# 85

AIX update

*November 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 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.*

# Conditional tests

## THE TEST COMMAND

The **test** command is invaluable for testing conditions within flow control constructs. It can produce answers to questions such as, “does a given file exist, and is it readable?” and “is the length of this string non-zero?”. The general form of the **test** command is:

```
test expr
```

The expression *expr* is evaluated by **test**, and if it is true, **test** terminates with an exit status of **0**. If the expression is false, it returns a non-zero value. Also, **test** does not display anything on the screen and you must check its exit status to find out whether the expression is true or false.

Some of the more frequently used forms of the **test** command are shown below. The command is very particular about spaces, so be sure to use them only where they are shown.

## FILE AND OPTION TESTS

The syntax for *file* and set option tests are as follows.

- **test -a *file***

  Returns 0 if *file* exists.

- **test -b *file***

  Returns 0 if *file* is a block device file.

- **test -c *file***

  Returns 0 if *file* is a character device file.

- **test -f *file***

  Returns 0 if *file* is a regular file, not a directory.

- **test -d *file***

Returns 0 if *file* is a directory.

- **test -w *file***

  Returns 0 if *file* is writable by the user (use **-r** for readable, **-x** for executable).

- **test -s *file***

  Returns 0 if *file* exists and is not empty.

- **test -u *file***

  Returns 0 if *file* has the setuid bit set (use **-g** for setgid bit, and **-k** for sticky bit).

- **test -L *file***

  Returns 0 if *file* is a symbolic link.

- **test *file1* -nt *file2***

  Returns 0 if *file1* has a date of modification more recent than that of *file2*.

- **test *file1* -ot *file2***

  Returns 0 if *file1* has a date of modification less recent than that of *file2*.

- **test -o *option***

  Returns 0 if the specified *option* is on. The options available are those used with the **set -o** command, for example **set -o vi** to set the **vi** editor.

You can test only a single filename at a time so that *file* can be only a relative or absolute pathname and cannot contain metacharacters.

You can also use the **!** operator to test the reverse of a normal value. For example:

```
test ! -d file
```

will return a zero value if *file* exists, but is not a directory. There must be a space between the **!** and the following **-d**.

STRING COMPARISONS

String comparison tests are as follows.

- **test *string1* = *string2*** (or ***string* = *pattern***)

  Returns 0 if the strings *string1* and *string2* are identical (or *string* and *pattern* are identical).

- **test *string1* != *string2*** (or ***string* != *pattern***)

  Returns **0** if the strings *string1* and *string2* are not identical (or *string* and *pattern* are not identical).

- **test *string1* < *string2***

  Returns 0 if *string1* is lexically less than *string2*.

- **test *string1* > *string2***

  Returns 0 if *string1* is lexically greater than *string2*.

- **test -z *string***

  Returns **0** if the length of *string* is zero, ie the *string* is set to null.

- **test -n *string***

  Returns 0 if the length of *string* is non-zero.

You should be aware that, when you are testing the values of variables and strings, if any has a null value, is set to ‘ ’, or contains a metacharacter, then you will get an error message from **test** if you do not enclose the variable name in double quotes.

You can try this with the following simple example:

```
$ unset a
$ test -z $a
/bin/ksh: test: 0403-004 Specify a parameter with this command.
$ print $?
1

$ test -z "$a"
$ print $?
0
```

Although we have indicated that you can use **test** to check whether strings come before or after each other in the dictionary order (the >

and < operators above), in reality you will not be able to do this using the standard **test** format since the shell will interpret the '>' and '<' as redirection symbols and so give you an error message. This type of comparison can be done only by using the brackets format to perform the test (see below).

For string comparisons, you can only use the negation operator, **!**, with the **=**, **-n**, and **-z** operators. There should be no spaces between the **!** and **=**, but there must be spaces between the **!** and the **-z** or **-n**.

## INTEGER COMPARISONS

The **test** command allows for the integer comparison of expressions using the following syntax:

- **test *expr1* -eq *expr2***

  Returns 0 if *expr1* is equal to *expr2*.

- **test *expr1* -ne *expr2***

  Returns 0 if *expr1* is not equal to *expr2*.

- **test *expr1* -gt *expr2***

  Returns 0 if *expr1* is greater than *expr2*.

- **test *expr1* -lt *expr2***

  Returns 0 if *expr1* is less than *expr2*.

- **test *expr1* -ge *expr2***

  Returns 0 if *expr1* is greater than or equal to *expr2*.

- **test *expr1* -le *expr2***

  Returns 0 if *expr1* is less than or equal to *expr2*.

Both *expr1* and *expr2* can be variable names, or they can be expressions which generate integer values, such as **4*3** to give the value **12**, or they can be mixtures of variables and arithmetic operators and integers, such as **$var*3** as a simple example; the syntax of these expressions will be discussed in detail in a future article.

In the same way that we get error messages during string comparisons

when a variable has a null value, integer comparisons also produce the same error messages. If you are performing integer comparisons and the variable contains non-numeric characters, then you will get a different error message. For example:

```
$ a=aa
$ test $a -eq 1
/bin/ksh: aa: 0403-009 The specified number is not valid for this
command.
```

Whether you should use integer or string comparisons in determining whether two variables are 'equal' is really dependent on the type of comparison you would like to make.

For example, consider two variables **a** and **b**, where the value of **a** is **0123**, and the value of **b** is **123**. If these two variables are required to be considered as text strings, then you would want to use a comparison which would indicate that they were different. If you used a test such as:

```
test "$a" = "$b" (or test $a = $b)
```

then this would perform a string comparison and return an exit status of 1, showing that they were indeed different strings.

If the two variables are required to be considered as numbers, then you would want to perform an integer comparison since they are essentially the same number if you strip off leading zeros. The test to be performed in this case to provide the expected result would be:

```
test "$a" -eq "$b" (or test $a -eq $b)
```

which would now return an exit status of 0, indicating that they were identical. An alternative would have been to initially define the variables using **typeset -LZ**, which would automatically strip off leading zeros and so it would not then matter whether you performed an integer or string comparison.

A similar problem will occur when you want to test, for example, the relationship between '2' and '12'. Lexically, '12' comes before '2', but numerically the opposite is true.

## TESTS USING [ ] AND [[ ]]

There are two other constructions that are more commonly used than

**test** and are equally suitable for file, string, and integer comparisons. Each uses the same syntax as **test**, but surrounds its comparison arguments with either **[ ]** or **[[ ]]**; each bracket, or pair of brackets, must be surrounded by blank spaces. For example:

```
test -z $1
```

can be replaced by:

```
[ -z $1 ] or [[ -z $1 ]]
```

This construction is most commonly used as a test for the **if** command, and various looping constructs, which will be covered in future articles.

### Differences between [ ] and [[ ]]

Although the **[ ]** and **[[ ]]** constructions can both be used to perform tests, they cannot be used interchangeably since there are a number of subtle differences between the two:

- Variables containing metacharacters, null, or empty – when using **[ ]** to compare or determine the values of variables and strings, you should always put the variable names in quotes if there is the remotest possibility that a variable can take a null value or an empty value, or could contain metacharacters. A test will display an error message if it detects a syntax error, which you would get if you used, for example, **[ test -z $var ]** and **var** had a null value.

  By comparison, **[[ ]]** will not complain if any variable or string takes on a null value and the variable or string has not been quoted. It will not complain of a syntax error and will continue with the test as if quotes had been used.

- Comparisons using redirection symbols – as we mentioned earlier, the standard form of **test** does not allow the use of > and < to compare the values of strings since this will produce error messages. The only way this can be achieved is by testing within **[[ ]]**. If you try the test using single brackets you will get an error message.

- Combining test expressions – as I'll explain in my next article, the operators used to combine test expressions **-a** and **-o** can be used within single brackets, but not within double brackets.

Generally speaking, the **[[ ]]** construction is preferred since it has a greater flexibility and ignores wildcard expansion and other metacharacters. With certain exceptions, and you will see later when we discuss the ways in which tests can be combined that these are purely a matter of personal preference, you should always use **[[ ]]** rather than **[ ]**.

