are the times^M
that try men's souls.^M
The Metropolitan Transit Authority,^M
better known as the MTA^M
etc.^M
```

Many Unix/Windows transfer utilities include a switch that can be set to indicate that a text file is being transferred, and the resulting file has its end of line character(s) translated. Some utilities have text translation as the default, and you must set a switch to suppress the translation when you are transferring binary files.

Movement of files in volume is done by combining and compacting the files using one of the versions of zip, gzip, tar, or what have

you, and the resulting file must be transferred as a binary file. The individual members of an archive lose any indication of whether they are text or binary files, and the entire archived and/or compressed file must be transferred as if it were a binary file.

At the other end, the unpacked files now suffer from the end-of-line marker ambiguity.

The extra carriage return can be removed or inserted with two simple scripts. I use scripts here so that you can save and reuse them. The first one takes two command arguments, the Unix file name and the MS-DOS/Windows file name. It adds a carriage return to a Unix text file and outputs it under the DOS text file name so that it can be transferred to DOS. To type this in using vi, you must be able to type an actual Control-M. When you have to type the ^M, type control-V then control-M. The control-V causes the next character to be inserted as a literal control character. After you have saved it as lf2crlf, change its mode to allow execution (chmod a+x lf2crlf).

```
# lf2crlf
# adds an extra carriage return to a Unix
# text file so that the end of line matches
# the Windows/DOS convention

usage()
{
      echo "usage: lf2crlf unix.txt dos.txt
      exit
}

if [ $# != 2 ]
then
      usage
fi

sed 's/$/^M/g' <$1 >$2
```

The second script reverses the process and strips an extra CR out for text files coming from Windows to Unix. There is an additional ‘gotcha’ in MS-DOS files. Some DOS editors and utilities append a Control-Z (26 or ASCII SUB) to the end of a text file. This will display in vi as ^Z at the end of a file. This script also removes that character. Note that the single quoted set of

commands start on one line and end on the next. Use control-V, control-M to create the ^M and control-V, control-Z to create the ^Z.

```
# crlf2lf
# removes an extra carriage return in a DOS/Windows
# text file so that the end of line matches
# the Unix convention.
# Also removes a control-Z at end of file
usage()
{
      echo "usage: crlf2lf dos.txt unix.txt
      exit
}
if [ $# != 2 ]
then
      usage
fi
sed 's/^M//g
s/^Z//g' <$1 >$2
```

Just as a final note, DEC systems used a carriage return only as an end-of-record marker. Porting source code between Windows, Unix and VAX produced an intimate knowledge of record and file terminator characters.

Mo Budlong
Middleware and Data Translation Specialist
King Computer Services (USA)

Network back-up manager – part 2

This month we publish the rest of the code for a network back-up manager.

```
  } else {
    if ( compress == 1 ) {
      if ( strcmp(backup_host,host_name) == 0 ) {
        childpid1=fork();
        if ( childpid1 == -1 ) {
          printf("Problem in fork process.\n");
          exit_program(115);
        }
```

```
        if ( childpid1 == Ø ) {
          sprintf(sys_cmd,"compress -c %s/%s > %s
",path_name,file_name,pipe1);
          if ( system(sys_cmd) != Ø ) {
            printf("Problem in compress...\n");
            exit_program(116);
          }
          exit(Ø);
        } else
          sprintf(sys_cmd,"cp %s %s/%s!at!%s!%s.V%ld.Z  > /dev/
null",pipe1,backup_path_name,path_name_with,host_name,file_name,file_version);
      } else {
        sprintf(sys_cmd,"compress -c %s/%s > %s
",path_name,file_name,tempfilename);
        if ( system(sys_cmd) != Ø ) {
          printf("Problem in compressing file:%s/%s
...\n",path_name,file_name);
          exit_program(117);
        }
        sprintf(sys_cmd,"rcp %s %s@%s:%s/%s!at!%s!%s.V%ld.Z  > /dev/
null",tempfilename,username,backup_host,backup_path_name,
path_name_with,host_name,file_name,file_version);
      }
    } else {
      if ( strcmp(backup_host,host_name) == Ø )
        sprintf(sys_cmd,"cp %s/%s %s/%s!at!%s!%s.V%ld  > /dev/
null",path_name,file_name,backup_path_name,path_name_with,
host_name,file_name,file_version);
      else
        sprintf(sys_cmd,"rcp %s/%s %s@%s:%s/%s!at!%s!%s.V%ld  > /dev/
null",path_name,file_name,username,backup_host,backup_path_name,path_name_with,
host_name,file_name,file_version);
    }
  }
  if ( system(sys_cmd) != Ø ) {
    printf("Problem in copying file:%s ...\n",file_name);
    exit_program(118);
  }
}
void nbm_copy()
{
  long file_size;
  get_random_number();
  if (gethostname(host_name,1Ø)!=Ø ) {
    printf("Problem in getting host name...\n");
    exit_program(119);
  }
  exec sql connect to nbmdb;
  if ( sqlca.sqlcode < Ø ) sql_error(24);
  if ( strcmp(path_name,"")==Ø)
```

```
  {
    if (getwd(path_name)==NULL){
      printf("Problem getting current directory...\n");
      exit_program(120);
    }
  }
  sprintf(file_to_be_copied,"%s/%s",path_name,file_name);
  file_size=get_file_size();
  get_a_path(file_size);
  file_version=get_version_number(path_name,file_name);

  exec sql insert into BACKUP_FILES
     values(:host_name,:path_name,:file_name,:path_no,:file_version,current
timestamp,:retpd,:is_dir,:compress);
  do_copy();
  printf("%s/%s version:%ld is successfully backed up to %s/
%s!at!%s!%s.V%ld on host
%s.\n",path_name,file_name,file_version,backup_path_name,path_name_with,
backup_host,file_name,file_version,backup_host);
}
int main(int argc,char **argv)
{
  int copt=0;
  strcpy(preferred_host_name,"");
  while(1) {
    copt=getopt(argc,argv,"p:f:r:s:g:ncih?");
    if (copt==-1) break;
    switch(copt) {
      case 'p' :
             strcpy(path_name,optarg);
             break;
      case 'f' :
             strcpy(file_name,optarg);
             break;
      case 'r' :
             retpd=atoi(optarg);
             break;
      case 's' :
             strcpy(preferred_host_name,optarg);
             break;
      case 'g' :
             preferred_group_num=atoi(optarg);
             break;
      case 'n' :
             not_on_this_host=1;
             break;
      case 'i' :
             interactive_mode=1;
             break;
      case 'c' :
```

```
                compress=1;
                break;
        case 'h' :
        case '?' :
                nbm_help();
                exit(2);
                break;
        otherwise :
                nbm_help();
                exit(3);
                break;
      }
    }
    if ( interactive_mode==1 ) {
      printf("Path Name (-p) - Enter Ø for current directory - :");
      scanf("%s",path_name);
      printf("\nFile Name (-f) :");
      scanf("%s",file_name);
      printf("\nRetention Period (-r) :");
      scanf("%ld",&retpd);
      printf("\nPreferred Host Name (-s) - Enter Ø for not preferring -
:");
      scanf("%s",preferred_host_name);
      printf("\npreferred_group_num (-g) - Enter Ø for not preferring -
:");
      scanf("%ld",&preferred_group_num);
      printf("\nØ:On the same host, 1:on another host (-n) =");
      scanf("%ld",&not_on_this_host);
      printf("\nØ:NOT compressed, 1:compressed (-c) =");
      scanf("%ld",&compress);
    }
    if ( strcmp(path_name,"Ø")==Ø ) strcpy(path_name,"");
    if ( strcmp(preferred_host_name,"Ø")==Ø )
strcpy(preferred_host_name,"");
    if ( strcmp(file_name,"")== Ø ) {
      printf("File name must be entered...\n");
      exit(2ØØ);
    }
    if ( retpd == Ø ) {
      printf("Retention Period must be other than Ø...\n");
      exit(2Ø1);
    }
    nbm_copy();
    exec sql commit;
    exit_program(Ø);
}
```

NBMRESTORE.SQC

```
#include <stdio.h>
```

```
#include <sqlca.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include <errno.h>
int interactive_mode=0;
char backup_file_name[400],host_name[10];
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  char file_name[200];
  char path_name[200];
  char backup_path_name[200];
  char backup_host[10];
  char file_host_name[10];
  char username[20];
  char after_time[26];
  char before_time[26];
  long version;
  long compressed;
  long restore_version=999;
  long ret_pd;
  char cre_date[26];
  long file_dir;
EXEC SQL END DECLARE SECTION;
void nbm_help()
{
  printf("Usage:\n");
  printf("  nbmrestore -p path_name -f file_name -v version -a YYYY-MM-
DD-HH.MI.SS.SSSSSS \n");
  printf("         -b YYYY-MM-DD-HH.MI.SS.SSSSSS -i ( interactive mode )
\n");
}
void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit(1);
}
void do_restore()
{
  char sys_cmd[1000],path_name_with[1000],*getenv(),*tempdir;
  int ind;
  for (ind=0;ind<=strlen(path_name);ind++)
    if ( path_name[ind] == '/' ) path_name_with[ind]='!'; else
path_name_with[ind]=path_name[ind];
  if (gethostname(host_name,10)!=0 ) {
    printf("Problem in getting host name...\n");
    exit(2);
```

