79

AIX update

May 2002

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 1998 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 and Richard Watson

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.

Perl and regular expressions

INTRODUCTION

Regular expressions are one of the most useful tools for string manipulation. AIX administrators and users should have a very in-depth knowledge of this topic because they probably need them every day when using awk, sed, vi, and shell. New users just starting to work with AIX need to know regular expressions or they could be in trouble while trying to use, for example, Perl. This article is intended to be a primer on this topic for Perl programmers.

REGULAR EXPRESSIONS AND PERL

Roughly speaking a regular expression is a pattern defining a set of strings. Let's consider for example the following regular expression:

```
/knowledge/
```

This expression defines the class of all the strings containing the word *knowledge*. The Perl interpreter contains a built-in regular expression checker that has the ability to decide whether a string belongs to the class specified by a regular expression or not. In order to perform this check the matching operator, **m/…./**, should be used. Let's consider for example the following code:

```
….
If ($a =~ m/knowledge/)
{ // branch 1
     ….
}
else
{ // branch 2
     …
}
….
```

This code tests the value of *$a*; if it matches the regular expression */knowledge/*, the code contained inside the first branch is executed, otherwise the code contained inside the second branch is executed.

The matching operator, **m/..../**, tries to decide as soon as possible whether or not the string contained in the variable *$a* matches the expression between the two // characters. Let's consider an example by assuming that *$a* equals:

```
"Perl is a scripting language. But what's a scripting language?"
```

and that the regular expression is:

```
"language".
```

You could imagine the checker works as follow. The checker scans the regular expression and finds its first character – in this case it is 'l'. The checker then places itself at the start of the string to be checked, ie *$a*, and scans it from left to right until it reaches the first occurrence of the 'l' character (because the first character in the regular expression was 'l'). In our example the first 'l' is the one at the end of 'Perl'. Then the checker scans the regular expression and finds its second character, 'a'. Now it continues scanning the value of *$a* and finds that the character following 'l' is a blank. ' ' and 'a' are not the same and this means that the regular expression could not be matched. So the checker continues scanning *$a*, but first comes back to the beginning of the regular expression. By following this algorithm the checker will be able to match the regular expression after reading the word 'language' inside *$a*. By then the checker will have established that *$a* matches the regular expression */language/* and will return.

CHARACTERS

The most frequently used special characters are shown below:

- \a – beep character
- \n – new line
- \r – carriage return
- \t – tab
- \e – escape
- \d – any character inside the set 0…9
- \D – any character not included inside the 0…9

- \w – any character inside a-z, A-Z, ‘_’, 0-9
- \W – any character not inside the \w set
- \s – any character inside the set ‘ ’, \t, \n, \r, \f
- \S – any character not inside the \s set
- \. – the character ‘.’
- . – any character other than \n.

CHARACTER CLASSES

Classes are specified inside opening and closing square brackets ‘[’, ‘]’. For example both [0-9] and [0123456789] are the same as \d, and [a-zA-Z_0-9] is the same as \w.

It should be noted that inside any class specification the character ‘–’ is used to specify a range: so if you want to specify the character ‘–’ itself you must use the syntax \–.

Any class can be ‘inverted’ by using the character ‘^’. For example:

```
[^0-9]\w
```

matches any string not beginning with a number. Again, \D is the same as /[^0-9]/.

You can specify:

- [<char#1><char#2>…<char#n>] – any character in the set specified.
- [<char#1>-<char#2>] – any character in the range specified.
- [^<char#1>…<char#N>] – any character not in the set specified.
- [^<char#1>-<char#2>] – any character not in the range specified.

For example:

1 k\d matches k19283464 but not kkkkkk.

2 \d\d:\d\d:\d\d\d\d matches 10:24:2000 but not Oc:24:2000.

3 \w\w\w\w\w\w\w\w\.\w\w\w matches any alphanumeric string in the format 8-dot-3.

Marco Pirini (Italy) © Xephon 2002

AIX cloning issues between MCA/PCI systems

I cloned an AIX 4.3. J50 microchannel system (7013-J50) to a F50 PCI system (7025). My clonebackup was on 8mm tape – and I booted from the AIX 4.3.3 ML=09/2001distribution.

Everything ran smoothly until the system rebooted. Then I got an LED 554 display. I consulted the *Common Diagnostics and Service Guide* (SA23-2687.02), Chapter 2 *Diagnostic Numbers and Codes*. Under the topic *Configuration Program Indicators*, it said, "554 Unable to define NFS swap devices during network boot".

After rebooting in service mode and checking the */etc/inittab*, */etc/rc.net*, */etc/rc.tcpip*, */etc/rc.nfs*, and the */etc/swapspaces* files, I found no reason why I should get the LED 554 display.

I contacted the AIX Hotline, who sent me three html documents – *Recovery from LED 552,554...*, *How to Clone an AIX 4.2 and 4.3 mksysb Tape*, and *How to Clone a mksysb Image Using NIM....*

Because I have successfully cloned other systems from MCA to PCI architecture in the past without any problems I was sure that the system should find all the necessary drivers from the boot CD-ROM to build a bootable kernel.

The system booted in service mode. I searched and found a very useful file in the */var/adm/ras* directory. The file name was *bosinstlog*, and in it I found a section telling me what the system was missing.

Here is the important section:

- Missing filesets:

  - devices.pci.pciclass.010000
  - devices.pci.scsi
- Requisite Failures ....
  - devices.pci.00100300.rte 4.3.3.25

    # PCI 16-bit SCSI I/O Control...
- CONFLICTING REQUISITES:
  - devices.common.IBM.scsi.rte 4.3.3.75

    # Common SCSI I/O Controller S...

Status of the filesystems and mounts:

```
# df -k
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/ram0           57344     22244   62%     2340    17% /
/dev/cd0
df: /SPOT: No such file or directory
/dev/hd4            57344     22244   62%     2340    17% /
/dev/hd2           901120    390204   57%    20601    10% /usr
/dev/hd3           131072     39812   70%      711     3% /tmp
/dev/hd9var         49152     40504   18%      373     4% /var
```

I unmounted the */SPOT* filesystem so that I could remove the CD-ROM and insert the other one, if the smit install screen advised me to do so.

Then I did a **smit devices**, **Install/Configure Devices Added After IPL** and selected the **cdrom** as the input device.

The system then found the missing device drivers, and the next reboot succeeded.

Imhotep
Unix System Administrator (Austria) © Xephon 2002

Quoting

WHY DO WE QUOTE

There will be many situations when you want to pass a metacharacter to a command as an argument, or as part of an argument, and you want it to be passed directly to the command and not interpreted by the shell.

Suppose, for example, you wanted to look for the name **Tonto Kowalski** in the **passwd** file. If you entered:

```
grep Tonto Kowalski /etc/passwd
```

then **grep** would look for the word **Tonto** in the files *Kowalski* and */etc/passwd*.

If you wanted the whole expression **Tonto Kowalski** treated as a single argument, then you must indicate to the shell that the space between **Tonto** and **Kowalski** is not to be treated as a word separator, but should be passed on to the **grep** command as part of the search string.