## MODIFYING THE SEARCH SCRIPT

Now that you are familiar with the **if** command, let us modify **search** so that the output of the **find** command does not scroll off the screen.

```
#!/bin/ksh
# Script name: search
# Usage: search
#############################################################
# Version History
# Version      Date      Remarks
# 1.Ø                  Original Version
# 1.1                  Catch-all case pattern added
# 2.Ø                  Modified to prevent scrolling off the screen
#############################################################
TMPFILE=/var/tmp/search.$$
#---------------------------------------------
# Function f_dsp_tmpfile
# Arguments: $1 - file or string
# Displays contents of temporary search file
#---------------------------------------------
f_dsp_tmpfile()
{
    if [[ ! -s $TMPFILE ]]
    then
      print "$1" not found
    else
      more $TMPFILE
    fi
    rm $TMPFILE
}
# determine type of search the user wants
print 'Are you searching'
print 'a: For a file/directory; or'
print 'b: For a string within a file'
print 'q: Quit'
print 'Enter your choice: \c'
read answer
case $answer in
a|A) # search for a file
    print '\nWhich file are you looking for?: \c'
```

```
    read file
    find / -name "$file" -ls > $TMPFILE 2>/dev/null
    # test if file has been found
    f_dsp_tmpfile $file
    ;;
 b|B) # search for a string within a file
    print '\nWhich file is it in?: \c'
    read file
    print 'What is the string?: \c'
    read string
    grep "$string" "$file" > $TMPFILE

    # test if string has been found
    f_dsp_tmpfile "$string"
    ;;
 q|Q)
    exit 0
    ;;
*)
    print "search: '$answer' is not a valid choice"
    exit 1
    ;;
esac
print Good- bye
```

The additions to **search** are the redirection commands to send the output of the **find** and **grep** commands to a temporary file, which we have defined by the variable **TMPFILE**, and the function **f_dsp_tmpfile**, which checks that we have some output to display; after the output has been viewed we then remove the temporary file.

We have also added a final stanza to the **case** command, which allows us to enter **q** to quit from the script without making any other choices. Previously we could not do this.

## EXAMPLE TO CHECK FILESYSTEM USAGE

The following example makes use of multiple **if** and **elif** statements and is used to check the current state of filesystem usage. It uses a text source file that contains stanzas for each of the filesystems you wish to check. The source file is called **fs_limits**, sample entries of which are shown below.

```
# Filename: fs_limits
#
# Source file for the chkfs command
```

```
############################################################
# There must be only one stanza for each filesystem and
# stanzas must be separated by blank lines
# Each stanza must contain a chk_type attribute, and either a
# max_used or min_free attribute
#
# chk_type can be either:
#       percent - checks percentage utilised (uses max_used)
#       bytes   - checks KB of free space (uses min_free)
#
# max_used:  maximum percentage usage allowed
#      - value between 1 and 99
# min_free:  minimum free space required
#      - value in bytes * 1000
############################################################

/:
      chk_type = percent
      max_used = 90

/tmp:
      chk_type = bytes
      min_free = 8000
```

The stanza for each filesystem consists of three lines; the first is the name of the filesystem followed by a colon. The second line contains the attribute **chk_type**, which can take two possible values:

- *percent* – checks the percentage of the filesystem that has been used.
- *bytes* – checks the available free space in KB.

The third line indicates the permitted threshold of the filesystem. It can take two possible values:

- *max_used* – maximum filesystem percentage usage allowed. If current usage is greater than this value then the threshold has been exceeded. Permissible values are from 1 to 99 inclusive. This attribute is used only by the **percent** *chk_type* attribute.
- *min_free* – minimum amount of free space in KB that must be available. If current usage shows less than this amount free then the threshold has been exceeded. This attribute is only used by the **bytes** *chk_type* attribute.

Our sample file contains only two filesystems, but you can enter any number of stanzas in the file with a limitation of one per filesystem.

Scripts which extract entries from text source files can become very complex indeed when extensive checking is performed to ensure that the source file is in the correct format. To reduce the checking to be done on our own source file to an acceptable level, we require that stanzas be separated by blank lines. If this is not the case then error messages will be generated, and our current knowledge of shell programming would require complex and cumbersome coding to extract the correct attribute for each filesystem when you bear in mind the number of possible error conditions that could exist when two or more stanzas are joined together.

The script performing the check is called **chkfs**, and its usage is:

```
chkfs filesystem_name
```

The script is shown below:

```
#!/bin/ksh
# Filename: chkfs
# Usage: chkfs filesystem_name
# Source file: fs_limits
############################################################
# Version History
# Version      Date    Remarks
# 1.Ø                        Original Version
############################################################
SOURCE=/usr/local/lib/fs_limits
#-----------------------------------------------
# Function: f_chk_args
# Arguments: $1 - filesystem name
# Checks usage and displays error messages
#-----------------------------------------------
f_chk_args()
{
      if [[ $# -ne 1 ]]
      then
            print "Usage: $(basename $Ø) filesystem_name"
            exit 1
      fi

      FS=$1
}
#-----------------------------------------------
# Function: f_chk_valid_fs
```

```
# Checks that the filesystem name is valid and mounted
#-----------------------------------------------
f_chk_valid_fs()
{
      # Is the filesystem name valid?
      lsfs $FS >/dev/null 2>&1
      if [[ $? -ne 0 ]]
      then
            print $FS is not a valid filesystem name
            exit 1
      fi
      # Is the filesystem mounted?
      df | tr -s ' ' | cut -d ' ' -f 7, | grep $FS >/dev/null 2>&1
      if [[ $? -ne 0 ]]
      then
            print $FS is not mounted
            exit 1
      fi
}
#-----------------------------------------------
# Function: f_chk_source
# Checks for existence of source file and filesystem entry
#-----------------------------------------------

f_chk_source()
{
      # Does the source file exist?
      if [[ ! -f $SOURCE ]]
      then
            print $SOURCE source file does not exist
            exit 1
      fi

      # Is there a filesystem entry in the source file?
      NUM=$(grep -c ^${FS}: $SOURCE)
      if [[ $NUM -eq 0 ]]
      then
            print No entry for $FS in $SOURCE
            exit 1
      elif [[ $NUM -gt 1 ]]
      then
            print Multiple entries for $FS in $SOURCE
            exit 1
      fi
}

#-----------------------------------------------
# Function: f_get_limits
# Get the limits for the filesystem from the source file
#-----------------------------------------------
```

```
f_get_limits()
{
      CHK_TYPE=$(grep -p ^$FS: $SOURCE | grep chk_type |
          tr -d ' ' | cut -d "=" -f 2,)
      if [[ -z $CHK_TYPE ]]
      then
          print There is no chk_type value for $FS
          print "\tin $SOURCE"
          exit 1
      fi

      if [[ $CHK_TYPE = percent ]]
      then
          MAXUSED=$(grep -p ^$FS: $SOURCE | grep max_used |
            tr -d ' ' | cut -d "=" -f 2,)
          if [[ -z $MAXUSED ]]
          then
            print There is no max_used value for $FS
            print "\tin $SOURCE"
            exit 1
          fi

          if [[ $MAXUSED -lt 1 ]]
          then
            print max_used for $FS in $SOURCE is $MAXUSED
            print "\tValue must be from 1 to 99"
            exit 1
          fi

          if [[ $MAXUSED -gt 99 ]]
          then
            print max_used for $FS in $SOURCE is $MAXUSED
            print "\tValue must be from 1 to 99"
            exit 1
          fi

      elif [[ $CHK_TYPE = bytes ]]
      then
          MINFREE=$(grep -p ^$FS: $SOURCE | grep min_free |
            tr -d ' ' | cut -d "=" -f 2,)
          if [[ -z $MINFREE ]]
          then
            print There is no min_free value for $FS
            print "\tin $SOURCE"
            exit 1
          fi
          if [[ $MINFREE -lt 1 ]]
          then
            print min_free for $FS in $SOURCE is $MINFREE KB
```

```
            print "\tValue must be greater than or equal to 1 KB"
            exit 1
          fi
      else
          print chk_type for $FS in $SOURCE is $CHK_TYPE
          print "\tIt should be percent or bytes"
          exit 1
      fi
}
#---------------------------------------------
# Function: f_get_fs_usage
# Get the current usage
#---------------------------------------------
f_get_fs_usage()
{
      if [[ $CHK_TYPE = percent ]]
      then
          USED=$(df -k $FS | tail +2 | tr -s ' ' |
            cut -d ' ' -f 4, | tr -d "%")
          if [[ $USED -gt $MAXUSED ]]
          then
            print $FS has exceeded its threshold
            print "\t${USED}% used - threshold ${MAXUSED}%"
          fi
      else
          FREE=$(df -k $FS | tail +2 | tr -s ' ' |
            cut -d ' ' -f 3,)
          if [[ $FREE -lt $MINFREE ]]
          then
            print $FS has exceeded its threshold
            print "\t${FREE} KB free - threshold \
${MINFREE} KB"
          fi
      fi
}
################################################
# Main section
################################################
f_chk_args $*            # Check arguments
f_chk_valid_fs           # Valid filesystem?
f_chk_source             # Check the source file
f_get_limits             # Get limits from source
f_get_fs_usage           # Get current usage
```

As with many scripts, the majority of code tests for error conditions, and so the first function (**f_chk_args**) checks that there is a single command line argument and prints an error message if this is not the case. If the filesystem is valid then **$1** is set to the **FS** variable, which is then used throughout the script.