```
  }
  tempdir=getenv("NBM_TEMPDIR");
  if ( strcmp(backup_host,host_name) == Ø )
    sprintf(sys_cmd,"cp %s/%s!at!%s!%s %s/%s  > /dev/
null",backup_path_name,path_name_with,file_host_name,backup_file_name,
tempdir,backup_file_name);
  else
    sprintf(sys_cmd,"rcp %s@%s:%s/%s!at!%s!%s %s/%s  > /dev/
null",username,backup_host,backup_path_name,path_name_with,file_host_name,
backup_file_name,tempdir,backup_file_name);
  if ( system(sys_cmd) != Ø ) {
    printf("Problem in copying file:%s ...\n",backup_file_name);
    exit(3);
  }
  printf("\n%s is successfully restored to
%s.\n",backup_file_name,tempdir);
}
int confirm_restore()
{
  char is_this;
      if ( file_dir == 1 || compressed == 1 )
        sprintf(backup_file_name,"%s.V%ld.Z",file_name,restore_version);
      else
        sprintf(backup_file_name,"%s.V%ld",file_name,restore_version);
      printf("Backup file %s stored on %s:%s taken at %s ( RETPD=%ld,
CURRENT VERSION=%ld, FILE TYPE=%s
)\n",backup_file_name,backup_host,backup_path_name,cre_date,ret_pd,version,(
file_dir == Ø ) ? "FILE":"DIR");
      printf("Is this the one you want to restore ( y/n) ?");
      scanf("%2c",&is_this);
      if ( is_this == 'y' || is_this == 'Y' ) {
        do_restore();
        return 1;
      }
      printf("\n\n");
  return Ø;
}
void nbm_restore()
{
  exec sql connect to nbmdb;
  if ( sqlca.sqlcode < Ø ) sql_error(1);
  if ( restore_version != 999 ) {
    exec sql select a.HOST,a.PATH_NAME, a.FILE_NAME, b.HOST, b.PATH,
a.VERSION,
                  c.VERSION-1, DATE_TAKEN, RETPD,
                  FILE_TYPE, COMPRESSED, rtrým(USERNAME)
             into :file_host_name,:path_name, :file_name, :backup_host,
                  :backup_path_name,:restore_version, :version,
:cre_date, :ret_pd,
                  :file_dir, :compressed, :username
```

```
            from BACKUP_FILES a,BACKUP_PATHS b,BACKUP_VERSIONS c
            where a.PATH_NAME = :path_name and
                  a.FILE_NAME = :file_name and
                  a.PATH_NAME = c.PATH_NAME and
                  a.FILE_NAME = c.FILE_NAME and
                  PATH_NO = BACKUP_PATH_NO and
                  a.VERSION = c.VERSION-1+:restore_version;
    if ( sqlca.sqlcode < 0 ) sql_error(2);
    if ( sqlca.sqlcode==100 ) {
      printf("No backup found for file %s/%s...\n",path_name,file_name);
      exit(4);
    }
    confirm_restore();
  }
  if ( strcmp(after_time,"") != 0 ) {

    exec sql declare crs1 cursor for
            select a.HOST,a.PATH_NAME, a.FILE_NAME, b.HOST, b.PATH,
a.VERSION,
                   c.VERSION-1, DATE_TAKEN, RETPD,
                   FILE_TYPE, COMPRESSED, rtrým(USERNAME)
            from BACKUP_FILES a,BACKUP_PATHS b,BACKUP_VERSIONS c
            where a.PATH_NAME = :path_name and
                  a.FILE_NAME = :file_name and
                  a.PATH_NAME = c.PATH_NAME and
                  a.FILE_NAME = c.FILE_NAME and
                  PATH_NO = BACKUP_PATH_NO and
                  DATE_TAKEN > :after_time
            order by a.DATE_TAKEN ;
    if ( sqlca.sqlcode < 0 ) sql_error(3);
    exec sql open crs1;
    if ( sqlca.sqlcode < 0 ) sql_error(4);
    exec sql fetch crs1 into :file_host_name,:path_name, :file_name,
:backup_host,
                             :backup_path_name,:restore_version,
:version, :cre_date,
                             :ret_pd,:file_dir, :compressed, :username;
    if ( sqlca.sqlcode < 0 ) sql_error(5);
    if ( sqlca.sqlcode==100 ) {
      printf("No backup found for file %s/%s...\n",path_name,file_name);
      exit(5);
    }
    while (sqlca.sqlcode!=100 ) {
      if ( confirm_restore() == 1 ) break;
      exec sql fetch crs1 into :file_host_name,:path_name, :file_name,
:backup_host,
                               :backup_path_name,:restore_version,
:version,
                               :cre_date,:ret_pd,:file_dir, :compressed,
:username;
```

```
      if ( sqlca.sqlcode < 0 ) sql_error(6);
    }
    exec sql close crs1;
    if ( sqlca.sqlcode < 0 ) sql_error(7);
  }
  if ( strcmp(before_time,"") != 0 ) {
    exec sql declare crs2 cursor for
             select a.HOST,a.PATH_NAME, a.FILE_NAME, b.HOST, b.PATH,
a.VERSION,
                    c.VERSION-1, DATE_TAKEN, RETPD,
                    FILE_TYPE, COMPRESSED, rtrým(USERNAME)
             from BACKUP_FILES a,BACKUP_PATHS b,BACKUP_VERSIONS c
             where a.PATH_NAME = :path_name and
                   a.FILE_NAME = :file_name and
                   a.PATH_NAME = c.PATH_NAME and
                   a.FILE_NAME = c.FILE_NAME and
                   PATH_NO = BACKUP_PATH_NO and
                   DATE_TAKEN < :before_time
             order by a.DATE_TAKEN DESC ;
    if ( sqlca.sqlcode < 0 ) sql_error(8);
    exec sql open crs2;
    if ( sqlca.sqlcode < 0 ) sql_error(9);
    exec sql fetch crs2 into :file_host_name,:path_name, :file_name,
:backup_host,
                             :backup_path_name,:restore_version,
:version, :cre_date,
                             :ret_pd,:file_dir, :compressed, :username;
    if ( sqlca.sqlcode < 0 ) sql_error(10);
    if ( sqlca.sqlcode==100 ) {
      printf("No backup found for file %s/%s...\n",path_name,file_name);
      exit(6);
    }
    while (sqlca.sqlcode!=100 ) {
      if ( confirm_restore() == 1 ) break;
      exec sql fetch crs2 into :file_host_name,:path_name, :file_name,
:backup_host,
                               :backup_path_name,:restore_version,
:version,
                               :cre_date,:ret_pd,:file_dir, :compressed,
:username;
      if ( sqlca.sqlcode < 0 ) sql_error(11);
    }
    exec sql close crs2;
    if ( sqlca.sqlcode < 0 ) sql_error(12);
  }
}
int main(int argc,char **argv)
{
  int copt=0;
  strcpy(after_time,"");
```

```
  strcpy(before_time,"");
  while(1) {
    copt=getopt(argc,argv,"p:f:v:a:b:ih?");
    if (copt==-1) break;
    switch(copt) {
      case 'p' :
             strcpy(path_name,optarg);
             break;
      case 'f' :
             strcpy(file_name,optarg);
             break;
      case 'v' :
             restore_version=atoi(optarg);
             break;
      case 'a' :
             strcpy(after_time,optarg);
             break;
      case 'b' :
             strcpy(before_time,optarg);
             break;
      case 'i' :
             interactive_mode=1;
             break;
      case 'h' :
      case '?' :
             nbm_help();
             exit(7);
             break;
      otherwise :
             nbm_help();
             exit(8);
             break;
    }
  }
  if ( interactive_mode==1 ) {
    printf("Path Name (-p) :");
    scanf("%s",path_name);
    printf("\nFile Name (-f) :");
    scanf("%s",file_name);
    printf("\nVersion (-v) - 999 for none, For the last version Ø, for
the previous version -1 and so on :");
    scanf("%ld",&restore_version);
    printf("\nAfter Time (-a) - Time Format is YYYY-MM-DD-
HH.MI.SS.SSSSSS - Enter Ø for none :");
    scanf("%s",after_time);
    printf("\nBefore Time (-b) - Time Format is YYYY-MM-DD-
HH.MI.SS.SSSSSS - Enter Ø for none :");
    scanf("%s",before_time);
    if ( strcmp(after_time,"Ø")==Ø ) strcpy(after_time,"");
    if ( strcmp(before_time,"Ø")==Ø ) strcpy(before_time,"");
```