You can instruct the shell to pass a metacharacter without interpreting it, by *quoting* the metacharacter. This can be achieved by enclosing it between single quotes ('), or by preceding it with a backslash (\).

Most metacharacters can also be quoted by enclosing them in double quotes ("), which act much like single quotes, except that there are some metacharacters that cannot be quoted with double quotes. Double quotes are covered later in the section *Partial quoting*.

Quoting is used extensively in shell programming and can be the source of seemingly endless frustration! Different commands use quotes in different ways and often only extensive experimentation will produce the required result. If at first you don’t succeed....

THE BACKSLASH

The backslash, \, is a metacharacter which quotes, or *escapes*, the character which immediately follows it. Like other metacharacters, \ itself is not passed to the command, and so the pair, **\x**, where **x** is any character, really just stands for the character **x** itself.

Here are some examples:

```
$ print \*
*

$ print \*\?
*?

$ print here \>there
here >there

$ rm all\*
```

In each of the above examples, the metacharacter following the backslash is not interpreted as it normally would be. Instead it is passed directly to the command.

Similarly, if you wanted to include a backslash in an argument to a command, you would also have to quote it with another backslash. For example:

```
$ print \\
\

$ print \\z
\z
```

Note that when a character that is not a metacharacter is quoted, the quoting has no effect. For example:

```
$ print \a
a

$ print \hello
hello
```

ESCAPE SEQUENCES

There are a number of commands, including **echo**, **print**, and **printf**, which do something special when they receive a backslash as an argument. Under certain circumstances, these commands interpret, as an example, the pair of characters **\n** to mean a new line character. These combinations of \ and other characters are known as *escape sequences*.

To ensure that the \ will not be treated just as a metacharacter, but will be passed on to **print** so that it can interpret the pair, **\n**, as a new line character, we must either *escape* the backslash with a further backslash, or enclose the combination with either single or double quotes (more on these last two later).

The following examples illustrate how the escape sequences work, and what happens if you do not escape the backslash:

```
$ print '\n'  or  print "\n"
<blank line>

$ print \\n
<blank line>

$ print \n
n
```

In addition to **\n**, which is equivalent to using the key combinations **CTRL J**, the following are also escape sequences, interpreted as follows:

- **\a** alert (rings a bell usually) – **CTRL G**.
- **\b** backspace – **CTRL H**.
- **\c** omit the final new-line.
- **\r** carriage return <Enter> – **CTRL M**.
- **\t** tab – **CTRL I**.
- **\0*n*** where *n* can be from 1 to 3 digits. This combination prints the associated ASCII character with octal (base 8) value *n*. We have already seen a previous example of this using the **tr** command, where we used **\012** to be interpreted as **\n** – see *Examples of shell scripts*, *AIX Update*, Issue 78, April 2002.

The only problem with the previous escape sequence is converting the ASCII decimal value to its octal value, which means you must already know the decimal value for the character. The occasions when you have to do this will be, hopefully, few and far between.

SINGLE QUOTES

You can use single quote marks to enclose a string of characters when you want to turn off the special meaning of all the enclosed characters. The effect is the same as if each of the characters were preceded by a backslash. The quote marks themselves are not part of the argument that is passed to the command.

Here are some examples:

```
$ print '*'
*

$ print '* * * *'
* * * *

$ rm 'all*'
```

If you want a single quote mark to be passed as an argument, or included in an argument, you must quote it. A single quote mark can be quoted with a backslash, but you cannot use ' ' ' to quote a single quote mark. For example:

```
$ print the letter \'a\'
the letter 'a'

$ grep Don\'t myfile
Don't look back
```

Likewise, a backslash may be quoted by enclosing it in single quotes, as we have mentioned above in the section on escape sequences.

EXAMPLES USING SINGLE QUOTES

Consider the following examples, where we use the backslash and ***** character to give you some idea of how the location of the surrounding single quotes can produce such different interpretations:

```
$ print '\*'
\*
```

Everything appearing between the single quotes is passed on without interpretation so that the argument passed to **print** is **\***.

```
$ print \*
*
```

The backslash quotes the metacharacter *****, so the argument passed to **print** is the character *****.

```
$ print '\'*
\*
```

The backslash has been escaped, so it will not quote the *****. Thus the argument becomes a filename generation pattern and the shell looks for filenames that match the pattern \ followed by any number of characters, and, since it is unable to find any, it passes the argument to **print**. This is equivalent to using filename generation expressions such as **ls f***.

```
$ print '\' *
\ bin log lib myfile
```

There are two arguments in this command line: **'\'** and *****. The backslash is quoted so it is passed directly to **print**. The ***** is not quoted, so it is treated as a pattern and the resultant list of file names is passed to **print**.

PARTIAL QUOTING

Double quotes act in an almost identical way to single quotes, except that there are certain metacharacters that do not lose their special meaning when they appear between double quote marks.

Metacharacters that *do* lose their special meaning include spaces, tabs, and filename generation characters. This means that spaces or tabs will not be used as word delimiters, and the entire quoted string will be treated as a single argument. Also the shell will not attempt to perform filename generation on the quoted string, even if it contains the *****, **?**, or **[ ]** metacharacters.

The metacharacters that *do not* lose their special meaning include **$**, \, and **‘** (backquote). In other words, the shell will perform variable and command substitution on the string that appears between the double quote marks. Here is an example:

```
$ author='William Shakespeare'
$ grep "$author" dict1 dict2
dict2: Romeo and Juliet was written by William Shakespeare
```

The value of the variable **author** is substituted, but the result is treated as a single argument. If you were to use single quote marks around

$author, the value of **author** would not be substituted, and **grep** would look for the string **$author** instead. Also, consider what would happen if you did not quote **$author** at all:

```
$ grep $author dict1 dict2
grep: can't open Shakespeare
dict2: Romeo and Juliet was written by William Shakespeare
```

In this case, the value of **author** is treated as two separate arguments. The system firstly looks for the characters **William** in the file **Shakespeare**, and, since this file does not exist, the error message that it can't open the file is displayed.

The single quote mark is another metacharacter that does lose its special meaning between double quote marks, as in the following example:

```
$ print "Don't go"
Don't go
```

Note that if you just enter:

```
$ print Don't go
>
```

the shell displays the secondary prompt because it expects to see a closing single quote.

The backslash does not lose its meaning between double quotes, which means that it can be used to quote the metacharacters **$** and **‘** (backquote). For example:

```
$ item=widgets
$ print "The price of $item is \$4.20 per kilo."
The price of widgets is $4.20 per kilo.
```

The value of the variable **item** will be substituted, because double quote marks do not prevent variable substitution. We must precede the second **$** with \ to prevent the shell from substituting the value of the fourth positional parameter.

COMMENT CHARACTER –

Comment characters are used to cause a string of characters to be ignored by the shell. This allows you to insert comments in a shell script explaining its purpose or construction.