We then check that the command line argument is a valid filesystem name, and that the filesystem is mounted (**f_chk_valid_fs**). The **cut** command uses spaces for field separators, but, since the **df** command output contains multiple spaces, we have piped this output into **tr -s** to *squeeze out* multiple occurrences of the space. Only then can we be certain of extracting the filesystem name from field number seven.

Next we check that there is a source file and that there is just a single entry for the filesystem. You will see that the **grep** command counts (**-c** option) the number of times that the filesystem name occurs in the file. We use the pattern **^${FS}:** to search from the start of a line up to the first colon so that we can exclude characters matching our filesystem name, which may also be embedded within another filesystem. For example, '**^/usr:**' will not match '**^/usr/local:**', whereas '**^/usr**' would.

Now that we are certain the filesystem exists, and is mounted, we extract the threshold limits from the source file (**f_get_limits**). The function determines the **chk_type** value, the associated threshold (free space or percent used), and the validity of the threshold. We have used **grep -p** to extract the whole filesystem stanza, and so stanzas in the source file must be separated by blank lines.

Finally, after all the error checking, we determine the current filesystem usage (**f_get_fs_usage**) and compare this with the threshold to see whether it has been exceeded.

You will note that we have used **exit 1** when the script cannot complete because of some error condition. We could have used a different exit status for each error, but since the script produces a message each time it exits, and the script is unlikely to be called by any other script, we do not need to do this.

The script as it currently stands takes only a single filesystem name as an argument. This is not particularly useful since you would normally want to check the usage of all filesystems, but using this script you would have to execute it multiple times, each time with a different filesystem name. You will see in a future article how we can check filesystem usage for all entries in our source file using a single command.

A script like this, and we are talking about the version which would

 

check multiple filesystems, would probably be run as a **cron** job, say every 15 minutes. This on its own is not a great deal of use since you would have to check **root**'s mail on a regular basis since this is where all output from the script would then go when run as a **cron** job; you may as well run the script from the command line in this case. For the script to really prove useful, you would need additional functions to, say, send a Lotus Notes message to selected users when a threshold had been breached.

We could, of course, just use the **df** command to check the usage statistics for all our filesystems, so why would we need the script? Filesystems can often fill up without warning, particularly when error conditions write continuous entries to logs. Some filesystems fill up on a gradual basis, and then either the system crashes or an application refuses to respond before this is spotted. To ensure that your system keeps running, you would probably have to run **df** very frequently, which may not be administratively possible or desirable. By automating the running of the script, and assuming you have some notification method for reporting threshold breaches, then you could remove this onerous task.

*Tonto Kowalski*
*Guru (UAE)*


## AIX security review

In Issue 81 of *AIX Update* (July 2002) we reviewed some of the security exposures that affected AIX between February 2001 and March 2002 (*A review of recent AIX security exposures*). As a supplement to this article, we will review some of the vulnerabilities that have been exposed between June and July 2002.

Unlike the previous review there have been no exposures recorded that are a direct result of bugs in the core AIX operating system code. The vulnerabilities are general Unix security issues (such as DNS, OpenSSH, and Linux) that can have an impact on AIX in some circumstances. Because these are sometimes tangential to AIX, their security is often overlooked, but they can provide a back door into a system. We

consistently find that this is an area of weakness that many organizations overlook. As before, the exploits and vulnerabilities are reviewed in chronological order of discovery, beginning with the most recent.

## UNIX SECURITY EXPOSURES

The following vulnerabilities have been discovered and recorded between June and July 2002.

### Vulnerabilities in CDE ToolTalk

In early July 2002, two vulnerabilities were reported by Core Security Technologies in systems running Common Desktop Environment (CDE) ToolTalk RPC database server. The first vulnerability could allow a remote attacker to delete files, cause a denial of service, or execute arbitrary code. The second vulnerability could allow a local attacker to overwrite files with alternative content.

The Common Desktop Environment (CDE) is an integrated GUI that runs on Unix operating systems including AIX. It is a message brokering system that allows applications to communicate with each other across hosts and platforms. The ToolTalk RPC database server, rpc.ttdbserverd, manages communication between ToolTalk applications.

The vulnerabilities discovered are:

- The CDE ToolTalk RPC database server (rpc.ttdbserverd) does not adequately validate file descriptor arguments to _TT_ISCLOSE(). Because of this, a remote attacker could overwrite certain locations in memory with zeros. The attacker could then use this exposure to delete any file that is accessible from the ToolTalk RPC database server. Because the server typically runs with root privileges, any file on a vulnerable system could be deleted. Overwriting memory or deleting files could result in a denial-of-service. It may also be possible to execute arbitrary code and commands.
- The CDE ToolTalk RPC database server (rpc.ttdbserverd) does not validate file operations properly. Therefore, it is unable to ensure that the target of a file write operation is a valid file and not a symbolic link. Because of this vulnerability, it is possible for a local attacker to overwrite any file that is accessible by the ToolTalk RPC

database server. Typically, the server runs with root privileges, so any file on a vulnerable system could be overwritten. Obviously, overwriting root-owned files could lead to privilege escalation or a denial-of-service attack.

The CDE desktop product shipped with AIX is vulnerable to both these issues. This affects AIX Releases 4.3.3 and 5.1. There are APARs available for AIX 4.3.3 (IY32368) and AIX 5.1 (IY32370).

If there is a delay in deploying the APARs, it is advisable to disable the ToolTalk RPC database service. The program number for the ToolTalk RPC database server is 100083. Alternatively, you could use a firewall or other packet-filtering technology to block the appropriate network ports. Remember that blocking ports at a network perimeter does not protect from attacks that originate from within the internal network.

**OpenSSH exposures in challenge response handling**

In late June 2002, it became clear that there were at least two vulnerabilities in OpenSSH challenge response handling. The exposures affect OpenSSH Versions 2.3.1p1 to 3.3. They may allow a remote intruder to execute arbitrary code as the user running sshd (often root). The first vulnerability affects OpenSSH Versions 2.9.9 through 3.3 that have the challenge response option enabled and that use SKEY or BSD_AUTH authentication. The second vulnerability affects PAM modules using interactive keyboard authentication. This applies to OpenSSH Versions 2.3.1p1 to 3.3, regardless of the challenge response option setting. Additionally, a number of other possible security problems have been corrected in OpenSSH Version 3.4.

- The first exposure is an integer overflow in the handling of the number of responses received during challenge response authentication. If the challenge response configuration option is set to yes and the system is using SKEY or BSD_AUTH authentication, then a remote intruder may be able to exploit the vulnerability to execute arbitrary code. This vulnerability is present in versions of OpenSSH 2.9.9 to 3.3. A hack exploiting this vulnerability is apparently available.
- The second vulnerability is a buffer overflow involving the number of responses received during challenge response authentication.

Regardless of the setting of the challenge response configuration option, systems using PAM modules that use interactive keyboard authentication (PAMAuthenticationViaKbdInt) may be vulnerable to the remote execution of code. A remote attacker can execute code with the privileges of the user running the sshd (which is often root). These vulnerabilities may also be used to cause a denial of service.