```
  }
  nbm_restore();
  exec sql commit;
  exit(0);
}
```

NBMCLEAN.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  char file_name[200];
  char path_name[200];
  char backup_path_name[200];
  char backup_host[10];
  char file_host_name[10];
  char username[10];
  long version=0;
  long ret_pd;
  char cre_date[26];
  long file_dir;
  long compressed;
EXEC SQL END DECLARE SECTION;
void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit(1);
}
void nbm_clean()
{
  char
backup_file_name[200],path_name_with[1000],sys_cmd[1000],host_name[10];
  int ind,rc;
  if (gethostname(host_name,10)!=0 ) {
    printf("Problem in getting host name...\n");
    exit(2);
  }
  exec sql declare crs1 cursor for
           select a.HOST,
                  PATH_NAME,
                  FILE_NAME,
                  b.HOST,
                  USERNAME,
```

```
                    PATH,
                    VERSION,
                    DATE_TAKEN,
                    RETPD,
                    FILE_TYPE,
                    COMPRESSED
             from BACKUP_FILES a,BACKUP_PATHS b
             where DATE_TAKEN < CURRENT TIMESTAMP - RETPD days and
                   PATH_NO = BACKUP_PATH_NO;
  if ( sqlca.sqlcode < 0 ) sql_error(1);
  exec sql open crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(2);
  exec sql fetch crs1 into :file_host_name,
                           :path_name,
                           :file_name,
                           :backup_host,
                           :username,
                           :backup_path_name,
                           :version,
                           :cre_date,
                           :ret_pd,
                           :file_dir,
                           :compressed;
  if ( sqlca.sqlcode < 0 ) sql_error(3);
  if ( sqlca.sqlcode==100 ) exit(3);
  while (sqlca.sqlcode!=100 ) {
    for (ind=0;ind<=strlen(path_name);ind++)
      if ( path_name[ind] == '/' ) path_name_with[ind]='!'; else
path_name_with[ind]=path_name[ind];
    if ( file_dir == 1 || compressed == 1 )
      sprintf(backup_file_name,"%s.V%ld%s",file_name,version,".Z");
    else
      sprintf(backup_file_name,"%s.V%ld",file_name,version);
    if ( strcmp(backup_host,host_name) == 0 ) {
      sprintf(sys_cmd,"rm -f %s/
%s!at!%s!%s",backup_path_name,path_name_with,file_host_name,backup_file_name);
    } else {
      sprintf(sys_cmd,"rsh %s -l %s 'rm -f %s/
%s!at!%s!%s'",backup_host,username,backup_path_name,path_name_with,
file_host_name,backup_file_name);
    }
    rc=system(sys_cmd);
    if ( rc == 0 || rc == 512 ) {
      exec sql delete from BACKUP_FILES
               where HOST=:file_host_name and
                     PATH_NAME=:path_name and
                     FILE_NAME=:file_name;;
      if ( sqlca.sqlcode < 0 ) sql_error(4);
      printf("Backup file:%s/%s!at!%s!%s on host %s is removed ( Backup
Date:%s, Retention Period:%ld
```

```
)\n",backup_path_name,path_name_with,backup_host,backup_file_name,backup_host,
cre_date,ret_pd);
    } else {
      printf("Problem in removing...Rc=%ld\n",rc);
      perror("Remove:");
    }
    exec sql fetch crs1 into :file_host_name,
                             :path_name,
                             :file_name,
                             :backup_host,
                             :username,
                             :backup_path_name,
                             :version,
                             :cre_date,
                             :ret_pd,
                             :file_dir,
                             :compressed;
    if ( sqlca.sqlcode < 0 ) sql_error(5);
  }
  exec sql close crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(6);
}

int main(int argc,char **argv)
{
  exec sql connect to nbmdb;
  if ( sqlca.sqlcode < 0 ) sql_error(7);
  nbm_clean();
  exec sql commit;
  if ( sqlca.sqlcode < 0 ) sql_error(8);
  exit(0);
}
```

NBMLIST.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>
#include <string.h>
char old_path_name[200],old_file_name[200],old_host_name[10];
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  char file_name[200];
  char path_name[200];
  char backup_path_name[200];
  char backup_host[10];
  char host_name[10];
  long version=0;
  long ret_pd;
```

```
  char cre_date[26];
  long file_dir;
EXEC SQL END DECLARE SECTION;
void nbm_help()
{
   printf("Usage :\n");
   printf(" nbmlist -p path_name -f file_name \n");
}
void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit(1);
}
void header()
{
  printf("\nFILE=%s/%s HOST=%s FILE_TYPE=%s
\n",path_name,file_name,host_name,(file_dir==Ø)?"FILE":"DIR");
  printf("-------------------------------------------------------------
-----------------------\n");
  printf("version\tbackup_host\t\tbackup_path\t\t
cre_date\t\tret_pd\n\n");
  strcpy(old_path_name,path_name);
  strcpy(old_file_name,file_name);
  strcpy(old_host_name,host_name);
}
void list_by_file_name()
{
  exec sql declare crs1 cursor for
           select PATH_NAME,
                  FILE_NAME,
                  a.HOST,
                  b.HOST,
                  PATH,
                  VERSION,
                  DATE_TAKEN,
                  RETPD,
                  FILE_TYPE
           from BACKUP_FILES a,BACKUP_PATHS b
           where PATH_NAME like :path_name and
                 FILE_NAME like :file_name and
                 PATH_NO = BACKUP_PATH_NO
           order by PATH_NAME,FILE_NAME,VERSION;
  if ( sqlca.sqlcode < Ø ) sql_error(1);
  exec sql open crs1;
  if ( sqlca.sqlcode < Ø ) sql_error(2);
  exec sql fetch crs1 into :path_name,
                           :file_name,
                           :host_name,
```

```
                            :backup_host,
                            :backup_path_name,
                            :version,
                            :cre_date,
                            :ret_pd,
                            :file_dir;
  if ( sqlca.sqlcode < 0 ) sql_error(3);
  if ( sqlca.sqlcode==100 ) exit(2);
  header();
  while (sqlca.sqlcode!=100 ) {
   printf("%d\t%8s\t%30s\t%17s\t%d\n",version,backup_host,backup_path_name,
cre_date,ret_pd);

    exec sql fetch crs1 into :path_name,
                             :file_name,
                             :host_name,
                             :backup_host,
                             :backup_path_name,
                             :version,
                             :cre_date,
                             :ret_pd,
                             :file_dir;
    if ( sqlca.sqlcode < 0 ) sql_error(4);

    if ( strcmp(path_name,old_path_name)!=0 ||
strcmp(file_name,old_file_name)!=0 ) header();
  }
  exec sql close crs1;
  if ( sqlca.sqlcode < 0 ) sql_error(5);
}
void nbm_list()
{
  exec sql connect to nbmdb;
  if ( sqlca.sqlcode < 0 ) sql_error(6);
  list_by_file_name();
}
int main(int argc,char **argv)
{
  int copt,i=0;
  while(1) {
    copt=getopt(argc,argv,"p:f:h?");
    if (copt==-1) break;
    switch(copt) {
      case 'p' :
              strcpy(path_name,optarg);
              break;
      case 'f' :
              strcpy(file_name,optarg);
              break;
      case 'h' :
```

```
      case '?' :
              nbm_help();
              exit(0);
              break;
      otherwise :
              nbm_help();
              exit(0);
              break;
    }
  }
  nbm_list();
  exit(0);
}
```

Abdullah Ongul
DBA
Disbank (Turkey)

Source code control system

INTRODUCTION

The Source Code Control System (SCCS) is a set of utilities to manage any kind of source. SCCS does not provide a menu-based application for management – instead it offers a number of commands with various options. As a result of this, a shell script is almost always required to automate this command line interface and hence make life easier for developers. The article discusses various aspects of SCCS and provides the listing of a shell script, sccs.sh, which interfaces to various SCCS commands.

SCCS TERMINOLOGY

Archive file

The archive file is the file that SCCS creates to manage a specific source and its deltas.

The file is named as s.<source file name>, where **s** refers to SCCS.

This file contains the original content of the source file as well as all the changes that have been made to it.

SCCS file

The SCCS file is the file that SCCS maintains to manage a specific source and its deltas.

This is also known as an archive file.

Working file

A working file is a copy of a specific source file with a particular revision number, which has been archived by SCCS.

Delta

Each revision of a source file recorded in an SCCS file is called a delta.

SID

The revision number of each revision for a specific source file is referred to as a SID (for SCCS ID) (eg 1.2, where 1 = release id or branch id, 2 = sequence no).

SCCS COMMANDS

The command syntaxes are as follows:

```
sccs  < options for sccs >  sub-command < options for subcommands >
```

Example 1:

```
sccs   -d ${SCCS_ROOT_DIR}    get  -r1.2     ${SOURCE_FILE_NAME}
```

Example 2:

```
get  -r1.2   ${SCCS_ROOT_DIR}/SCCS/s. ${SOURCE_FILE_NAME}
```

Note: subcommands themselves are capable of carrying on all

SCCS activities but one needs to provide more information. In the example above, the subcommand, **get**, will not know where the archive file lives, and, therefore, one needs to specify its full path, as well as the prefix **s.**, which stands for archive file.

SETTING UP SCCS

Single directory set-up

1 Create a Unix account (eg sccsadm), which will own all SCCS files and serve as the SCCS administrator.

2 Decide on an SCCS root directory name (eg */u1/sources/*) and create this directory, making SCCS administrator the owner of that directory.

3 Make a subdirectory called *SCCS* under this SCCS root directory.

4 Make all SCCS commands owned by the SCCS administrator.

5 Set the user bit on for all checkin- and checkout-related commands.

Note: all the archive files will reside under the SCCS root directory/SCCS directory.

Multiple directory set-up

1 Create a Unix account (eg sccsadm) that will own all SCCS files and serve as the SCCS administrator.

2 Decide on a SCCS root directory name (eg */u1/sources/*) and create this directory, making SCCS administrator the owner of that directory.

3 Establish all the different file extension types for source files.

4 Make as many subdirectories, called *SCCS/<file extension type>*, as there are file extension types under this SCCS root directory.

5 Make all SCCS commands owned by the SCCS administrator.

6 Set the user bit on for all checkin- and checkout-related commands.

Note: archive files with same file extension type will reside under the same directory

REVISION TREE

How branches are created

By default, SCCS checks out the latest source from branch 1 (eg 1.5). But if SCCS is asked to check out an earlier version of a source for editing, SCCS will create a second branch for the source at check in. Now, if the same source is checked out again for editing, because a second branch already exists, SCCS will create a third branch.

Cheking out source files for update from a multiple branch revision tree

If all the branches reside in the same SCCS directory, one needs to know precisely which version they want to modify. There is no easy way to do this, but document the purpose of multiple branches for source file. Alternatively, you can look at the history of changes, which includes comments for deltas from all the branches, and decide which version to modify.

REMOVING CHANGES FROM SCCS

SCCS allows only the latest change from the revision tree.

PROGRAM LISTING

```
#########################################################################
#   Name     : source code control system  (sccs)
#   Overview : The shell script provides an interfaces to Unix's SCCS
#              utility
#   Notes    : 1. The usage of the script is as follows:
#                    sccs.sh  -D  where -D is optional debug
#                                 parameter which displays full
#                                 error message in vi editor
#              2.  All the underlying utilities of SCCS must be owned
#                  by the SCCS administrator (eg sccsadm) and have
```

```
#                  their user-id bits set.
#                  This administrator cannot be root.
#              3.  The script contains the following functions:
#                   o InitialiseVariables
#                   o ParseCommandLine
#                   o DisplayMessage
#                   o HandleInterrupt
#                   o MoveCursor
#                   o ProcessExit
#                   o FormatUnderscores
#                   o DisplayMenu
#                   o ProcessOption
#                   o DisplayListOfValues
#                   o GetSourceFileName
#                   o GetDirectoryName
#                   o GetReleaseId
#                   o CheckInNewSourceFile
#                   o CheckOutLatestSourceFileForUpdate
#                   o CheckInUpdatedSourceFile
#                   o GetReadOnlyLatestSourceFile
#                   o GetReadOnlySpecificVersionOfSourceFile
#                   o ShowListOfCheckedOutFiles
#                   o ReleaseCheckedOutSourceFile
#                   o ShowSourceReleaseHistory
#                   o ShowVersionDifference
#                   o ShowChangesMade
#                   o UpdateDeltaComment
#                   o InsertSCCSKeyWordsIntoFile
#                   o RemoveLatestDelta
#                   o main
#              4. You  can modify InsertSCCSKeyWordsIntoFile () to
#                 allow for more file extension types
#              5. You can alter the default sub-directory SCCS
#                 under the sccs root directory by creating these
#                 alternative sub-directories under sccs root
#                 directory and running sccs command as follows:
#  sccs –d${SCCS_ROOT_DIRECTORY}
#                       –p<required sub-directory name> < subcommands >
#   History  :
#####################################################################
#####################################################################
#  Name     : InitialiseVariable
#  Overview : The function initializes all module constants and
#             working variables.
#  Notes    :
#####################################################################
InitialiseVariable  ()
{
#  sccs  directory struture
# define SCCS  root directory
SCCS_ROOT_DIR="/export/home/release/mas_dir" ; export SCCS_ROOT_DIR
```

```
# define SCCS sub-directories
SCCS_SQL_DIR="SCCS"    ; export SCCS_SQL_DIR
SCCS_PC_DIR="SCCS"     ; export SCCS_PC_DIR
SCCS_C_DIR="SCCS"      ; export SCCS_C_DIR
SCCS_SH_DIR="SCCS"     ; export SCCS_SH_DIR
#  sccs  command directory
SCCS_BIN_DIR="/usr/ccs/bin"
USER='id  | cut -d'(' -f2 | cut -d')' -f1'
# terminal capabilities
BOLDON=`tput smso`
BOLDOFF=`tput rmso`
#
ESC="\ØØ33["
# menu title
MENU_TITLE="${BOLDON}Source Code Control System (SCCS) Menu${BOLDOFF}"
#
ERROR="sccs.sh:ERROR:"
INFO="sccs.sh:INFO:"
# fuction return values
TRUE=Ø
FALSE=1
#
DEBUG="${FALSE}"
# exit status
SEC=Ø
FEC=1
# sleep duration
SLEEP_DURATION=3
# define  signals
SIGINT=2    ;  export SIGINT     # ctrl-c command
SIGTERM=15  ;  export SIGTERM    # kill  command
SIGTSTP=18  ;  export SIGTSTP    # ctrl-z  command
# temporary file
TEMP_FILE_1=/tmp/sccs_1_$$.tmp
TEMP_FILE_2=/tmp/sccs_2_$$.tmp
LOV_FILE_1=/tmp/sccs_lov1_$$.tmp
LOV_FILE_2=/tmp/sccs_lov2_$$.tmp
# messages
INTERRUPT="Program Interrupted\; Quitting early"
WORKING="Working"
INVALID_ENTRY="Invalid Entry"
OS_ERROR="\${ERR_MSG}"
NEW_CHKIN_FAILED="Failed to check in new source \${SOURCE_FILE_NAME}"
EDITED_CHKIN_FAILED="Failed to check in modified source
\${SOURCE_FILE_NAME}"
EDIT_CHKOUT_FAILED="Failed to reserve the source \${SOURCE_FILE_NAME}
for editing"
READ_CHKOUT_FAILED="Failed to check out the source \${SOURCE_FILE_NAME}
for read"
DIR_NOT_WRITABLE="Directory, \${DIR_NAME} is not writable by user,
${USER}"
```

```
RELEASE_LOCK_FAILED="Failed to cancel reservation for source
\${SOURCE_FILE_NAME}"
CHKOUT_LIST_FAILED="Filed to list checked out sources"
HIST_LIST_FAILED="Failed to list Source Release History"
DEBUG_SET="Debug set"
LOV_FOR_VERSION_FAILED="Failed to generate list of values for source
release"
INVALID_FILE_EXTENSION="Invalid file extension"
DELTA_NOT_REMOVED="Failed to remove delta \${RELEASE_ID} for source
\${SOURCE_FILE_NAME}"
SID_NOT_RETRIEVED="Failed to retrieve latest SID for source
\${SOURCE_FILE_NAME}"
COMMENT_UPDATE_FAILED="Failed to update comment for delta \${RELEASE_ID}
for \${SOURCE_FILE_NAME}"
SOURCE_FILE_NOT_EXISTS="Source file \${SOURCE_FILE_NAME} does not exist
in current directory"
VERSION_DIFF_FAILED="Failed to compare two releases for
\${SOURCE_FILE_NAME}"
CHANGE_DIFF_FAILED="Failed to compare checked out version with latest
delta for \${SOURCE_FILE_NAME}"
}
#########################################################################
#  Name     : HandleInterrupt
#  Overview : The function calls ProcessExit.
#  Notes    :
#########################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"  N
ProcessExit  $FEC
}
#########################################################################
#    Name     : MoveCursor
#    Input    : Y and X coordinates
#    Returns  : None
#    Overview : It moves the cursor to the required location (Y,X).
#    Notes    :
#########################################################################
MoveCursor  ( )
{
YCOR=$1
XCOR=$2
echo     "${ESC}${YCOR};${XCOR}H"
}
#########################################################################
#  Name     : ProcessExit
#  Overview : The function calls ProcessExit.
#  Input    : Exit Code
#  Notes    :
#########################################################################
ProcessExit ()
```

```
{
# assign parameter
EXIT_CODE="$1"
rm -f ${TEMP_FILE_1}
rm -f ${TEMP_FILE_2}
rm -f ${LOV_FILE_1}
rm -f ${LOV_FILE_2}

exit ${EXIT_CODE}
}
#########################################################################
#  Name     : DisplayMessage
#  Overview : The function displays message
#  Input    : 1. Message type (E = Error, I = Informative)
#             2. Error Code as defined in DefineMessages ().
#             3. Message to be acknowledged flag  (Y=yes N=no)
#  Notes    :
#########################################################################
DisplayMessage ( )
{
trap "HandleInterrupt " $SIGINT  $SIGTERM $SIGHUP $SIGTSTP
MESSAGE_TYPE=$1
MESSAGE_TEXT=`eval echo $2`
ACKNOWLEDGE_FLAG="$3"
# default the message acknowledge flag
if [ "${ACKNOWLEDGE_FLAG}" = ""  ]
then
  ACKNOWLEDGE_FLAG="Y"
fi
#clear
MoveCursor 24 1
if [ "${MESSAGE_TYPE}" = "E" ]
then
  if [ "${ACKNOWLEDGE_FLAG}" = "Y"  ]
  then
     echo  "${BOLDON}${ERROR}${MESSAGE_TEXT}${BOLDOFF}\c"
  else
     echo  "${BOLDON}${ERROR}${MESSAGE_TEXT}...${BOLDOFF}\c"
  fi
else
  if [ "${ACKNOWLEDGE_FLAG}" = "Y"  ]
  then
     echo  "${BOLDON}${INFO}${MESSAGE_TEXT}${BOLDOFF}\c"
  else
     echo  "${BOLDON}${INFO}${MESSAGE_TEXT}...${BOLDOFF}\c"
  fi
fi
# examine message acknowledge flag
if [ "${ACKNOWLEDGE_FLAG}"  =  "Y"  ]
then
      read  DUMMY
```