Whenever the character **#** appears in a line in an executable shell script, apart from the previously mentioned exception where the first line in a file starts with the **#!** combination, that character and everything up to the end of the line are ignored.

You can put a comment on a line by itself by starting the line with **#**. You can also put a comment on the same line as a command by following the command with **#**. Normally you put a space between the end of the command and the comment character. This is always recommended because some commands may interpret the **#** as part of the command, or as part of an argument, and so complain.

You should be aware that there are many files containing comment lines that do not start with a **#**. These are usually administrative files used by other commands and are not shell scripts; they are treated as pure text files. Typical comment lines in these files begin with characters such as **:**, *****, or **!**.

Tonto Kowalski
Guru (UAE)

AIX memory management

Memory is made up of pages, which can be either in RAM, where they are immediately available, or stored on disk in what's called virtual storage. A page is 4096 bytes in size. Both working pages and file pages are stored.

Virtual storage is storage that appears to be available, as far as applications are concerned, but isn't! If a program needs a page then it is looked for in real storage (RAM). If it is not there, but a copy is available on disk, then it must be copied into real storage. This is called a page fault. Each page has an address in virtual storage: if a wanted page's address does not translate to a real memory address, the Virtual Memory Manager (VMM) gets the page from disk and places it in RAM. Moving a page from disk to RAM is referred to as a 'page-in', and moving it from real memory to disk is called a 'page-out'.

But before paging in, VMM first has to check that there is space in RAM in which to load the page.

If there is enough room, VMM checks to see whether the wanted page has been used previously by this process. If it has, a 'repage fault' I/O is scheduled to bring the page back from disk and into RAM. The process that requires the page-in cannot be doing any work until the page is back in RAM. It is, therefore, said to be in a 'page wait state'.

If the wanted page has not been previously used by the process asking for it, then an 'initial page fault' occurs. When this happens, VMM allocates two pages for the data – one in RAM and the other on disk. This second one will be used after a page-out occurs. It is referred to as 'late page space allocation'.

However, on a busy AIX machine where a lot of page-in and page-out operations have been occurring through the day, it's very likely that there won't be any space in RAM for the page-in operation to page to. Somehow, AIX has to make a space. It does this with a 'page stealer', which ensures that there is always a supply of free RAM pages available for an initial page fault. A value is specified and the page stealer makes sure that the number of free RAM pages doesn't drop below that level.

The page stealer chooses the Least Recently Used (LRU) pages. It then has to check whether the contents of the page match the second copy on disk. If they do match (the page is described as ‘clean’) the page in RAM is purged. If the page has been modified in RAM (called a ‘dirty’ page), it is written to a backing store (either page space or a filesystem). A page space is used for non-persistent or working pages, and the filesystem is used for persistent or file pages.

However, it’s slightly more complicated than this! The *maxperm* and *minperm* threshold values are specified. Increased paging activity makes VMM act upon the different types of (stale) pages in a different manner. When the number of stale file pages exceeds the *maxperm* threshold number, the page stealer steals only file pages.

When the number of stale file pages falls below *maxperm* (but above the set *minperm* threshold) then two other considerations come into play. The VMM checks the repage rates of both file and nonfile pages, and then steals file pages if the file page repage rate is higher than the repage rate for nonfiles.

If this is not the case, then both types of page are treated equally as targets.

Page out I/O occurs only when a page is stolen by the page stealer *and* is flagged as ‘dirty’. This happens only when there is a shortage of free RAM pages. Hence the page-out figure can be an indicator of how memory-constrained the system is. The **vmstat** command is only of limited use because it reports just activity concerned with page space (and not paging to/from filesystem space).

If the system consistently appears to hover around the *minperm* value, then the system is not necessarily memory constrained. It may be the case that an initial page fault is resolved by purging a clean, but stale, page. In this case there is paging activity but no corresponding I/O.

Page faults don’t (necessarily) result in disk activity – only the repage fault (the act of bringing back previously used data into memory) causes disk I/O to be scheduled.

System performance may be improved by reducing the amount of

RAM that file pages occupy – this ensures that working pages are not continually being pushed out to make way for file pages. This can be done by the **vmtune** command (/usr/samples/kernel) and lowering the values for *minperm* and *maxperm*.

There are tried-and-trusted rules of thumb for calculating the amount of page space required:

- For systems that have up to 256MB of real memory:

  page_space = 2 x real_memory

- For systems with more than 256MB of real memory:

  page_space = 512MB + (real_memory - 256MB) * 1.25

Other useful rules include:

1 Configure just one paging space per disk.

2 Use between two and six paging spaces in a medium size system.

3 Configure the paging spaces on each disk to be the same size.

Christopher Chapman
AIX Specialist (UK)

Creating a RAM filesystem

We wanted to create a RAM filesystem in AIX on one of our machines, and came up with the following procedure, which others may find of use.

Firstly we entered the command **mkramdisk** followed by the number of 512KB blocks we needed. So, we wanted 12MB of RAM filesystem, which worked out at 24,576 blocks. The command looked like:

```
mkramdisk 24576
```

Next we needed to create the filesystem and mount the RAM disk. The commands for this are:

```
mkfs -V jfs -o "log=nointegrity,nbpi=512" /dev/ramdisk0
mount -V jfs -o nointegrity /dev/ramdisk0 /mnt
```

When our testing was over, we had to unmount the disk and remove the RAM disk we'd previously created. The commands to do this were:

```
unmount /mnt
rmramdisk /dev/ramdisk0
```

Stephen Hare
AIX Systems Expert (USA)

Ensuring space is available within a filesystem

Running out of space is a common problem, and often when filesystems fill up the knock-on effects can result in numerous other problems. Filesystems used for logging and trace files have a tendency to use large amounts of space in a short period of time. For these it is important to regularly run housekeeping scripts to remove old files. This can be done easily by putting a simple **find** command in root's crontab file. For example:

```
ØØ ØØ * * * /bin/find /LOGS -type f -mtime +7 -exec /bin/rm -f {} \;
```

will delete all files in */LOGS* that have not been modified for more than a week. If you are really pushed for space you can reduce to a day the period of time for which files are kept.

However this strategy will never pick up log files produced by long-running processes that are continually being updated. Also it will not pick up fast growing files.