Both of these vulnerabilities exploit features present only in Version 2 of the SSH protocol. AIX does not ship with OpenSSH; however, OpenSSH is available for installation on AIX using the Linux Affinity Toolkit. The versions included on the CD and Web site containing the Toolkit are vulnerable to the exposures. Version 3.4 can be downloaded from http://www6.software.ibm.com/dl/aixtbx/aixtbx-p

Some alternative workarounds have been proposed:

- *Disable SSH protocol Version 2*. Because both exposures are present in protocol Version 2 only, disabling Version 2 of the protocol will prevent both vulnerabilities from being exploited. This can be accomplished by adding the following line to */etc/ssh/sshd_config*:

  ```
  Protocol 1
  ```

  Remember that disabling protocol Version 2 could prevent the sshd daemon from accepting connections in some situations. It may be advisable to disable the challenge response authentication and/or use the interactive keyboard to disable PAM authentication.

- *Disable challenge response authentication*. With OpenSSH Versions 2.9 and above, it is possible to disable the vulnerable portion of the code by setting the 'ChallengeResponseAuthentication' configuration option to 'no' in their sshd configuration file. Typically, this can be done by including the following line in */etc/ssh/sshd_config*:

  ```
  ChallengeResponseAuthentication no
  ```

  This should prevent the first vulnerability from being exploited if SKEY or BSD_AUTH authentication is used. However, it will not prevent the possible exploitation of the vulnerability using the PAM interactive keyboard authentication.

- *Use the interactive keyboard to disable PAM authentication*. For OpenSSH Version 2.9 and above, it is possible to disable the vulnerable portion of the code affecting the PAM authentication issue by setting the 'PAMAuthenticationViaKbdInt' configuration option to 'no' in the sshd configuration file. Adding the following line to */etc/ssh/sshd_config* should achieve this goal:

  ```
  PAMAuthenticationViaKbdInt no
  ```

  This workaround should prevent the second vulnerability from being exploited if PAM interactive keyboard authentication is used. It will not prevent the possible exploitation of the vulnerability using SKEY or BSD_AUTH authentication.

- *Disable both options in older versions of OpenSSH*. In the older versions of OpenSSH between 2.3.1p1 and 2.9, system administrators will need to disable the following options in the ssh configuration file:

  ```
  KbdInteractiveAuthentication no
  ChallengeResponseAuthentication no
  ```

  Disabling both options should prevent the exploitation of the vulnerabilities.

**Apache Web Server chunk handling vulnerability**

The Apache Web server includes support for chunk-encoded data conforming to the HTTP 1.1 standard (see RFC2616). In July 2002 it became apparent that the Apache Web server contains a vulnerability in its handling of certain chunk-encoded HTTP requests. This could allow remote attackers to execute arbitrary code.

This vulnerability is present by default in configurations of Apache Web server Versions 1.2.2 and above, 1.3 to 1.3.24, and Versions 2.0 to 2.0.36. The impact of this vulnerability depends on the software version.

For example, with Apache Versions 1.2.2 to 1.3.24 inclusive, this vulnerability could allow a remote attacker to execute arbitrary code. Hacks are currently available on the Web that allow the execution of arbitrary code.

For Apache Versions 2.0 to 2.0.36 inclusive, the condition causing the

vulnerability is correctly detected and causes the child process to exit. Depending on a variety of factors, including the threading model supported by the vulnerable system, this may lead to a denial-of-service attack against the Apache Web server.

The principal means of combating this vulnerability is to upgrade to the latest version of Apache. The Apache Software Foundation has released two new versions of Apache that correct this vulnerability. It is possible to prevent the vulnerability from being exploited by upgrading to Apache httpd Version 1.3.26 or 2.0.39. These can be found at the Apache Web site at the following URL, http://www.apache.org/dist/httpd/.

IBM makes the Apache Server available for AIX as a software package in the Linux Affinity Toolkit. This package is included on the AIX Toolbox for Linux Applications CD, and can be downloaded via the IBM Linux Affinity Web site. The currently available version of Apache Server is susceptible to the vulnerability described here. An updated and patched version of Apache Server (Version 1.3.23) is available from the following URL, http://www-1.ibm.com/servers/aix/products/aixos/linux/download.html.

Note: remember to check whether you are using Apache in any other locations in your shop. Linux products are becoming surprisingly insidious in enterprises because of their inherent reliability. Furthermore, Apache can also be found integrated with other software. For example, IBM's HTTP Server, which is bundled with WebSphere, is also based on the Apache server. Therefore, it is also vulnerable to the current chunk handling exposure. This should be patched soon. IBM's Lotus software division does not use Apache as a code base.

**Buffer overflow in DNS resolver libraries**

In June 2002, it became clear that DNS resolver libraries were vulnerable to buffer overflow in certain circumstances. Buffer overflows have been a favourite means of disruption for malicious hackers.

The DNS protocol provides name, address, and other information about Internet Protocol (IP) networks and devices. In order to access DNS information, a network application needs to use a resolver. This performs DNS queries on its behalf. The Resolver functionality is

 

usually implemented in libraries that are included with operating systems, such as AIX.

All applications that use the vulnerable implementations of the Domain Name System (DNS) resolver libraries are affected. These include, but are not limited to: the Internet Software Consortium (ISC) Berkeley Internet Name Domain (BIND) DNS resolver library (libbind), the Berkeley Software Distribution (BSD) DNS resolver library (libc), and the GNU DNS resolver library (glibc).

Both BSD (libc) and ISC (libbind) resolver libraries share a common code base and are exposed to this problem; any DNS resolver implementation that derives code from either of these libraries could also be vulnerable. Network applications that make use of vulnerable resolver libraries are probably affected, therefore this problem is not limited to DNS or BIND servers, but is quite a widespread issue. The DNS resolver code supplied in AIX 4.3 and 5.1 is vulnerable.

Using this vulnerability, a malicious user could exploit operating systems and applications that use vulnerable DNS resolver libraries to execute arbitrary code or cause a denial of service. Any code executed by the attacker would run with the privileges of the process that calls the vulnerable resolver function.

It is possible for a malicious hacker to use the target organization's network services to make a DNS request to a DNS server under the control of the attacker. This would allow the attacker to remotely exploit this vulnerability.

One workaround that has been suggested involves the use of a local cacheing DNS server. If you have BIND 9 it may be possible to use a local cacheing DNS server that reconstructs DNS responses to prevent malicious responses from reaching systems using vulnerable DNS resolver libraries. BIND 9 reconstructs responses in this way, with the exception of forwarded dynamic DNS update messages. However BIND 8 does not reconstruct all responses; therefore this workaround may not be effective when using BIND 8 as a cacheing DNS server.

IBM has provided the following APARs for AIX 4.3 (IY32719) and AIX 5.1 (IY32746). Remember that the APARs and other fixes for the vulnerabilities mentioned in this article can be obtained from the following URL, http://techsupport.services.ibm.com/rs6k/fixdb.html.

CONCLUSIONS

All of the vulnerabilities recorded in this period by CERT, IBM's ERS, Core Security Technologies, and other security organizations originate outside of the AIX code base. This is good news for the majority of users. However, the exposures still mean that systems administrators need to check whether they have the vulnerable Unix applications running in their shops.

Many of the more recent vulnerabilities that have been shown to affect AIX originate in the Linux Affinity Toolkit. In March 2002, we saw the bug in the zlib Compression Library. In June, vulnerabilities were exposed in the OpenSSH challenge response handling, and in July the Apache Web Server chunk handling vulnerability was revealed. There are two principal reasons why so many bugs are coming out of the Linux Affinity Toolkit. First, there are quite a number of packages included in the Toolkit, but, most importantly, the Linux community is extremely efficient at locating and patching software vulnerabilities. This is good news for end users in the long run.

However, what this means for end users is, if you have the Linux Affinity Toolkit in your shop, make sure that it is patched to the appropriate level – and keep the patches up to date.

*Systems Programmer (UK)* 

## Understanding the at command

AT COMMAND BASICS

Have you ever wanted to have a simple command that would remind you of important events such as meetings, anniversaries, luncheon dates, etc?

What about having a command that can schedule the execution of one of your shell scripts during non-peak system usage times?

The **at** command can help you with these tasks. This command takes as input one or more strings of commands, or a pointer to a script or

 

list of commands, and executes the commands at a time specified by the user.