```
else
      sleep ${SLEEP_DURATION}
fi
return ${TRUE}
}
#########################################################################
#  Name     : FormatUnderscores
#  Overview : The function assigns appropriate number of
#             underscores(=) to the variable UNDERSCORE to be used
#             in conjunction with a header.
#  Input    : Line containing the header
#  Notes    :
#########################################################################
FormatUnderscores ()
{
# assign parameter
LINE="$1"
# initialize  UNDERSCORE
UNDERSCORE=
# initialize  index
IND=1
#  get no of characters in  $LINE
NO_CHARS=`echo "$LINE" |  wc  -c`
#  subtract the carriage return
NO_CHARS=`expr $NO_CHARS - 1`
while [ "$IND" -le "$NO_CHARS" ]
do
  UNDERSCORE="${UNDERSCORE}="
  IND=`expr $IND + 1`
done
}
#########################################################################
#  Name     : DisplayListOfValues
#  Overview : The function displays a list of values for source file
#             names.
#  Inputs   : List of Value type (S (source) or V (version no))
#  Returns  : $TRUE  or $FALSE
#  Notes    :
#########################################################################
DisplayListOfValues ()
{
# assign parameter
LOV_TYPE="$1"
DATETIME=`date "+%d/%m/%Y at %H:%M:%S"`
# initialize selected value variable
SELECTED_VALUE=""
#
if  [  "${LOV_TYPE}"  =  "S"  ]
then
    SCCS_DIR="${SCCS_ROOT_DIR}/SCCS"
```

```
    ls -1 ${SCCS_DIR}   >  ${TEMP_FILE_1}
    # prepare LOV files
    #
    HEADER="List of values for Source Files on ${DATETIME}"
    FormatUnderscores  "${HEADER}"
    echo "  ${HEADER}"        >   ${LOV_FILE_1}
    echo "  ${UNDERSCORE}"   >>  ${LOV_FILE_1}
    echo "  To Select Delete Corresponding Line and Save
File\n">>${LOV_FILE_1}
    cat ${TEMP_FILE_1}  >> ${LOV_FILE_1}
    cp   ${LOV_FILE_1} ${LOV_FILE_2}
    #
    vi ${LOV_FILE_1}
    SELECTED_VALUE=`diff ${LOV_FILE_1}  ${LOV_FILE_2} | tail -1 | \
                          awk {'print $2'}`
    # strip off file types
    SELECTED_VALUE=`echo ${SELECTED_VALUE} |sed s/^..//`
#
elif  [  "${LOV_TYPE}"  =  "V"  ]
then
   # get source history
   ${SCCS_BIN_DIR}/sccs -d${SCCS_ROOT_DIR} prs -l -r1.1
${SOURCE_FILE_NAME} >\
                                                ${TEMP_FILE_1}  2>&1
   if  [  $?  -ne  0  ]
   then
     DisplayMessage E "${LOV_FOR_VERSION_FAILED}" N ;
     ERR_MSG=`cat  ${TEMP_FILE_1}` ;
     DisplayMessage E "${OS_ERROR}" ;
     #
     if [ "${DEBUG}"  = "${TRUE}"   ]
     then
        view ${TEMP_FILE_1}
     fi
     #
     return $FALSE
   fi
   # prepare list of values file
   HEADER="List of values for Release Ids for source ${SOURCE_FILE_NAME}
on ${DATETIME}"
    FormatUnderscores  "${HEADER}"
    echo "  ${HEADER}"        >   ${LOV_FILE_1}
    echo "  ${UNDERSCORE}"   >>  ${LOV_FILE_1}
    echo "  To Select Delete Corresponding Line and Save
File\n">>${LOV_FILE_1}
    cat ${TEMP_FILE_1}  >> ${LOV_FILE_1}
    cp   ${LOV_FILE_1} ${LOV_FILE_2}
    #
    vi ${LOV_FILE_1}
    SELECTED_VALUE=`diff ${LOV_FILE_1}  ${LOV_FILE_2} | tail -1 | \
                          awk {'print $3'}`
```

```
fi
return $TRUE
}
#########################################################################
#  Name     : InsertSCCSKeyWordsIntoFile
#  Overview : The function inserts SCCS keywords into a new source
#             file.
#  Returns  : $TRUE  or $FALSE
#  Notes    :
#########################################################################
InsertSCCSKeyWordsIntoFile ()
{
#       ____________________________________________________________
#        Keyword                          Value
#       ____________________________________________________________
#          %A%      Shorthand notation for an ID line with data for
#                   what(1): %Z%%Y%  %M%  %I%%Z%
#       ____________________________________________________________
#          %B%                    SID branch component
#       ____________________________________________________________
#          %C%      Current line number. Intended for identifying
#                   messages output by the program such as ``this
#                   shouldn't have happened`` type errors. It is
#                   not intended to be used on every line to pro-
#                   vide sequence numbers.
#       ____________________________________________________________
#          %D%                  Current date: yy/mm/dd
#       ____________________________________________________________
#          %E%      Date newest applied delta was created: yy/mm/dd
#       ____________________________________________________________
#          %F%                     SCCS s.file name
#       ____________________________________________________________
#          %G%      Date newest applied delta was created: mm/dd/yy
#       ____________________________________________________________
#          %H%                  Current date: mm/dd/yy
#       ____________________________________________________________
#          %I%       SID of the retrieved version: %R%.%L%.%B%.%S%
#       ____________________________________________________________
#          %L%                    SID level component
#       ____________________________________________________________
#          %M%      Module name: either the value of the m flag  in
#                   the  s.file (see sccs-admin(1)), or the name of
#                   the s.file less the prefix
#       ____________________________________________________________
#          %P%                Fully qualified s.file name
#       ____________________________________________________________
#          %Q%             Value of the q flag in the s.file
#       ____________________________________________________________
#          %R%                    SID Release component
#       ____________________________________________________________
#          %S%                    SID Sequence component
```

```
#      ____________________________________________________________
#         %T%                  Current time: hh:mm:ss
#      ____________________________________________________________
#         %U%       Time the newest applied delta was created:
#                   hh:mm:ss
#      ____________________________________________________________
#         %W%       Shorthand notation for an ID line with data for
#                   what : %Z%%M%  %I%
#      ____________________________________________________________
#         %Y%       Module type: value of the t flag in the s.file
#      ____________________________________________________________
#         %Z%       4-character string: `@(#)', recognized by what
#      ____________________________________________________________
#
# extract file extension
FILE_EXT=`echo "${SOURCE_FILE_NAME}" | sed s/.*\\\./g`
# prepare comment symbol according to file extension
case ${FILE_EXT}  in
       pc) START_COMMENT="/*"   ;
             END_COMMENT="*/"   ;;
      sql) START_COMMENT="/*"   ;
             END_COMMENT="*/"   ;;
        c) START_COMMENT="/*"   ;
             END_COMMENT="*/"   ;;
       sh) START_COMMENT="#"   ;;
      "" ) DisplayMessage  E "${INVALID_FILE_EXTENSION}" N ;
           return $FALSE ;;
       * ) DisplayMessage  E "${INVALID_FILE_EXTENSION}" N ;
                             return $FALSE ;;
esac
# prepare keyword strings
if [ "${FILE_EXT}" = "sql" -o "${FILE_EXT}" = "c"  -o "${FILE_EXT}" =
"pc" ]
then
   KEYWORDS_STRINGS="${START_COMMENT}\nSCCS Control Information\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}======================\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}\nModule Name: %M%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}    Version: %I%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}    Created: %H%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}   Filename: %P%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}Last Edited: %E%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}    What ID: %W%\n"
   KEYWORDS_STRINGS="${KEYWORDS_STRINGS}${END_COMMENT}"
#
elif [ "${FILE_EXT}" = "sh" ]
then
   KEYWORDS_STRINGS="${START_COMMENT} SCCS Control Information\n"
   KEYWORDS_STRINGS=
        "${KEYWORDS_STRINGS}${START_COMMENT} ======================\n"
   KEYWORDS_STRINGS=
               "${KEYWORDS_STRINGS}${START_COMMENT} Module Name: %M%\n"
```