The following script has been written to address this problem. It first clears down files by age, until only the files under a specific number of days are left. It then switches strategy and clears down files in order of size starting with the largest first. In this way rapidly growing files are caught, as well as those being continuously updated.

```
#!/usr/bin/ksh
#
# Script: spacePanic
# Author: Roger Wickings
# Date:   24th October 2ØØ1
# Aim:    Ensure a filesystem does not fill up
#

awk="/usr/bin/awk"
basename="/usr/bin/basename"
cat="/usr/bin/cat"
date="/usr/bin/date"
df="/usr/bin/df"
find="/usr/bin/find"
grep="/usr/bin/grep"
head="/usr/bin/head"
ls="/usr/bin/ls"
```

```
rm="/usr/bin/rm"
sed="/usr/bin/sed"
sort="/usr/bin/sort"
tail="/usr/bin/tail"
touch="/usr/bin/touch"
tr="/usr/bin/tr"

TMPDIR="/tmp"

DEFAULTPCT="80"
DEFAULTDAYS="1"

#
# Functions
#

abend()
{
  echo "$SCRIPT: $* " 1>&2
  exit 1
}

checkFileSystem()
{
  check='$df -k $FILESYSTEM |
           $grep "/" |
           $awk '{print $NF }' |
           $tail -1 '
  if test "$check" != "$FILESYSTEM"
  then
    abend "Directory $FILESYSTEM is not a filesystem"
  fi
  return
}

checkParms()
{
  count=0
  for parm in $PARMS
  do
    case "$parm" in
      -\?)    usageError               ;;
      debug)  DEBUG="debug" ; continue ;;
      *)      :                        ;;
    esac

    check='echo "$parm" | $grep "^[^=][^=]*=[^=][^=]*$" '
    if test "$check" != ""
    then
 keyword='echo "$parm" | $awk -F= '{print $1 }' | $tr "[A-Z]" "[a-z]" '
```

```
      value='  echo "$parm" | $awk -F= '{print $2 }' '
      case "$keyword" in
        deldays)  DELETEDAYS='echo "$value" | $sed "s,[^Ø-9],,g" '  ;;
        *)        abend "Invalid option $parm"                      ;;
      esac
      continue
    fi

    let count=$count+1
    case "$count" in
      1) FILESYSTEM="$parm"                          ;;
      2) LIMIT='echo "$parm" | $sed "s/[^Ø-9]//g" '  ;;
      *) abend "Invalid option $parm"                ;;
    esac
  done
  if test "$FILESYSTEM" = ""
  then
    abend "Filesystem not specified"
  fi
  if test ! -d "$FILESYSTEM"
  then
    abend "Directory $FILESYSTEM not found"
  fi
  if test "$LIMIT" = ""
  then
    LIMIT="$DEFAULTPCT"
  fi
  if test "$DELETEDAYS" = ""
  then
    DELETEDAYS="$DEFAULTDAYS"
  fi
  return
}

checkSpace()
{
  if test "$SPACETYPE" = "pcused"
  then
    space='$df -k $FILESYSTEM |
             $grep "/" |
             $awk '{print $4 }' |
             $sed "s/[^Ø-9]//g" |
             $head -1 '
    if test "$space" = ""
    then
      abend "Can not determine space left in filesystem $FILESYSTEM"
    fi
    if test "$DEBUG" = "debug"
    then
      echo "$SCRIPT: debug: space=$space  LIMIT=$LIMIT"
    fi
    if test $space -lt $LIMIT
    then
      exit Ø
    fi
  else
    space='$df -k $FILESYSTEM |
             $grep "/" |
             $awk '{print $3 }' |
             $sed "s/[^Ø-9]//g" |
             $head -1 '
    if test "$space" = ""
    then
      abend "Can not determine space left in filesystem $FILESYSTEM"
    fi
    if test "$DEBUG" = "debug"
    then
      echo "$SCRIPT: debug: space=$space  LIMIT=$LIMIT"
    fi
    if test $space -gt $LIMIT
    then
      exit Ø
    fi
  fi
  return
}

clearDown()
{
  file="$1"
  $ls -l $file
  $touch -fmr $file $REFFILE
  $cat /dev/null > $file
  $touch -fmr $REFFILE $file
  $rm -f $REFFILE
  return
}

deleteByAge()
{
  daylist='(
             $date +"%Y %j"
             $find $FILESYSTEM -type f -ls
           ) |
             $awk '
                  NR == 1 {
                            curryear = $1
                            currjuln = $2
                            currepoc = currjuln + (curryear-197Ø)*365 +
int((curryear-1969)/4)
```

```
                        }
                NR != 1 {
                          inmonth = $8
                          inday   = $9
                          check = index( $10 , ":" )
                          if ( check == 0 ) { fileyear = $10 }
                          else              { fileyear = curryear }
                          if ( fileyear%4 == 0  ) { leapyear = 1 }
                          else                    { leapyear = 0 }
                         if ( inmonth == "Jan" ) { filejuln = inday }
                   if ( inmonth == "Feb" ) { filejuln = inday +  31 }
        if ( inmonth == "Mar" ) { filejuln = inday +  59 + leapyear }
        if ( inmonth == "Apr" ) { filejuln = inday +  90 + leapyear }
        if ( inmonth == "May" ) { filejuln = inday + 120 + leapyear }
        if ( inmonth == "Jun" ) { filejuln = inday + 151 + leapyear }
        if ( inmonth == "Jul" ) { filejuln = inday + 181 + leapyear }
        if ( inmonth == "Aug" ) { filejuln = inday + 212 + leapyear }
        if ( inmonth == "Sep" ) { filejuln = inday + 243 + leapyear }
        if ( inmonth == "Oct" ) { filejuln = inday + 273 + leapyear }
        if ( inmonth == "Nov" ) { filejuln = inday + 304 + leapyear }
        if ( inmonth == "Dec" ) { filejuln = inday + 334 + leapyear }
   fileepoc = filejuln + (fileyear-1970)*365 + int((fileyear-1969)/4)
               if ( currepoc > fileepoc ) { print currepoc-fileepoc }
                        }
               ' |
          $sort -unr '

if test "$DEBUG" = "debug"
then
  ddaylist='echo "$daylist" | tr "\n" " " '
  echo "$SCRIPT: debug: daylist=$ddaylist"
fi

for daysold in $daylist
do
  if test $daysold -lt $DELETEDAYS
  then
    break
  fi
  if test "$DEBUG" = "debug"
  then
  echo "$SCRIPT: debug: Deleting files which are $daysold day(s) old"
  fi
  for file in '$find $FILESYSTEM -type f -mtime +$daysold -print '
  do
    clearDown $file
    $rm -f    $file
  done
  checkSpace
done
```

```
  return
}

deleteBySize()
{
  if test "$DEBUG" = "debug"
  then
    echo "$SCRIPT: debug: Starting deleteBySize"
  fi

  $find $FILESYSTEM -xdev -type f -ls |
    $awk '{ print $7 ": " $NF }' |
    $sort -t: +Ø -1 -nr |
    $awk '{print $2 }' |
    (
      while read bigfile
      do
        clearDown $bigfile
        checkSpace
      done
    )

  return
}

initialization()
{
  checkParms
  setSpaceType
  checkFileSystem
  checkSpace
  REFFILE="$TMPDIR/$SCRIPT.$$.reffile"
  export LANG=en_US
  return
}

setSpaceType()
{
  check='echo "$LIMIT" | $sed "s/[Ø-9]//g" | $tr "[A-Z]" "[a-z]" '
  value='echo "$LIMIT" | $sed "s/[^Ø-9]//g" '
  case "$check" in
    k)   SPACETYPE="kbfree"
         LIMIT=$value
         ;;
    m)   SPACETYPE="kbfree"
         let LIMIT=$value*1Ø28
         ;;
    *)   SPACETYPE="pcused"
         LIMIT=$value
        ;;
  esac
  if test "$DEBUG" = "debug"
  then
    echo "$SCRIPT: debug: spacetype=$SPACETYPE"
  fi
  return
}

usageError()
{
  abend "Usage $SCRIPT \$FILESYSTEM [\$PERCENT|\$FREE[m|k]]
[deldays=\$DAYS] [debug]"
}

#
# Start of main processing
#

SCRIPT='$basename $Ø '
PARMS='echo "$*" | $sed "s,\",,g" '

initialization

deleteByAge
deleteBySize

exit Ø
```

```
The options for this script are displayed by typing the following

$ spacePanic -?
spacePanic: Usage spacePanic $FILESYSTEM [$PERCENT|$FREE[m|k]]
[deldays=$DAYS] [debug]
```