For example, suppose you wanted to run the calendar command at 10:00pm and mail the results to a particular user. To do that, you would enter the following commands:

```
at 22:00
calendar | mail psmith
CNTL+D
```

The first line is the **at** command and the time parameter, followed by the desired command string to be run, and finally CNTL+D to end input into the **at** command.

Now suppose you wanted to send a reminder to yourself about an anniversary in five weeks.

```
at 3 PM next month
echo Anniversary next week! | mail rjones
CNTL+D
```

The results would be as follows. At 3:00pm on the current date (or the date after, if issued past 3pm on the current date) of the month following the current month, mail would be sent to user rjones with the string "Anniversary next week!" contained within.

The basic syntax of the **at** command is shown below:

- Scheduling a job:

  ```
  at SHELL QUEUE flags TIME
  ```

- Listing jobs:

  ```
  at flags JOBS QUEUE
  at flags USER
  ```

- Removing jobs:

  ```
  at flags JOBS
  at flags USER
  ```

where:

- *SHELL* – optional flag for the shell in which the command is to run, for example the C, Korn, or Bourne shells.
- *QUEUE* – optional flag for the queue to which the command is to

belong, for example, the **at**, batch, KSH, or CSH queues.

- *TIME* – the time and/or date specification the job is to run.
- *JOBS* – the alphanumeric identifier for the job or jobs.
- *USER* – the user for which the jobs are scheduled.
- *flags* – other flags meaningful to the operation.

## RUNNING THE AT COMMAND

There are two ways you can run the **at** command – the stanza method and the command line method.

### Stanza method

The stanza method allows you to give the **at** command a list of commands it is to execute when the scheduled job is set to run.

To use this method, specify the time parameter after the **at** command and press *Enter*. This will give you a blank line with your cursor. You are now in data entry mode for the **at** command. Enter one or more commands you would like the **at** command to schedule for this job. After entering the last command, press CNTL+D to end data entry and to return to the command prompt. You have now scheduled an **at** job. The names of the commands you have specified must have the executable flag set in the permissions, but need not be in the current directory:

```
at TIME
command 1
command 2
command 3
CNTL+D
```

For example, let's say you have a file called FormatDoc that supplies parameters in order to perform formatting of a document. Since each formatting process may take over an hour, you want to schedule the activity during non-peak system usage:

```
at 3:ØØ AM
FormatDoc /home/psmith/docs/book1.source
FormatDoc /home/psmith/docs/book2.source
echo book1 and book2 formatted | mail psmith
```

```
echo book1 and book2 formatted | mail teamlead
CNTL+D
```

This sample would perform the formatting operation on the two book source files, then send the string 'book1 and book2 formatted' via AIX mail to the two users psmith and teamlead beginning at 3:00 the following morning.

**Command line method**

The other method for scheduling **at** jobs is to use the command line method. In this case, the entire job is scheduled by typing one line from a command prompt. There is no need for the CNTL+D key sequence.

This method is useful if you do not need to execute more than one command or if the list of commands can be executed from a single file. Unlike the stanza method, this method does not require the name of the file to be executable. The name would be considered a list of commands for the **at** job, even if the file contains only one command. If the command specified using the command line method is not in the current directory, you must specify a path. Also, there can be no arguments entered for the command using this method:

```
at TIME command
```

Each of the following three examples will run the list of commands in the file called 'compile' at 1:59am:

1 Type the name of the command list after the TIME parameter:

```
at 1:59 am compile
```

2 Direct the name of the command list into the **at** command:

```
at 1:59 am <compile
```

3 Use the **-f** *file* parameter of the **at** command to specify the command list:

```
at -f compile 1:59 am
```

FLAGS FOR THE AT COMMAND

The following flags extend the usefulness of the **at** command:

- -c – tells the **at** command that the scheduled job is to be run in the

CSH (C shell) environment.

- -f *file* – file to be used as input.
- -F – does not display warning message prior to the deletion of a job when used with the -r flag. Useful for deleting scheduled jobs in automated script files.
- -i – displays a prompt before deleting a job when used with the -r flag. Useful for selectively deleting jobs from a queue.
- -k – tells the **at** command that the scheduled job is to be run in the KSH (Korn shell) environment.
- -l – displays a list of scheduled jobs. A root user would see all jobs. In that case, you can pipe the output into the **grep** filter to display only the jobs for a particular user.
- -m – mails successful execution message to user.
- -n *user* – displays the number of jobs in queue for the specified user.
- -o – displays scheduled jobs in chronological order when used with the -l flag. The default is to display in the order in which the schedules entered the system.
- -q a – tells the **at** command to schedule the job to the **at** queue, which is the default queue. Displays jobs queued to the **at** queue when used with the -l flag.
- -q b – tells the **at** command to schedule the job to the batch queue. Displays jobs queued to the batch queue when used with the -l flag.
- -q e – tells the **at** command to schedule the job to the KSH queue. Displays jobs queued to the KSH queue when used with the -l flag.
- -q f – tells the **at** command to schedule the job to the CSH queue. Displays jobs queued to the CSH queue when used with the -l flag.
- -r *job* – removes the specified scheduled job. The *job* parameter is not needed when used with the -u flag.
- -s – tells the **at** command that the scheduled job is to be run in the BSH (Bourne shell) environment.
- -t DATE – schedules the job to run at a particular time/date.

- -u USER – tells the **at** command the user for whom jobs are to be deleted when used with the -r flag.

TELLING THE AT COMMAND WHEN TO SCHEDULE A JOB

The time is specified using either the -t *date* flag or by Time/Day/Increment values. If you try to schedule two jobs to run at the exact same time, the **at** command will increment each duplicated scheduled time to be one second later than the previously scheduled job. In other words, if you scheduled three jobs to begin at 3:00pm on the same date, **at** would schedule them as follows:

```
user.1025550000.a      Mon Jul  1 15:00:00 EDT 2002
user.1025550001.a      Mon Jul  1 15:00:01 EDT 2002
user.1025550002.a      Mon Jul  1 15:00:02 EDT 2002
```

SPECIFYING THE TIME USING THE -t DATE FLAG

The format of the date parameter of the -t flag is:

YYYYMMDDHHMM.SS

where:

- YYYY – optional year, for example, 2003. You can leave off the first two digits. In that case, 03 would be assumed to be 2003. The default is the current year.
- MMDD – month and day. For example 0531 would be 31 May.
- HHMM.SS – hours, minutes, and, optionally, seconds. This specification is based on a 24-hour clock. For example 1730 would be 5:30pm. The default value for seconds is 00.

  For example:

  ```
  at -t 200305311730.59
  ```

  would execute 59 seconds after 5:30pm on 31 May 2003.

SPECIFYING THE TIME USING THE TIME/DAY/INCREMENT VALUES

You can also tell the **at** command when to schedule a job using time, day, and increment parameters.

For example:

```
at 5:ØØ pm May 9, 2ØØ3 +2 hours
```

would execute at 7:00pm on 9 May 2003. In this case, 5:00pm is the *time*, 9 May 2003 is the *day*, and +2 hours is the *increment* added to the *time* value. Let's look at the individual parameters.

**The time parameter**

The time parameter can be specified as follows:

- H – hour. 3 would be 3:00am, 9pm would be 9:00pm.
- HH – hour. 11 would be 11:00am, 16 would be 4:00pm.
- HHMM – hour and minutes. 0945 would be 9:45am, 1015pm would be 10:15pm.

A colon (:) can also be used between the hours and minutes as follows:

- H:MM – hour (between 0 and 9) and minutes. 1:30 would be 1:30am, 9:20pm would be 9:20pm.
- HH:MM – hour (between 00 and 12 with suffix or between 00 and 23) and minutes. 04:00 would be 4:00am; 03:10pm or 15:10 would be 3:10pm.
- noon – 12:00pm. *at noon fri* (*at N friday*) would cause the command to run at 12:00 noon on Friday.
- midnight – 12:00am. *at midnight December 25* (*at M Dec 25*) would cause the command to run at 12:00 midnight on 25 December.
- now – immediately, or at this time on another date. *at now* would run the command immediately following the depression of the *Enter* key. *at now tomorrow* would run the command in 24 hours' time.
- A – am. 5 a or 5 A would be 5:00am.
- P – pm. 2 p or 2 P would be 2:00pm.