```
   KEYWORDS_STRINGS=
                "${KEYWORDS_STRINGS}${START_COMMENT}     Version: %I%\n"
   KEYWORDS_STRINGS=
                "${KEYWORDS_STRINGS}${START_COMMENT}     Created: %H%\n"
   KEYWORDS_STRINGS=
                "${KEYWORDS_STRINGS}${START_COMMENT}    Filename: %P%\n"
   KEYWORDS_STRINGS=
                "${KEYWORDS_STRINGS}${START_COMMENT} Last Edited: %E%\n"
   KEYWORDS_STRINGS=
                "${KEYWORDS_STRINGS}${START_COMMENT}     What ID: %W%\n"
fi
#
echo "${KEYWORDS_STRINGS}"  >  ${TEMP_FILE_1}
cat   ${SOURCE_FILE_NAME}   >> ${TEMP_FILE_1}
cat   ${TEMP_FILE_1}        >  ${SOURCE_FILE_NAME}
#
if [ "${DEBUG}"  = "${TRUE}"   ]
then
    view ${SOURCE_FILE_NAME}
fi
return $TRUE
#
}
#########################################################################
#  Name     : GetSourceFileName
#  Overview : The function accepts a source file name.
#  Inputs   : Mode (CI (checkin) or CO (checkout))
#  Returns  : $TRUE  or $FALSE
#  Notes    :
#########################################################################
GetSourceFileName ()
{
# assign parameter
MODE="$1"
#
while  true
do
   clear
   if [  "${MODE}" = "CI"  ]
   then
      # get checkin file name
      echo "Enter File name in current directory(a to abandon):\c"
   else
      # get checkout file name
      # display list of values
     SELECTED_VALUE=""
     echo "Enter File name (l for list of values or a to abandon ):\c"
   fi
   read  SOURCE_FILE_NAME
   #
   case  ${SOURCE_FILE_NAME}  in
```

```
       "") DisplayMessage E "${INVALID_ENTRY}" ;;
        a) return  $FALSE  ;;
        l) DisplayListOfValues "S"  ;
           if [ "${SELECTED_VALUE}" = ""  ]
           then
                 : ;
           else
                SOURCE_FILE_NAME="${SELECTED_VALUE}" ;
                break ;
           fi ;;
       * ) if [  "${MODE}" = "CI"  ]
           then
              # validate this file name
              if [ ! -f ./${SOURCE_FILE_NAME}  ]
              then
                   DisplayMessage  E "${SOURCE_FILE_NOT_EXISTS}" N ;
                   return $FALSE ;
              else
                   return $TRUE ;
              fi  ;
           else
               # checkout file name
                  return $TRUE ;
           fi ;;
   esac
done
}
#########################################################################
#  Name     : GetDirectoryName
#  Overview : The function accepts a directory name.
#  Returns  : $TRUE  or  $FALSE
#  Notes    :
#########################################################################
GetDirectoryName ()
{
#
while  true
do
   clear
   echo "Enter directory name for output file"
   echo "Press Retrun for current directory (a to abandon):\c"
   read DIR_NAME
   case  ${DIR_NAME}  in
        a) return  $FALSE  ;;
       "") DIR_NAME=`pwd` ;
           return $TRUE ;;
        *) # validate directory
           if [ !  -w ${DIR_NAME}  ]
           then
                 DisplayMessage E "${DIR_NOT_WRITABLE}" ;
           else
```

```
                return $TRUE ;
            fi ;;
   esac
done
}
#########################################################################
#  Name     : GetReleaseId
#  Overview : The function accepts a release id for a specific source
#             file.
#  Returns  : $TRUE  or  $FALSE
#  Notes    :
#########################################################################
GetReleaseId ()
{
# get release id
while  true
do
   clear
   echo "Enter the release id( l for list of values )"
   echo "(a to abandon):\c"
   read  RELEASE_ID
   case  ${RELEASE_ID} in
      "" ) DisplayMessage E "${INVALID_ENTRY}" ;;
        a) return  $FALSE  ;;
        l) DisplayListOfValues "V"  ;
           if [ "${SELECTED_VALUE}" = ""  ]
           then
                 : ;
           else
                RELEASE_ID="${SELECTED_VALUE}" ;
                break ;
           fi ;;
       * ) break    ;;
   esac
done
#
}
#########################################################################
#  Name     : DisplayMenu
#  Overview : The function displays menu.
#  Notes    :
#########################################################################
DisplayMenu ()
{
    clear
    echo "
        ##############################################################
        #                                                            #
        #      ${MENU_TITLE}                                         #
        #                                                            #
```

```
        #    5.  Check In New  Source                              #
        #   1Ø.  Check Out Latest Version for Update               #
        #   15.  Check Out Specific Version for Update             #
        #   2Ø.  Check In Updated Source                           #
        #   25.  Check Out Read Only Latest Source                 #
        #   3Ø.  Check Out Read Only Specific Version of Source    #
        #   35.  Display List of Checked Out Source Files          #
        #   4Ø.  Release Checked Out Source File                   #
        #   45.  Show Source Release History                       #
        #   5Ø.  Update Comment for Specific Change (Delta)        #
        #   55.  Remove Latest Delta (change)                      #
        #   6Ø.  Show Difference Between Two Versions              #
        #   65.  Show Difference Between Checked Out and Archive   #
        #                                                          #
        #   99.  Exit                                              #
        #                                                          #
        ############################################################
                 Enter Option ----> \c

      "
  read  OPTION
}
#########################################################################
#  Name     : CheckInUpdatedSourceFile
#  Overview : The function checks in updated source file.
#  Notes    :
#########################################################################
CheckInUpdatedSourceFile  ( )
{
if ! GetSourceFileName "CI"
then
    return $FALSE
fi
#
#
${SCCS_BIN_DIR}/sccs -d${SCCS_ROOT_DIR} delget  ${SOURCE_FILE_NAME} \
                                                  2> ${TEMP_FILE_1}
if  [  $?  -ne  Ø  ]
then
    DisplayMessage E "${EDITED_CHKIN_FAILED}" N ;
    ERR_MSG=`cat  ${TEMP_FILE_1}` ;
    DisplayMessage E "${OS_ERROR}" ;
    #
    if [ "${DEBUG}"  = "${TRUE}"   ]
    then
        view ${TEMP_FILE_1}
    fi
    #
    return $FALSE
fi
```

```
#
return $TRUE
}
#########################################################################
#  Name     : CheckOutLatestSourceFileForUpdate  ( )
#  Overview : The function checks out the lastest source file for
#             update.
#  Notes    :
#########################################################################
CheckOutLatestSourceFileForUpdate   ()
{
#
if ! GetSourceFileName "CO"
then
    return $FALSE
fi
#
if ! GetDirectoryName
then
    return $FALSE
fi
# remove the file to be checked out from target directory
rm  -f ${DIR_NAME}/${SOURCE_FILE_NAME}
#
${SCCS_BIN_DIR}/sccs -d${SCCS_ROOT_DIR} edit -p  ${SOURCE_FILE_NAME} \
                     1> ${DIR_NAME}/${SOURCE_FILE_NAME} 2>
${TEMP_FILE_1}
if  [  $?  -ne  Ø  ]
then
   DisplayMessage E "${EDIT_CHKOUT_FAILED}" N
   ERR_MSG=`cat  ${TEMP_FILE_1}`
   DisplayMessage E "${OS_ERROR}"
   #
   if [ "${DEBUG}"  = "${TRUE}"   ]
   then
        view ${TEMP_FILE_1}
   fi
   #
   return $FALSE
else
    return $TRUE
fi
#
}
```

Editor's note: this article will be concluded next month.

Arif Zaman
ETL Developer (UK)

The until and while loops

The following two looping constructs, **until** and **while**, are very similar in that they both allow a series of commands to be run repetitively while (or until) a condition is true. As you will discover, **until** is much less useful than **while** and most of your scripts will contain the latter construction.

Both **while** and **until** are frequently used with integer arithmetic tests, such as 'while we have not performed the loop 10 times', or 'until we have performed the loop 10 times' and we will see in a future article how this can be achieved.

Generally speaking, you can convert any **while** to an **until** by simply negating the condition, or using the opposite of a particular test.

THE UNTIL LOOP

The **until** loop has the following format:

```
until condition_command
do
        loop_body
done
```

The *condition_command* may be any command or pipeline, and its exit status will determine whether the commands in *loop_body* are to be executed. The looping continues *until condition_command* is successful/true; that is, it returns an exit status of 0. Remember that in a pipeline it will be the exit status of the last command that is important.

You should be aware that the commands in the body of the loop might never be executed. If *condition_command* is successful the first time it is run, the loop terminates and the body of the loop is not executed.

The conventions for the format, and rules regarding the positioning of keywords, are the same as those for the **for** command.

You should consider using an **until** loop any time a task involves waiting for something to happen. As an example, consider the following simple script that checks periodically to see whether a particular user has logged in; the script will make your terminal beep and display a message on your screen when the specified user logs in:

```
$ vi waitfor

until who | grep $1 >/dev/null
do
      sleep 100
done

print '$1 logged in'
# escape code to make the terminal beep
print '\07 \07'
```

The *condition_command* is the pipeline:

```
who | grep $1 >/dev/null
```

The exit status of the **grep** command is used to determine whether or not the looping is to continue. **grep** has a zero exit status only when it finds the string you are searching for, otherwise it returns a non-zero value. We have redirected the output of **grep** to */dev/null* since we are not interested in displaying the matching lines of output from the **who** command.

The **sleep** command is used to delay execution of the condition command for the specified number of seconds, 100 in this case. Every 100 seconds, **who | grep $1 >/dev/null** is run to check for the user. When the user specified by **$1** logs in, **grep** prints the line from the **who** output that contains the specified username, and returns an exit status of **0**. This causes the looping to stop, and the two **print** commands are executed.

The **print** command interprets the quoted sequence of characters **\0nnn**, where **nnn** is an octal number, as the character whose ASCII code is **nnn**. For instance, if **print** is passed the characters **\0141**, it will print the letter **a** since **141** is the octal code for **a**. This format is normally used to print escape sequences and control characters, such as the one with ASCII code **7** that we have used

in our example. Printing this control character causes the terminal to beep. The *backslash* must be quoted or the shell will remove it before running **print**.

It is advisable to run **waitfor** in the background; for example, **waitfor fred &**. Two beeps will alert you when **fred** has logged in.

THE WHILE LOOP

The **while** loop is very similar to the **until** loop, except that the body of a **while** loop is repeated as long as the *condition_command* is successful. The looping will stop when the condition command returns a non-zero exit status. It is also possible to create a **while** loop in which the body of the loop is never executed – many shell programmers do this without even trying!

The syntax of the **while** loop is:

```
while condition_command
do
        loop_body
done
```

If *condition_command* returns a 0 exit status, the shell then executes the commands in the body of the loop, and repeats the loop initialization step. As soon as *condition_command* returns a non-zero exit status, execution of the loop is completed, and the loop is exited.

A **while** loop can be used to create an interactive program that will accept the user's requests until the user gives a termination request.

MODIFYING LVMAN TO USE WHILE LOOPS

Let us now see how we can modify our **lvman** script so that it can be used to check the free space status of all our disks and volume groups by using a **while** loop.

The modified script is as follows, and changes are shown in italics:

```
#!/bin/ksh
# Script name: lvman
# Usage: lvman {[-v VG1name -v VG2name..| -v all]|
#            [-p PV1name -p PV2name..| -p all]}
############################################################
# Version History
############################################################
#-----------------------------------------------
# Function: f_dsp_usage
# Displays usage messages
#-----------------------------------------------
f_dsp_usage()
{
    print "Usage: $(basename $0) {[-v VG1name -v VG2name..| -v all] |"
    print "            [-p PV1name -p PV2name..| -p all]}"
    print "Where:"
    print "\t-v VG1name -v VG2name.. specifies volume group list"
    print "\t-v all specifies all volume groups"
    print "\t-p PV1name -p PV2name.. specifies physical volume list"
    print "\t-p all specifies all physical volumes"
}
#-----------------------------------------------
# Function: f_chk_valid
# Arguments: $1 - volume group or physical volume
# Checks the volume group or physical volume name is valid
#-----------------------------------------------
f_chk_valid()
{
    DEV=$1
    [[ $(echo "$DEV" | grep -c "-") -ne 0 ]] && return 2
    lsattr -El $DEV >/dev/null 2>&1
    # lsattr returns 0 for valid device,
    # or 255 for non valid device
    case $? in
    0)
      return 0 ;;
    *)
      return 1 ;;
    esac
}
#-----------------------------------------------
# Function: f_get_vg_space
# Arguments: $1 - volume group name
# Gets the total and free space of the volume group
#-----------------------------------------------
f_get_vg_space()
{
    VG=$1
    # Get total space and free space
    TOTAL=$(lsvg $VG | grep "TOTAL PPs" | cut -f2 -d "(" |
```

```
      tr ' ' '\t' | cut -f1)
    FREE=$(lsvg $VG | grep "FREE PPs" | cut -f2 -d "(" |
      tr ' ' '\t' | cut -f1)
    eval ${VG}_LVNUM=$(lsvg -l $VG | tail +3 | wc -l | tr -d " ")
    eval NUMLVS='$'${VG}_LVNUM
    # Print output
    if [[ $FIRST -ne 1 ]]
    then
      printf "\n%-20s %-15s %-15s %-15s\n" \
          "Volume Group" "Total Size" "Free Space" "Number LVs"
    fi
    printf "%-20s %-15s %-15s %-15s\n" \
          $VG "$TOTAL MB" "$FREE MB" $NUMLVS
}
#---------------------------------------------
# Function: f_get_pv_space
# Arguments: $1 - physical volume name
# Gets the total and free space on a physical volume
#---------------------------------------------
f_get_pv_space()
{
    PV=$1
    # Get total space and free space
    TOTAL=$(lspv $PV | grep "TOTAL PPs" | cut -f2 -d "(" |
      tr ' ' '\t' | cut -f1)
    FREE=$(lspv $PV | grep "FREE PPs" | cut -f2 -d "(" |
      tr ' ' '\t' | cut -f1)
    eval ${PV}_LVNUM=$(lspv -l $PV | tail +3 | wc -l | tr -d " ")
    eval NUMLVS='$'${PV}_LVNUM
    # Print output
    if [[ $FIRST -ne 1 ]]
    then
      printf "\n%-20s %-15s %-15s %-15s\n" \
          "Physical Volume" "Total Size" "Free Space" "Number LVs"
    fi
    printf "%-20s %-15s %-15s %-15s\n" \
          $PV "$TOTAL MB" "$FREE MB" $NUMLVS
}
################################################
# Main section
################################################
while getopts :v:p: opt
do
    case $opt in
    v)
      if [[ $OPTARG = all ]]
      then
          VGS=$(lsvg -o | sort)      # all volume groups
      else
          VGS=$OPTARG
```

```
  fi
  FIRST=Ø
  for VG in $VGS
  do
      f_chk_valid $VG
      case $? in
      Ø)
        f_get_vg_space $VG
        FIRST=1
        ;;
      1)
        print $VG is not a valid volume group
        exit 1
        ;;
      2)
        f_dsp_usage
        ;;
      esac
  done
  ;;
p)
  if [[ $OPTARG = all ]]
  then
      pvs=$(lsdev -Cc disk -r name)      # all physical volumes
      for pv in $pvs
      do
        if [[ $(lsdev -Cl $pv | grep -c Available) -eq 1 ]]
        then
            PVS=$PVS" $pv"            # only want Available disks
        fi
      done
  else
      PVS=$OPTARG
  fi
  FIRST=Ø
  for PV in $PVS
  do
      f_chk_valid $PV
      case $? in
      Ø)
        f_get_pv_space $PV
        FIRST=1
        ;;
      1)
        print $PV is not a valid physical volume
        exit 2
        ;;
      2)
        f_dsp_usage
        ;;
```

```
            esac
        done
        ;;
    *)
        f_dsp_usage
        exit 3
        ;;
    esac
done
```

Our previous version of **lvman** allowed us to check only one volume group or one physical volume at a time. As you will see, we have made a number of changes to the script to allow all volume groups and all the physical volumes, or a specified number of volumes, groups, and physical volumes, to be checked at the same time using modified command line arguments.

When we previously used **getopts** we ran it just once so that our script checked only the first option it encountered. By putting **getopts** in a while loop using **while getopts :v:p opt**, we run **getopts** as many times as there are command line options. This allows us to specify multiple **-v** and **-p** options, and we have additionally introduced the **all** argument to both of these options to make it easier to check all of our volume groups or physical volumes without having to specify the full list.

Our **f_dsp_usage** function has been modified to show the new argument combinations.

Within our main section the **case** statement stanzas relating to the **-v** and **-p** options first check whether there is an **all** argument. If so, then the list of all *varied-on* volume groups, or all *available* physical volumes, is generated, and **for** loops are used to run the validity checks and get the free space.

If the **all** argument is not used, then there will be only one value for the variable **VGS** (or **PVS**) and this portion of code will be repeated for however many **-v** (or **-p**) options are used.

We now check that each of the volume groups or physical volume names in our list is a valid device name. This is necessary because we may have entered an invalid name as an argument to a **-v** or **-p** option, rather than using the **all** argument,

and since the same variable name is used in our coding to contain the list of volume groups (or physical volumes) to be checked, no matter which options and arguments were used with **lvman**, then we need to include this additional error checking in our script.

One of the problems with using **getopts** is that it does not scan the whole of the command line before deciding whether there are errors in the usage; every time **getopts** is executed it checks the option and argument combination it encounters, rather than all of them. This can cause problems when in error you run the script using, for example:

```
lvman -v -v rootvg
```

and forget to give an argument to the first **-v**. In this situation **getopts** sets the **OPTARG** variable to the second **-v**. To overcome this possible error condition we have introduced a further check in the **f_chk_valid** function to test whether there is a 'minus' sign in the argument:

```
[[ $(echo "$DEV" | grep -c "-") -ne Ø ]] && return 2
```

You will note that we have used the **echo** command in this test, rather than **print**. The reason for this is that if **$DEV** contains a minus sign, which would occur in the example above, then **print "$DEV"** will produce an error condition since it will interpret the minus, and, whatever follows it, as an argument to **print**, and will probably tell you that "A specified flag is not valid for this command"; **echo**, on the other hand, will take the characters as is.

A further failing of **getopts** (perhaps it's not quite the panacea you first thought!) is that it will not produce an error condition if it encounters an unexpected argument which is not preceded by an option starting with a minus or a plus. For example, **getopts** will not complain if you run your script with **lvman rootvg**, or **lvman -v rootvg myvg**. You will see when we come to arithmetic evaluation how we can introduce further tests to overcome these failures.

The remaining changes to the script are purely cosmetic in that the f**_get_vg_space** and **f_get_pv_space** functions have been modified so that only single header lines are printed at the top of each volume group or physical volume section; to do this we check the value of the **FIRST** variable. We have also added an extra column to display the number of logical volumes in the volume group, or on the disk, rather than having these printed on a separate line.

MORE ABOUT BREAK AND CONTINUE

Like the **continue** command, **break** is only meaningful inside a loop. Both **break** and **continue** are used inside **for**, **while**, and **until** loops. The **break** command always causes immediate termination of the surrounding loop; the remainder of the body of the loop is skipped, and the shell proceeds to execute the commands that follow the loop, if any such commands exist.

Regardless of the type of loop in which it is used, **continue** always causes the next iteration of the loop to begin immediately. The remainder of the loop body is skipped and loop initialization is performed again, as if the last command of the loop had just terminated.

You can think of **continue** as instructing the shell to go to the bottom of the loop (the position right after the last command in the body of the loop). Once the shell arrives at this point, it automatically returns to the loop initialization step, and proceeds with the next iteration of the loop.

The **break** command, however, tells the shell to go to the command that immediately follows the **done** keyword. Note that a **break** command in a **case** construct will terminate the loop that surrounds the whole **case** statement, if there is one; this behaves differently from **break** used in C. The double semi-colon at the end of each pattern-matching stanza in a **case** statement performs the same function as a **break** command.

Many traditionalist programmers consider the use of **break** commands to exit loops as bad coding style, since it can often

lead to confusion when using multiple nested loops, but if it helps you to produce simple coding which you can easily understand, then feel free to use it.

REDIRECTING A LOOP'S INPUT AND OUTPUT

As mentioned before, the shell treats an entire loop as if it were a single command. As a result, the standard input of a loop can be redirected, causing each of the commands executed in the loop initialization, and in the body of the loop, to read from a specified file. Redirection is either done on the last line of the loop, following the keyword **done**, or by piping the output of another command into the loop itself.

The output of a loop is redirected from the **done** keyword in exactly the same way that you would redirect the standard output, or standard error, of any command. Redirecting the output at the end of a loop once, rather than from individual commands within the loop, will make a shell script run faster.

The following example shows how we can redirect the standard input and output of a loop. Let us assume that we have run the **iostat** command over a number of time intervals and redirected the output to a text file, called **diskout**. From this file we want to extract only the number of KB read and written during each time interval. The source file will contain lines similar to the following:

```
tty:  tin    tout      avg-cpu:  %user     %sys    %idle     %iowait
      Ø.Ø    Ø.4                   12.9      7.1     21.8       58.2

Disks        %tmact      Kbps      tps      Kb_read      KB_wrtn
hdiskØ         Ø.3       1.4      Ø.3        347857      2614583
hdisk1         Ø.Ø       Ø.1      Ø.Ø        3Ø3372          124
cdØ            Ø.Ø       Ø.Ø      Ø.Ø             Ø            Ø
```

The first group of seven lines will contain cumulative information since the system was booted. Any subsequent groups of seven lines contain statistics for each time interval specified when we run the **iostat** command.

The script is called **diskio**, and introduces a new form of the **set** command. As you are probably aware, **set** on its own will display

all shell variables, and you may also have used commands such as **set -x**, **set +x**, etc, to turn various options on and off. When **set** is invoked with the **--** argument, however, it does something entirely different. It assigns an argument to the first positional parameter, **$1**, and if there are several such arguments, they are assigned to the subsequent positional parameters, **$2**, **$3** etc. It also updates **$#**, **$***, and **$@** accordingly.

You will probably need to read the explanation that follows before you can understand how the script works.

```
$ vi diskio

SOURCE=diskout
DISK=$1
printf "%10s %15s %15s\n" Disk "KB Read" "KB Written"

while read line
do
      [[ -z $line ]] && continue

      set -- $line
      [[ $1 != $DISK ]] && continue
      printf "%10s %15s %15s\n" $1 $5 $6
done < $SOURCE | tail +2
```

The script can be executed by entering, for example:

```
diskio hdisk0
```

After setting the **SOURCE** and **DISK** variables and printing out the headings, our **while** loop gets its standard input from **$SOURCE** via the input redirection symbol after the **done** keyword, and every time it encounters a carriage return it reads the whole of the line into the **line** variable.

If a blank line is read, **$line** will be set to the null string, and **[[-z $line]]** will return a successful exit status. When this happens, the **continue** command is executed, and the loop initialization will begin again. Thus, when a blank line is read, the script immediately reads the next line without printing anything.

If we did not check whether **$line** was null, **set** would be run without arguments whenever a blank line is read, and this would display the name and value of each shell variable, which is not

what we want to do. For this reason, you must always ensure that **set** is passed a non-null argument when you are using it to set positional parameters.

The **set --** command then assigns the contents of **line** to the various positional parameters, and if **$1** does not equal the disk name we passed in from the command line, then we jump to the start of the loop using the **continue** command and read in the next line. When there is a match, **$1** (the disk name), **$5** (KB read), and **$6** (KB written) are printed out.

Finally we pipe the output from the loop to the **tail +2** command to remove the first line of the output, as this line contains the total values since the system was booted and we are not interested in these. We could also send this output to a new file using:

```
done < $SOURCE | tail +2 > newfile
```

In the same way that the redirection of the output of individual commands within a script overrides any redirection from the command line, then the redirection of the output of commands within the loop itself will also override the redirection for the whole loop.

Of course, in the preceding example, you could get the same result by redirecting the input and output on the command line instead of within the script. There are situations, however, where the redirection cannot be handled on the command line. For example, you may want the commands in a loop to write to one destination, while other commands in the script write to a different file or device.

TRUE AND FALSE

There are two commands that are not built into the shell, but which are very useful when you write shell programs containing looping constructs.

The first of these is **false**, which always returns a non-zero (**255**) exit status. It is used to control an **until** loop as follows:

```
until false
do
      .
      .
done
```

Since **false** cannot return a *zero* exit status, this loop will continue to run until such time as some condition is satisfied within the loop which allows you to break out of it.

The **true** command always returns a **0** exit status, and is used to create an endless **while** loop. The following example is a variation on the **waitfor** script above:

```
while true
do
      if [[ $(who | grep -c fred) -eq 0 ]]
      then
            sleep 60
      else
            print fred logged in
            break
      fi
done
```

Tonto Kowalski
Guru (UAE)

Articles for inclusion in *AIX Update* can be sent to the editor, Trevor Eddolls, at trevore@xephon.com. A copy of our *Notes for Contributors* can be downloaded from www.xephon.com/nfc.

AIX news

IBM has announced upgraded WebSphere Business Integration Adapters for pulling information from systems including a range of application types, technology protocols, databases, and trading partner systems.

The new release includes the iSoft Peer-to-Peer Agent and the WebSphere Business Integration Trading Partner Interchange On-Ramp, as well as WebSphere Business Integration Adapters for Retek, i2, Spirent applications, Telcordia applications, eMatrix, enhanced components, and e-business.

It runs on AIX, Solaris 7 or 8, and Windows 2000 or Windows NT 4.

For further information contact your local IBM representative.
URL: http://www.ibm.com/websphere/integration.

* * *

Candle has introduced its PathWAI modular suite of applications, which helps sites design, develop, deploy, and manage WebSphere infrastructures, accelerate return on investment, and enable consistent performance, according to the vendor.

The first PathWAI application is Architecture for WebSphere, which helps give sites the knowledge required to avoid architecture errors that drive delays and cost overruns. It also helps minimize new application development cycles and downtime risk and ensures that the final applications meet required business customer service levels.

It includes application performance testing and tuning tools for WebSphere Application Server or MQ Integrator, plus services and training to meet required business performance metrics. The management applications for testing and monitoring MQ Integrator and WebSphere Application Server are available on AIX and OS/390, Solaris, and Windows NT platforms.

The Monitor for WebSphere Application Server allows users to test, tune, and monitor the performance and availability for the WebSphere AS and its Java-based tools. It monitors the performance characteristics of specific servlets, Java Server Pages, and EJBs, speeding resolution times and providing a central point of control for management. It supports WebSphere AS on AIX, OS/390, Solaris, Windows NT and 2000, and z/OS.

For further information contact:
candle, 201 N Douglas St, El Segundo, CA 90245, USA.
URL: http://www.candle.com/www1/cnd/portal/CNDportal_Channel_Master/0,2938,1904094_2889,00.html.

* * *

SAS company DataFlux has announced the availability of Version 5.0 of its dfPower Studio, Blue Fusion SDK, and dfIntelliServer (formerly called Blue Fusion CS). The company has re-engineered the products to enable users to use an integrated set of products that allows either virtual or physical integration of any type of data.

Supported platforms include AIX, HP-UX, Linux, Solaris, Tru64, Windows NT, 2000, and XP, and OS/390 on BlueFusion SDK.

For further information contact:
DataFlux Corporation, 4001 Weston Parkway, Suite 300, Cary, North Carolina 27513, USA
URL: http://www.dataflux.com/products/database.asp.