The filesystem which needs housekeeping must be specified. The second parameter is the watermark to which the filesystem will be cleared down. This can be specified as the maximum percentage used, or as an absolute amount of free space in either megabytes or kilobytes. The free space option is useful if you want to ensure that a particular amount of space is available before running a specific task. For example we run the following before performing a **mksysb**:

```
$ spacePanic /tmp 9m
```

The default watermark is 80% used, but this can easily be amended as it is specified at the top of the script.

The **deldays** parameter is used to specify the minimum age at which files can be deleted. Once all files over this age have been deleted, the script starts processing files in order of size, starting with the largest. However it does not delete these files but clears them down to zero bytes. This is important when a process has a file open. Removing the file will not free up any space, it will just make the file invisible, and the file will continue to occupy space until it is closed.

Hence the following crontab entry could be used for housekeeping a log file directory:

```
00 *  * * * /bin/spacePanic /LOGS 75
```

Roger Wickings
Systems Programmer
FT Interactive Data (UK)

Program execution

EXECUTION OF COMPILED PROGRAMS

When the shell reads the name of a command, it instructs the operating system to start a new process. The name of the system request used to start a new process is called **fork**. Note that **fork** is not a command that you can enter from your keyboard, or use in a shell script. It is an operating system request that compiled programs, such as the shell, can make.

As the name **fork** suggests, this system call causes the calling process to be duplicated. After the shell calls **fork**, there are two shell processes running: the process which called **fork** is called the **parent** process, and the new process is called the **child** process.

The first thing the child process does is to request that the operating system overlays it with another executable program. The child does this by using the system request **exec**. After **exec** has been called, the child process will no longer be a shell process, but will instead be a **vi** process, **pg** process, or **cat** process, depending on the command you entered.

In the meantime, the parent process (your login shell) waits idly (sleeps) for the child to complete. When the child process terminates, the system re-activates the parent, the login shell displays a prompt, and the cycle can then begin again.

As an example, consider what happens when you enter the command **vi memo**. The login shell first issues a **fork** system request and, as a result, a new shell process is created. This new shell process then issues the **exec** system request, and at the same time the child process passes to the system the name of the program to be run.

In this case the child process instructs the system to run the **vi** program in its place, and so the child process is overlaid with **vi**. The parent process then waits for it to complete.

The way in which this happens is shown in the following diagram. We have shown the parent process as **-ksh**, but in a windows environment this will be **/bin/ksh**.

BACKGROUND JOBS

When you run a process in the background, the process is started as previously described. However, the parent process does not wait for the child to complete. Instead, it immediately issues another prompt, reads the next command, and calls **fork** to create another child process. This child calls **exec**, and the new command runs as well.

At this point, unless the background job has already completed, there are three separate processes running:

- The background job.
- The new command.
- The login shell.

Depending on whether or not the new command is a background job, the shell either waits for it to complete or goes on to accept yet another command.

EXECUTION OF BUILT-IN SHELL COMMANDS

You will remember that the name of a command is the name of a file that contains the shell script or compiled program, and that the shell uses your search path to locate the program to be run. This is not the case for built-in shell commands.

A built-in shell command is executed by the shell process itself and no new processes are started. When the shell reads the name of a command, it first checks to see whether the command is built-in. If not, it then uses the search path to locate the command.

There are a small number of built-in commands, some of which you are probably already familiar with, such as **echo**, **cd**, **export**, and **set**. In the latest versions of AIX, **echo** and **cd** are also executables located in */usr/bin*; try running **which** and **type** (or **whence -v**) against the four commands above and note the output. You do not normally need to know whether a particular command is built-in or not.

An important point about the **cd** command is that it changes only the current directory of the calling process, and the descendants of that process.

This means that if you run a shell script containing the command **cd /tmp**, the current directory will be */tmp* for the remaining commands in the shell script, but when the script completes, your current directory will not be changed. The **cd** command does not affect the parent of the process that executes **cd**.

Try creating a simple shell script that contains a **cd** command followed by an **ls** or **pwd** command to verify that the current directory has been changed. After you run the shell script, use the **pwd** command to verify that your current directory is the same as it was before you ran the shell script.

EXECUTION OF SHELL SCRIPTS

When you are interacting with your login shell, you can start another shell by entering the **ksh** command. This shell process is referred to as a sub-shell of your login shell. The sub-shell displays a prompt and accepts commands just like the login shell does, because it is the same program as your login shell. If you now use **ps**, you will see that there are two shell processes running on your terminal.

If you were to keep entering **ksh**, you would continue to create new shell processes, and the previous shell processes would wait for their descendants to complete.

You can terminate the current shell in the same way you terminate your login shell, by pressing **Ctrl D**, or entering **exit**. When the current shell dies, the parent is re-activated, displays a prompt, and continues to accept commands.

When you enter a command such as:

```
ksh filename
```

where *filename* is the name of a shell script, a new shell process is started.

Instead of reading commands you enter from your keyboard, this shell process reads commands from *filename*; the file does not need to have execution mode for this to happen. When it has executed the last command in the file, the shell process terminates.

The above command is also identical to:

```
ksh < filename
```

When you run a shell script by just entering its name on a command line, the **fork** and **exec** system requests are issued, and the name of the script is passed to the system as the program to be run. At this point the system recognizes that the file is not a compiled program, and since a shell script is not a machine language file, the system cannot simply overlay the child process with the script.

Instead, the system overlays the child process with the **ksh** program, and this new process interprets the commands in the shell script.

Whether you run a shell script by entering **ksh *filename***, or **ksh < *filename***, or just ***filename***, the net result is the same. A new shell process is started, which interprets the commands in ***filename***, and terminates after the last command in ***filename*** has been run.

Tonto Kowalski
Guru (UAE)

Achieving I/O dynamic load balancing and availability with EMC PowerPath software

EMC PowerPath is a software layer that enables efficient management of I/O for AIX-based servers connected to Symmetrix storage systems. Its functionality is similar to HP's Autopath software, which has been described in my article *Implementing I/O multipathing using Autopath*, Issues 68 and 69 of *AIX Update*, June and July 2001.

PowerPath provides the following features and benefits:

1 Dynamic load balancing – utilization of multiple I/O paths connecting between the host and an EMC volume. The PowerPath software monitors the load caused by server I/O and dynamically distributes it between available I/O paths according to pre-defined policies.