Note: for the hour specifications below, a 24-hour clock is assumed

(HH in the range of 13 - 23, unless a specific suffix is added). Suffixes can be expressed in either upper or lower case.

### The day parameter

The day parameter is optional and can be expressed (in mixed case) as a month and day; a month, day, and year; just the day of the week; or even the words *today* or *tomorrow*, as follows:

```
June 4          June 4
jun 4

jun 4, 2003     June 4, 2003
June 4, 03

We              Wednesday
wed
Wed
Wednesday

today           Today's date

tomorrow        Tomorrow's date
```

### The increment parameter

The increment parameter is optional and adds a value to any other specified timing parameter. Increment can be the plus sign (+), then a number, followed by one of the words: minutes, hours, days, weeks, months, or years. For example:

- +3 minutes – job will begin after three minutes following any other specified parameter. For example, *at now +3 minutes* will run in three minutes following the depression of the *Enter* key.
- +2 hours – job will begin two hours following any other specified parameter. For example, *at now tomorrow +2 hours* will run the command in 26 hours' time.
- +7 days – job will begin one week following any other specified parameter. Equivalent to +1 week. For example, *at 3 PM +7 days* will run the command in seven days' time at 3:00pm.

Increment can also be the word 'next' followed by minute, hour, day, week, month, or year:

- *next day* – job will begin one day following any other specified parameter. For example, *at noon next day* will run the command *at 12:00 noon* on the next calendar day.
- *next week* – job will begin one week following any other specified parameter. For example, *at now next week* will run the command at the current hour, seven days from the current date.

Note, be very careful with the increment value. If the time specified has passed for the current day when you enter the timing parameters, the command will be scheduled beginning with the *following* day, and the increment will be added to that value. Therefore, be certain to look at the resulting scheduled time carefully after entering **at** command timing parameters to ensure that you will get the desired results.

## DISPLAYING JOBS

There are two views of scheduled **at** jobs. The first is a list of the jobs with their corresponding scheduled dates and times. The other view is simply a numeric indication of the number of jobs scheduled for a user.

Examples:

- *at -l* would display all jobs for a non-root userid. A root user would see all jobs for all users.
- *at -l -o* would display all jobs in chronological order.
- *at -l -q f* would display all jobs for the user queued to the C shell queue.
- *at -n psmith* would display the quantity of scheduled jobs for user psmith.

## REMOVING JOBS

Jobs can be deleted at the individual job level or by all jobs belonging to a user.

Examples:

- *at -r psmith.1025550000.a* would delete that particular job if you are user psmith or the root user.

- *at -r -i -u psmith* would allow selective removal of a job or jobs. Each scheduled job for user psmith would display with a prompt whether or not to delete. Once the desired job is located, enter 'yes' to delete. Then you can press *Enter* to skip through the remaining jobs or just press CNTL+C.

MAIL

The **at** command uses AIX mail to the issuing user in the following ways:

1 If there is an error when the job is run.

2 If there is output produced by the job that would normally go to the display, unless redirected.

3 If the -m flag is specified, in which case the **at** command will send mail to the user indicating the job was successfully run. This is useful if you need to keep a log of important jobs or otherwise need to keep track of job results other than 1 and 2 above.

| file exists | Who can use at command? |
|---|---|
| at.allow only | Only users whose login name is in at.allow list |
| at.deny only | Any user whose login name is NOT in at.deny, unless list is empty, in which case, all users allowed |
| both at.allow & at.deny | Only users whose login name is in at.allow, and excludes those whose names are in at.deny |
| neither | Root user only |

*Figure1: Who can use the **at** command?*

USER ACCESS TO THE AT COMMAND

There are two administrative files the root user can use to control which users (including the root user) can execute the **at** command. These are the /var/adm/cron/at.allow and /var/adm/cron/at.deny files. The files can contain one user login name per line. There are four possible conditions for the two files, and user access depends on the presence or lack thereof of the two files and whose names appear in the files. The four conditions are: only the access list is present; only the deny list is present; both lists are present; or neither list is present. To learn whether a user would have access to the **at** command, consult Figure 1.

*David Chakmakian*
*Programmer (USA)*


## Extended Change Directory

INTRODUCTION

When working in a development environment, you may have a great number of subdirectories under your $HOME directory, and sometimes these subdirectories can be deeply nested. In these circumstances, navigating from one directory to another can be quite cumbersome. ECD (extended change directory) is a shell script that lets you switch interactively to any subdirectory under the $HOME directory. The shell script, once executed, presents the user with a list of all subdirectories prefixed with a unique number and then switches to the directory which is selected by the user using the number associated with that subdirectory.

ECD

```
#!  /bin/echo Usage:. ./ecd
#########################################################################
#  Name     : ecd  ( extended change directory )
```

```
#  Overview : The script displays a list of all subdirectories under
#             $HOME and then switches to a specific directory,
#             selected from the list, by user.
#  Notes    : 1. The script must be executed in current shell as
#                 follows:
#                              . cde
#             2. The script contains the following functions:
#                         o main
#                         o ProcessExit
#                         o MoveCursor
#                         o DisplayMessage
#                         o HandleInterrupt
#                         o InitialiseVariables
#                         o PrepareDirectoryListing
#                         o DisplayDirectoryList
#                         o ChangeToTargetDirectory
########################################################################
#  Name     : InitialiseVariables
#  Overview : The function initializes all working variables.
#  Notes    :
########################################################################
InitialiseVariables  ( )
{
#
SCRIPT_NAME="ecd"
# save current directory
CURDIR='pwd'
# terminal capabilities
BOLDON='tput smso'
BOLDOFF='tput rmso'
#
ESC="\ØØ33["
ERROR="${SCRIPT_NAME}:ERROR:"
INFO="${SCRIPT_NAME}:INFO:"
# fuction return values
TRUE=Ø
FALSE=1
# exit status
SEC=Ø
FEC=1
# sleep duration
SLEEP_DURATION=1
# define  signals
SIGINT=2    ;  export SIGINT     # ctrl-c command
SIGTERM=15  ;  export SIGTERM    # kill  command
SIGTSTP=18  ;  export SIGTSTP    # ctrl-z  command
# messages
INTERRUPT="Program Interrupted\; Quitting early"
WORKING="Working"
INVALID_ENTRY="Invalid Entry"
```

```
OS_ERROR="\${ERR_MSG}"
DIR_NOT_CHANGED="Failed to change to \${TARGET_DIR} directory"
#  temporary files
export TEMP_FILE_1="/tmp/${SCRIPT_NAME}_$$_1.tmp"
export TEMP_FILE_2="/tmp/${SCRIPT_NAME}_$$_2.tmp"
export DIR_LIST="/tmp/${SCRIPT_NAME}_$$_3.dir"
}
#########################################################################
#  Name     : HandleInterrupt
#  Overview : The function calls ProcessExit.
#  Notes    :
#########################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"  N
# make sure to be in current directory
cd ${CURDIR}
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     : 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}
}
ResetSignals ()
{
:
}
#########################################################################
#  Name     : ProcessExit
#  Overview : The function removes all temporary files but does not
#             actually issue an exit command  because the script is
#             meant to be running in the current shell and, therefore,
#             exit command will exit from the current shell.
#  Notes    :
#########################################################################
ProcessExit ()
{
EXIT_CODE=$1
# re-define rm comand in case an alias with definition
# rm -i  exists ; unalias does not seems to work
RM_COM="rm  -f"
#  remove temporary files
${RM_COM}  ${TEMP_FILE_1}
${RM_COM}  ${TEMP_FILE_2}
${RM_COM}  ${DIR_LIST}
```

```
}
#########################################################################
#  Name     : PrepareDirectoryListing
#  Overview : The function prepares a list of all subdirectories
#             by purging the output from the du command.
#  Notes    :
#########################################################################
PrepareDirectoryListing ( )
{
# switch to home directory
cd $HOME
du | awk {'print $2'} | sed s/.// | sed s/^/\${HOME}/  >  ${TEMP_FILE_1}
# number each line containing directory name
cat -n  ${TEMP_FILE_1}  | awk {'print $1"-->  "$2'}  >  ${DIR_LIST}
}
#########################################################################
#  Name     : DisplayDirectoryList
#  Overview : The function displays the directory listing and accepts
#             a number for the selected directory to switch to.
#  Notes    :
#########################################################################
DisplayDirectoryList ( )
{
# declare DIR_INT as an integer representing a subdirectory name
integer  DIR_INT
while  true
do
  clear
  cat  ${DIR_LIST}
  echo "${BOLDON}Select directory by number ${BOLDOFF} :\c"
  read  DIR
  case ${DIR}  in
    "" )  DisplayMessage E "${INVALID_ENTRY}"  E   ;;
     * )  (DIR_INT=${DIR}) 2>  ${TEMP_FILE_1}  ;
          if [ $? -ne  Ø  ]
          then
              ERR_MSG='cat ${TEMP_FILE_1}'  ;
              DisplayMessage E "${OS_ERROR}" N   ;
          else
              DIR_INT="${DIR}"
              if ! grep "${DIR_INT}-->"  ${DIR_LIST}  > /dev/null 2>&1
              then
                  DisplayMessage E "${INVALID_ENTRY}"  E   ;
              else
                  break ;
              fi ;
          fi ;;
  esac
done
```

```
TARGET_DIR='grep "${DIR_INT}-->"  ${DIR_LIST}   | awk {'print $2'}'
TARGET_DIR='eval echo "${TARGET_DIR}"'
}
#########################################################################
#  Name     : ChangeToTargetDirectory
#  Overview : The function switches to the selected directory.
#  Notes    :
#########################################################################
ChangeToTargetDirectory ()
{
cd ${TARGET_DIR} 2> ${TEMP_FILE_1}
if [ $?  -ne  Ø  ]
then
     DisplayMessage E "${DIR_NOT_CHANGED}"   N
     ERR_MSG='cat ${TEMP_FILE_1}'
     DisplayMessage E "${OS_ERROR}"   N
     cd ${CURDIR}
fi
}
#########################################################################
#  Name     : main
#  Overview : The function  controls the process logic.
#  Notes    :
#########################################################################
main ()
{
InitialiseVariables
PrepareDirectoryListing
DisplayDirectoryList
ChangeToTargetDirectory
ProcessExit $SEC
}
# set traps
trap "HandleInterrupt "     2        15      18
# invoke main
main
# unset traps
trap    2        15      18
```

## SAMPLE OUTPUT

```
/export/home/zamana/temp>. ecd1-->  ${HOME}/bin2-->  ${HOME}/sh3-->
${HOME}/pc4-->  ${HOME}/sql5-->  ${HOME}/temp6-->  ${HOME}/dpa/source/
check7-->  ${HOME}/dpa/source/backup8-->  ${HOME}/dpa/source9-->
${HOME}/dpa/unittest/alter_feeder_tables1Ø-->  ${HOME}/dpa/unittest/
alter_integration_tables11-->  ${HOME}/dpa/unittest/
cr_cleanse_dpa_preferences_sp12-->  ${HOME}/dpa/unittest13-->  ${HOME}/
dpa14-->  ${HOME}/log15-->  ${HOME}/.jetadmin16-->  ${HOME}/c17-->
```

```
${HOME}/dba18-->  ${HOME}/awk19-->  ${HOME}/sed2Ø-->  ${HOME}/java21-->
${HOME}/doc22-->  ${HOME}/oracle/jpub/reflect23-->  ${HOME}/oracle/jpub/
java24-->  ${HOME}/oracle/jpub/mesg25-->  ${HOME}/oracle/jpub/c26-->
${HOME}/oracle/jpub27-->  ${HOME}/oracle/sqlj/runtime/util28-->
${HOME}/oracle/sqlj/runtime/error29-->  ${HOME}/oracle/sqlj/runtime3Ø-->
${HOME}/oracle/sqlj/checker31-->  ${HOME}/oracle/sqlj/mesg32-->
${HOME}/oracle/sqlj33-->  ${HOME}/oracle/util/tcheck34-->  ${HOME}/
oracle/util35-->  ${HOME}/oracle/aurora/sqljdecl36-->  ${HOME}/oracle/
aurora37-->  ${HOME}/oracle/core/lmx38-->  ${HOME}/oracle/core/lvf39-->
${HOME}/oracle/core4Ø-->  ${HOME}/oracle/sql41-->  ${HOME}/oracle42-->
${HOME}/sqlj/demo/server43-->  ${HOME}/sqlj/demo44-->  ${HOME}/sqlj45-->
${HOME}/source46-->  ${HOME}/gemini47-->  ${HOME}/extproc48-->  ${HOME}/
sccs/SCCS/sql49-->  ${HOME}/sccs/SCCS5Ø-->  ${HOME}/sccs51-->  ${HOME}/
abinitio/.WORK52-->  ${HOME}/abinitio53-->  ${HOME}/xxx54-->  ${HOME}/
cpf55-->  ${HOME}/make
56-->  ${HOME}/dq/168
57-->  ${HOME}/dq58-->  ${HOME}/.WORK59-->  ${HOME}Select directory by
number :55/export/home/zamana/make>
```

*Arif Zaman*
*ETL Developer (UK)*


## System configuration listing program

The attached C program lists various CPU and kernel-related characteristics of IBM RS/6000 and eServer pSeries computers.