2 Automatic path failover – automatic redirection of I/O requests to available I/O paths when some of them became unavailable, enabling the applications to function without downtime.

3 Automatic on-line recovery – execution of periodic tests to check for repair of failed I/O paths. If a path passes a test, PowerPath returns that path to service and resumes sending I/O to it.

4 Consistency groups – consistency group support provides support for maintenance of the integrity of databases distributed across multiple Symmetrix remote data facility units. It is used for remote mirroring of data.

COMPATIBILITY AND INSTALLATION REQUIREMENTS

Compatibility and installation requirements are:

1 IBM RISC/6000, IBM pSeries, or Bull Escala system.

2 Two to 32 host-based adapters supporting fat-wide differential, ultra-wide differential SCSI, and/or two to 16 Fibre Channel host-based adapters utilizing an arbitrated loop topology using direct connections or hubs, or fabric topology using switches.

3 Single-initiator configuration on SCSI, single- or multi-initiator on Fibre Channel.

4 AIX version 4.2*x* or higher for SCSI adapters, AIX 4.3 or higher for EMC Fibre Channel Interface, AIX 4.3.3 or higher with APAR IY12528 for IBM Fibre Channel drivers.

5 5MB of disk space for PowerPath product files.

6 AIX system administrator privileges.

Proper configuration of hdisk devices should be assured before the installation of PowerPath software on the AIX host. Each logical path that PowerPath will use to access a Symmetrix device must have an hdisk device available.

For example if Symmetrix is configured with four logical volumes and each one of them is connected using four different ports, each one of which is attached to an FC adapter on an AIX host, then the host should have 16 Symmetrix hdisk devices available.

If the number of available hdisks differs from your definitions, you should perform the following procedure:

1 Make sure that all physical devices are properly connected.

2 Execute the following command to remove all hdisks corresponding to Symmetrix devices:

```
lsdev -CtSYMM* -Fname | xargs -n1 rmdev -dl
```

3 Configure all hdisks by executing the script **emc_cfgmgr.sh**. This script invokes the AIX **cfgmgr** utility to probe each adapter bus in order to identify all connected devices. This script is available in the AIX directory of the installation CD as well as being downloadable from the following URL: ftp://ftp.emc.com/pub/symm3000/powerpath/aix/emc_cfgmgr.sh.

INSTALLATION INSTRUCTIONS

During installation of the PowerPath software, each Symmetrix volume connected to AIX becomes represented by a single hdiskpower device, which has to be used subsequently to access the volume. All volume groups, as well as applications that are accessing hdisk devices directly, have to be reconfigured to use PowerPath hdisk devices instead.

The following actions have to be performed before the installation of the software:

1 Stop all applications using volume groups that contain Symmetrix hdisk devices.

2 Unmount all filesystems that belong to these volume groups.

3 Varyoff the volume groups.

4 Edit the file */etc/powerpath_registration*, inserting the registration number that arrived with the installation medium.

5 Mount the CD-ROM by executing the following command:

```
mount -v cdrfs -p -r /dev/cd0 /mnt
```

6 Change your current directory according to OS level and the type of your connection to Symmetrix:

– If you are running AIX 5.*x*, perform the command:

```
cd /cdrom/AIX/aix5
```

– If you are running AIX 4.*x* and using either IBM SCSI adapters or EMC Fibre Channel Interface for AIX platform adapters, perform the command:

```
cd /mnt/AIX/standard
```

– If you are running AIX 4.*x* and using either IBM Fibre Channel adapters, perform the command:

```
cd /cdrom/AIX/fcp
```

7 Install the software by executing the following command:

```
installp -ac -gqX -d . EMCpower
```

8 After conclusion of the installation, the system will display informational messages specifying that installation of the following filesets has been successful:

- EMC power.base
- EMCpower.multipath
- EMCpower.consistency_grp.

9 At this stage you should validate your registration by executing the command:

```
powermt check_registration
```

You should receive the following response:

```
Registration is valid.
```

10 Unmount the CD by executing the following commands:

```
cd /; unmount /mnt
```

CONFIGURATION INSTRUCTIONS

The following command has to be executed in order to initialize PowerPath hdiskpower devices and make them available to the host:

```
powermt config
```

The command **powermt** is installed in the directory */usr/sbin*; it is the main command that is used to operate the PowerPath software. Execution of this command with the **config** argument performs scanning of all paths to Symmetrix volumes, which are identified according to a 12-digit Symmetrix frame serial number and a 3-digit volume serial number. Together these values uniquely identify a Symmetrix volume. Execution of this command does not configure any Symmetrix volumes or paths that are not functioning at the time the command executes. Nor does it remove any previously configured paths that are not functioning at the time it executes.

On AIX hosts, PowerPath is configured at boot time using ODM configuration rules. You need to run the **powermt config** command only if additional device paths are added after boot time, for example following re-connection of disconnected cables.

After the installation of PowerPath, you may need to configure volume groups or applications to use PowerPath hdiskpower devices. During the installation, existing volume groups that use hdisks residing on

Symmetrix are converted to use corresponding hdiskpower devices. You just need to vary on the volume groups and mount the corresponding filesystems. Existing applications that are directly accessing the hdisk devices will have to be reconfigured to use the corresponding hdispower devices in order to utilize the advantages of PowerPath software.

hdiskpower devices should be used for the definition of new volume groups as well as for the configuration of new applications that are accessing disk devices directly.

USING THE POWERPATH SOFTWARE

Monitoring host adapters used by PowerPath

The status of host adapters used by PowerPath can be displayed using the following commands:

```
powermt display
powermt watch [every=seconds]
```

The watch command has an optional *every* argument, which specifies the number of seconds between the screen updates performed by the command.

The **powermt display** command displays a snapshot of the current state of the host adapters used by PowerPath.

The **powermt watch** command displays a similar report, refreshing the display at a specified interval, with 5 seconds being the default value. Type interrupt or quit signal to stop the command.

The following is an example of the output of the command on an AIX host:

```
# powermt display
Symmetrix volume count=14
========================================================
------ Host Adapters ---      -- Device Paths --  --- Stats ---
### HW-Path                 summary   total closed  IO/sec q-IOs errors
========================================================
  0 fscsi0                 optimal      12      0       -     0      0
  1 fscsi1                 optimal      12      0       -     0      0
```

Figure 1 gives an explanation of the fields that appear in the report.

Field	Value	Description
Symmetrix volume count	**Integer in the range between 0 and 2048**	**Total number of unique Symmetrix volumes configured by PowerPath software on the host.**
Host Adapters ###	**Integer between 0 and 255**	**Unique number assigned to host adapter during installation of PowerPath.**
Host Adapters HW-path	**Alphanumeric string**	**The adapter name from the ODM, such as fcp0.**
Device Paths Summary	**Optimal Degraded Failed**	**Status of paths originating from this adapter:** Optimal **all device paths originating from the adapter are open.** Degraded **some but not all paths originating from the adapter are open.** Failed **means that all paths originating from the adapter are closed and no data is passing through this adapter.**
Device Paths Total	**Integer in the range between 0 and 2048**	**Total number of paths that are originating through the adapter.**
Device Paths closed	**Integer in the range between 0 and *Device Paths Total***	**Total number of paths that are originating through the adapter and are out of service due to I/O errors.**
Stats IO/sec	**Non-negative integer**	**This field is blank in display produced by** powermt display **or first iteration of** powermt watch **command; subsequent iterations display the number of I/Os sent thru this adapter every second.**
Stats q-IOs	**Non-negative integer**	**Total number of uncompleted I/O operations sent to this adapter.**
Stats errors	**Non-negative integer**	**Total number of times this adapter has transitioned from open to closed states. This number is always equal to or less than the total number of I/O errors. This number is cleared during reboots and when** powermt restore **command is executed.**

Figure 1: Monitoring host adapters

Monitoring the PowerPath devices

The status of PowerPath devices can be displayed using the following commands:

```
powermt display dev=device|all
powermt watch dev=device|all [every=seconds]
```

The arguments of these commands are the specification of a particular PowerPath device, which can be specified using its full name, such as hdiskpower4, or just 4. Additionally the watch command has an optional *every* argument that specifies the number of seconds between the screen updates performed by the command.

The **powermt display dev** command displays a snapshot of the current state of the specified device or of all PowerPath devices on the host.

The **powermt watch dev** command displays a similar report, refreshing the display at the specified interval, with 5 seconds being the default value. Type interrupt or quit signal to stop the command.

The following is the abridged output of the command on an AIX host:

```
Pseudo name=hdiskpower3
Symmetrix frame ID=000184701412; volume ID=318
state=alive; policy=SymmOpt; priority=0; queued-IOs=0
==========================================================================
------ Host Devices ------  - Symm -  — Path —  — Stats —-
### HW-path            device     director  mode    state  q-IOs errors
==========================================================================
  0 fscsi0             hdisk10    FA  4a    active  open       0      0
  1 fscsi1             hdisk23    FA  3a    active  open       0      0

Pseudo name=hdiskpower4
Symmetrix frame ID=000184701412; volume ID=319
state=alive; policy=SymmOpt; priority=0; queued-IOs=0
==========================================================================
------ Host Devices ------  - Symm -  — Path —  — Stats —-
### HW-path            device     director  mode    state  q-IOs errors
==========================================================================
  0 fscsi0             hdisk11    FA  4a    active  open       0      0
  1 fscsi1             hdisk24    FA  3a    active  open       0      0
```

Figure 2 gives an explanation of the fields that appear in the report.

Field	Value	Description
Pseudo name	Alphanumeric string	hdiskpower*N*
Symmetrix frame ID	Hexadecimal Value	The unique 12-digit Symmetrix frame serial number.
Volume ID	Hexadecimal Value	3-digit Symmetrix volume serial number.
State	alive dead	State of Symmetrix volume: alive means that the volume has at least one active path dead means that the volume has no active path
Policy	RoundRobin LeastIos LeastBlocks SymmOpt REquest NoRedirect	Current load-balancing policy for the Symmetrix volume: RoundRobin distributes new I/O requests for each available path in turn, regardless of other factors. LeastIos distributes new I/O requests to the path with lowest number of requests, regardless of how many blocks are loaded. LeastBlocks distributes new I/O requests to the path with lowest number of queued blocks, regardless of how many requests are loaded. SymmOpt distributes new I/O requests to paths balancing the load according to path load and the path priority set by the operator. REquest no I/O load balancing is performed; PowerPath is used exclusively for failover. NoRedirect neither load balancing nor failover are enforced.
Priority	Integer between 0 and 10	Specifies the scheduling priority of the path to be used for Symmetrix-optimized load-balancing policy.
Queued-IOs	Non-negative integer	Number of I/O requests queued to specified Symmetrix volume, using specific path.

Figure 2a: Monitoring devices

Host Devices	Integer between 0 and 31	Unique number assigned to host adapter during installation of PowerPath.
Host Devices HW-path	Alphanumeric string	The adapter name from the ODM, such as fcp0.
Host Devices device	Alphanumeric string	The device name for the path. Such as hdisk0.
Symm	FA, SA	The type (FA-Fibre, SA-SCSI) of

Figure 2b: Monitoring devices

Checking the consistency of a PowerPath configuration

The consistency of PowerPath devices can be checked using the following command:

```
powermt check [force] adapter=adapter#|all [dev=device/all]
```

The *force* argument instructs the command to remove all paths that failed the connectivity test.

Adapter# limits the check to the adapter specified by the PowerPath adapter number that is shown in the ### column of the **powermt display** command. *All* specifies all adapters used by PowerPath.

Device limits the checks to the specified PowerPath device, indicated by using hdiskpower*N* or just the *N* for the number of the PowerPath device. *All* specifies all PowerPath devices.

The **powermt check** command tests I/O accessibility of each specified path as well as checking that there is a match for the recorded volume serial number. If either test fails, the command prompts the administrator to remove the path:

```
Warning: Symmetrix device path path_name is currently dead
Do you want to remove it (y/n/a/q)?
```

Valid responses are:

- Y – to remove the failed path and continue the testing.

- N – to leave the failed path configured and continue the testing.
- A – to remove the current path and any subsequent paths that fail the tests.
- Q – to leavethe failed path configured and exit the command. Any paths that have already been removed remain removed.

By default, the command scans all defined PowerPath devices for all system adapters. You can limit the checks for a specific adapter or specific PowerPath device.

Restoring paths to PowerPath management

You can restore the disconnected paths by using the following command:

```
powermt restore [force] adapter=adapter#|all [dev=device|all]
```

The force argument instructs the command to remove all paths that failed the connectivity test and to restore all paths that have passed the test.

Adapter# limits the check to the adapter specified by the PowerPath adapter number that is shown in the ### column of the **powermt display** command. *All* specifies all adapters used by PowerPath.

Device limits the checks to the specified PowerPath device, indicated by using hdiskpower*N*, or just the *N*, for the number of the PowerPath device. *All* specifies all PowerPath devices.

The **powermt restore** command tests the I/O accessibility of each specified path:

- If a path that has been opened passes the test, the command does nothing.
- If a path that has been opened fails the test, the command closes it and prints a warning:

  ```
  Warning: Symmetrix path path_name is currently dead.
  ```

- If a path that has been closed passes the test, the command reopens it.
- If all paths to a volume fail the path test, the command prompts the administrator to resurrect the volume:

```
Warning: Symmetrix device volume volume_name is currently dead
Do you want to resurrect it (y/n/a/q)?
```

Valid responses are:

- Y – to resurrect the failed volume and continue the testing.
- N – to leave the failed volume dead and continue the testing.
- A – to resurrect the current volume and any subsequent volumes to which all paths have failed the tests.
- Q – to leave the failed volume dead and exit the command. Any paths that have already been restored or volumes that have already been resurrected remain so.

By default, the command scans all defined PowerPath devices for all system adapters. You can limit the checks for a specific adapter or specific PowerPath device.

Removing paths from PowerPath management

You can remove obsolete paths by using the following command:

```
powermt remove adapter=adapter#|all [dev=device/all]
```

Adapter# limits the path removal to the adapter specified by the PowerPath adapter number that is shown in the ### column of the **powermt display** command. *All* specifies all adapters used by PowerPath.

Device limits the path removal to the specified PowerPath device, indicated by using hdiskpower*N*, or just the *N*, for the number of the PowerPath device. *All* specifies all PowerPath devices.

A path becomes obsolete when its associated physical devices (Symmetrix volumes and/or host adapters) have been removed from the system and its corresponding hdisk device has been removed from the system.

Setting the I/O priority of a PowerPath device

You can set the I/O priority of a PowerPath device by using the following command:

```
powermt set priority=# [dev=device|all]
```

is an integer between 0 and 10. The default is 0. The higher the number, the higher the priority setting. Setting all PowerPath devices to the same priority negates the effect of the command.

Device limits the priority change to the specified PowerPath device, indicated by using hdiskpower*N*, or just the *N*, for the number of the PowerPath device. *All* specifies all PowerPath devices.

This command is relevant only for PowerPath devices whose load-balancing policy is set to Symmetrix optimization. Although the command will set priorities for other PowerPath devices, it will have no effect on the load balancing on those devices.

If a specific PowerPath device is given a higher priority than all other devices, it tends to favour and dominate the first path that uses it.

Setting a high priority for a particular PowerPath devices improves its I/O performance at the expense of the remaining PowerPath devices. This feature is useful in cases where a system administrator is interested in favouring certain applications over others.

Setting the load-balancing policy of a PowerPath device

You can set the load-balancing policy of a PowerPath device by using the following command:

```
powermt set policy=policy [dev=device|all]
```

policy is one of:

- **rr** – distributes new I/O requests for each available path in turn, regardless of other factors. This policy imposes the least overhead on the I/O subsystem, but also has the least likelihood of maintaining balanced I/O over different paths.
- **li** – distributes new I/O requests to the path with the lowest number of requests, regardless of how many blocks are loaded.
- **lb** – distributes new I/O requests to the path with the lowest number of queued blocks, regardless of how many requests are loaded.

- **so** – distributes new I/O requests to paths balancing the load according to the path load and the path priority set by the operator. This is the default policy.
- **re** – no I/O load balancing is performed; PowerPath is used exclusively for failover.
- **nr** – neither load balancing nor failover is enforced.

Device limits the policy change to the specified PowerPath device, indicated by using hdiskpower*N*, or just the *N*, for the number of the PowerPath device. *All* specifies all PowerPath devices.

This command sets the load balancing policy for the specified PowerPath device or all PowerPath devices. Use the *device* argument to set the policy for a specific device. Any combination of policies can be chosen for various devices.

Setting the path mode of PowerPath adapter and/or device

You can set the path mode of a PowerPath device by using the following command:

```
powermt set mode=active|standby [adapter=adapter# dev=device|all]
```

Adapter# limits the mode change to a specific adapter. *Device* limits the mode change to the specified PowerPath device, indicated by using hdiskpower*N*, or just the *N*, for the number of the PowerPath device. *All* specifies all PowerPath devices.

This command is used to set a PowerPath path or a PowerPath device to active or standby mode.

For most applications, setting all paths leading to a specific device as **active** assures achievement of the best performance. Usage of this command can be helpful when isolation of I/O for different PowerPath devices can lead to improved performance.

Saving a custom PowerPath configuration

You can save the current PowerPath configuration to a text file using the following command:

```
powermt save [file=filename]
```

This command saves the following information – mode (alive or dead) for each configured path; serial number policy, priority, pseudo device node name – for each configured Symmetrix volume.

The default name of the saved file is */etc/powermt.custom*. Several files can be used to keep configurations for different workloads.

Restoring a custom PowerPath configuration

You can load a saved PowerPath configuration from a text file using the following command:

```
powermt load [file=filename]
```

This command applies the volume and path settings stored in the configuration file by the **powermt save** command to the current configuration.

CONCLUSION

PowerPath is very powerful software that greatly improves the usability of EMC Symmetrix storage devices in an AIX environment.

A future article will cover the additional features of PowerPath such as version upgrade and deinstallation, usage with boot devices, and integration with HACMP Cluster software.

REFERENCES

I have used the following references:

1 *PowerPath for Unix Concepts and Facilities Guide*, P/N 300-999-265, EMC Corporation

2 *PowerPath for Unix Installation Guide*, P/N 300-999-266, EMC Corporation

3 *PowerPath for Unix Administration Guide*, P/N 300-999-267, EMC Corporation

Alex Polyak
System Engineer
APS Israel

AIX news

Imperial Software has begun shipping X-Designer 7: Enterprise Edition, a new version of its GUI builder for Motif, Windows, and Java that runs under AIX.

Among the new features is integral support for Motif 2, the latest version of the standard GUI toolkit for Unix and Linux workstations, enabling designers of application interfaces for Unix to take full advantage of the features and power of Motif 2, and still be able to migrate these interfaces to Windows and Java for full cross-platform capability.

Also new is the ability to save GUI designs in XML, which makes the saved design available for processing by further tools, such as design documentation utilities, or for translation to other types of interface.

For further information contact:
Imperial Software Technology, Kings Court, 185 Kings Road, Reading, RG1 4EX, UK.
Tel: (0118) 958 7055.
URL: http://www.ist-inc.com.

* * *

Fujitsu Softek has announced an enterprise storage resource management tool, which centralizes and automates storage management in a multi-vendor environment from a single point.

The software allows companies to pool management of disparate data and storage hardware types. It lets them predict, supply, and manage an entire storage infrastructure, providing SRM, visualization, virtualization, replication, and migration tools.

It's out on AIX, OS/390 and z/OS, HP-UX, Solaris, and Windows NT/2000.

For further information contact:
Fujitsu Softek, 1250 East Arques Avenue, M/S 317 Sunnyvale, CA 94085 USA.
Tel: (877) 887 4562.
URL: http://www.softek.fujitsu.com.

* * *

IBM has announced enhancements to its WebSphere Application Server Enterprise Edition (AS EE) Version 4.1 with a host of enterprise services, TXSeries support for AIX, Solaris, and Windows NT/2000, MQSeries support, and a business process beans technology preview.

The enterprise services include business rule beans, message beans and JMS listener, internationalization, shared work areas, bidirectional CORBA connectivity (AIX, Solaris, and Windows NT/2000), a C++ CORBA software development kit, and an ActiveX to Enterprise Java Beans bridge for AIX, Solaris, and Windows NT/2000.

The new release runs on AIX, HP-UX, Solaris, Windows NT, and Linux (Red Hat and SuSE). It consists of the WebSphere Application Server Advanced Edition (AS AE), Enterprise Services, TXSeries, and MQSeries.

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

* * *