PROGRAM CODE

```
/*---------------------------------------------------------------*/
/* CPU info display for IBM RS/6ØØØ and pSeries AIX servers      */
/*---------------------------------------------------------------*/

#include <sys/types.h>
#include <fcntl.h>
#include <sys/systemcfg.h>
#include <stdio.h>
#include <nlist.h>

struct nlist    info;

void
main()
```

```
{
      int             fd;
      int             imp;


      info.n_name = "_system_configuration";

      fd = open("/dev/kmem", O_RDONLY);
      knlist(&info, 1, sizeof(struct nlist));
      lseek(fd, info.n_value, Ø);
      read(fd, &_system_configuration, sizeof(_system_configuration));

      if (_system_configuration.architecture == POWER_RS)
            printf("system architecture           : POWER_RS\n");
      else if (_system_configuration.architecture == POWER_PC)
            printf("system architecture           : POWER_PC\n");
      else if (_system_configuration.architecture == IA64) {
            printf("system architecture           : IA64\n");
      } else
            printf("system architecture           : unknown\n");


      printf("processor width               : %d\n",
_system_configuration.width);

      imp = _system_configuration.implementation;
      if (imp == POWER_RS1)
            printf("processor class               : POWER_RS1\n");
      else if (imp == POWER_RSC)
            printf("processor class               : POWER_RSC\n");
      else if (imp == POWER_RS2)
            printf("processor class               : POWER_RS2\n");
      else if (imp == POWER_6Ø1)
            printf("processor class               : POWER_6Ø1\n");
      else if (imp == POWER_6Ø3)
            printf("processor class               : POWER_6Ø3\n");
      else if (imp == POWER_6Ø4)
            printf("processor class               : POWER_6Ø4\n");
      else if (imp == POWER_62Ø)
            printf("processor class               : POWER_6Ø4\n");
      else if (imp == POWER_63Ø)
            printf("processor class               : POWER_63Ø\n");
      else if (imp == POWER_A35)
            printf("processor class               : POWER_A35\n");
      else if (imp == POWER_RS64II)
            printf("processor class               : POWER_RS64II\n");
      else if (imp == POWER_RS64III)
            printf("processor class               : POWER_RS64III\n");
      else if (imp == POWER_MPC745Ø)
            printf("processor class               : POWER_MPC745Ø\n");
```

```
      else if (imp == POWER_MPC745Ø)
            printf("processor class              : POWER_MPC745Ø\n");
      else
            printf("processor class              : unknown\n");

      imp = _system_configuration.version;
      if (imp == PV_RS1)
            printf("processor implementation     : PV_RS1\n");
      else if (imp == PV_RSC)
            printf("processor implementation     : PV_RSC\n");
      else if (imp == PV_RS2)
            printf("processor implementation     : PV_RS2\n");
      else if (imp == PV_6Ø1)
            printf("processor implementation     : PV_6Ø1\n");
      else if (imp == PV_6Ø3)
            printf("processor implementation     : PV_6Ø3\n");
      else if (imp == PV_6Ø4)
            printf("processor implementation     : PV_6Ø4\n");
      else if (imp == PV_62Ø)
            printf("processor implementation     : PV_62Ø\n");
      else if (imp == PV_63Ø)
            printf("processor implementation     : PV_63Ø\n");
      else if (imp == PV_A35)
            printf("processor implementation     : PV_A35\n");
      else if (imp == PV_RS64II)
            printf("processor implementation     : PV_RS64II\n");
      else if (imp == PV_RS64III)
            printf("processor implementation     : PV_RS64III\n");
      else if (imp == PV_MPC745Ø)
            printf("processor implementation     : PV_MPC745Ø\n");
      else if (imp == PV_MPC745Ø)
            printf("processor implementation     : PV_MPC745Ø\n");
      else if (imp == PV_M1)
            printf("processor implementation     : PV_M1\n");
      else if (imp == PV_M2)
            printf("processor implementation     : PV_M2\n");
      else
            printf("processor implementation     : unknown\n");

      printf("number of cpus               : %d\n",
_system_configuration.ncpus);
      printf("original number of cpus      : %d\n",
_system_configuration.original_ncpus);

      printf("L1 Inst Cache Size (KB)      : %d\n",
_system_configuration.icache_size / 1Ø24);
      printf("L1 Inst Cache Associativity  : %d\n",
_system_configuration.icache_asc);
      printf("L1 Inst Cache Block Size (B) : %d\n",
_system_configuration.icache_block);
```

```
      printf("L1 Inst Cache Line Size  (B)  : %d\n",
_system_configuration.icache_line);

      printf("L1 Data Cache Size   (KB)     : %d\n",
_system_configuration.dcache_size / 1Ø24);
      printf("L1 Data Cache Associativity   : %d\n",
_system_configuration.dcache_asc);
      printf("L1 Data Cache Block Size (B)  : %d\n",
_system_configuration.dcache_block);
      printf("L1 Data Cache Line Size (B)   : %d\n",
_system_configuration.dcache_line);

      printf("L2 Cache Size        (KB)     : %d\n",
_system_configuration.L2_cache_size / 1Ø24);
      printf("L2 Cache Associativity        : %d\n",
_system_configuration.L2_cache_asc);
      if ((_system_configuration.tlb_attrib & Øx1) == Ø)
            printf("TLB status                    : Not present\n");
      else if ((_system_configuration.tlb_attrib & Øx11) == 1) {
            printf("TLB status                    : Present, combined
Instruction and Data\n");
            printf("Entries in TLB                : %d\n",
_system_configuration.itlb_size);
            printf("Associativity of TLB          : %d\n",
_system_configuration.itlb_asc);
      } else {
            printf("TLB status                    : Present, separate
Instruction and Data TLB\n");
            printf("Entries in Instruction TLB    : %d\n",
_system_configuration.itlb_size);
            printf("Associativity of Instruction TLB  : %d\n",
_system_configuration.itlb_asc);
            printf("Entries in Data TLB           : %d\n",
_system_configuration.dtlb_size);
            printf("Associativity of Data TLB     : %d\n",
_system_configuration.dtlb_asc);
      }
      switch (_system_configuration.rtc_type) {
      case RTC_POWER:
            printf("Rtc type                      : RTC_POWER\n");
            break;
      case RTC_POWER_PC:
            printf("Rtc type                      : RTC_POWER_PC\n");
            break;
      case RTC_IA64:
            printf("Rtc type                      : RTC_IA64\n");
            break;
      default:
            printf("Rtc type                      : unknown\n");
      }
```

```
      if (_system_configuration.virt_alias)
            printf("Virtual aliasing              : supported\n");
      else
            printf("Virtual aliasing              : unsupported\n");
      if (__KERNEL_32())
            printf("Kernel width                  : 32bit\n");
      else
            printf("Kernel width                  : 64bit\n");
      if (_system_configuration.slb_attr & Øx1) {
            printf("SLB                           : implemented\n");
            printf("Size of SLB is                : %d\n",
_system_configuration.slb_size);
      } else
            printf("SLB                           : unimplemented\n");
}
```

The following is the output for some computers that were available for me to test my program on.

Model C20:

```
system architecture           : POWER_PC
processor width               : 32
processor class               : POWER_6Ø4
processor implementation      : PV_6Ø4
number of cpus                : 1
original number of cpus       : 1
L1 Inst Cache Size (KB)       : 16
L1 Inst Cache Associativity   : 4
L1 Inst Cache Block Size (B)  : 32
L1 Inst Cache Line Size  (B)  : 64
L1 Data Cache Size   (KB)     : 16
L1 Data Cache Associativity   : 4
L1 Data Cache Block Size (B)  : 32
L1 Data Cache Line Size (B)   : 64
L2 Cache Size        (KB)     : 1Ø24
L2 Cache Associativity        : 1
TLB status                    : Present, combined Instruction and Data
Entries in TLB                : 128
Associativity of TLB          : 2
Rtc type                      : RTC_POWER_PC
Virtual aliasing              : unsupported
Kernel width                  : 32bit
SLB                           : unimplemented
```

Model F50:

```
system architecture           : POWER_PC
processor width               : 32
processor class               : POWER_6Ø4
```

```
processor implementation      : PV_6Ø4
number of cpus                : 4
original number of cpus       : Ø
L1 Inst Cache Size (KB)       : 32
L1 Inst Cache Associativity   : 4
L1 Inst Cache Block Size (B)  : 32
L1 Inst Cache Line Size  (B)  : 32
L1 Data Cache Size   (KB)     : 32
L1 Data Cache Associativity   : 4
L1 Data Cache Block Size (B)  : 32
L1 Data Cache Line Size (B)   : 32
L2 Cache Size        (KB)     : 256
L2 Cache Associativity        : 1
TLB status                    : Present, combined Instruction and Data
Entries in TLB                : 128
Associativity of TLB          : 2
Rtc type                      : RTC_POWER_PC
Virtual aliasing              : unsupported
Kernel width                  : 32bit
SLB                           : unimplemented
```

Model p640:

```
system architecture           : POWER_PC
processor width               : 64
processor class               : POWER_63Ø
processor implementation      : PV_63Ø
number of cpus                : 4
original number of cpus       : 4
L1 Inst Cache Size (KB)       : 32
L1 Inst Cache Associativity   : 128
L1 Inst Cache Block Size (B)  : 128
L1 Inst Cache Line Size  (B)  : 128
L1 Data Cache Size   (KB)     : 64
L1 Data Cache Associativity   : 128
L1 Data Cache Block Size (B)  : 128
L1 Data Cache Line Size (B)   : 128
L2 Cache Size        (KB)     : 8192
L2 Cache Associativity        : 1
TLB status                    : Present, combined Instruction and Data
Entries in TLB                : 128
Associativity of TLB          : 2
Rtc type                      : RTC_POWER_PC
Virtual aliasing              : unsupported
Kernel width                  : 32bit
SLB                           : unimplemented
```

Model H70:

```
system architecture           : POWER_PC
processor width               : 64
```

```
processor class              : POWER_RS64II
processor implementation     : PV_RS64II
number of cpus               : 4
original number of cpus      : 4
L1 Inst Cache Size (KB)      : 64
L1 Inst Cache Associativity  : 1
L1 Inst Cache Block Size (B) : 128
L1 Inst Cache Line Size  (B) : 128
L1 Data Cache Size   (KB)    : 64
L1 Data Cache Associativity  : 2
L1 Data Cache Block Size (B) : 128
L1 Data Cache Line Size (B)  : 128
L2 Cache Size        (KB)    : 4Ø96
L2 Cache Associativity       : 1
TLB status                   : Present, combined Instruction and Data
Entries in TLB               : 512
Associativity of TLB         : 4
Rtc type                     : RTC_POWER_PC
Virtual aliasing             : unsupported
Kernel width                 : 32bit
SLB                          : unimplemented
```

Model p620:

```
system architecture          : POWER_PC
processor width              : 64
processor class              : POWER_RS64III
processor implementation     : PV_RS64III
number of cpus               : 6
original number of cpus      : 6
L1 Inst Cache Size (KB)      : 128
L1 Inst Cache Associativity  : 2
L1 Inst Cache Block Size (B) : 128
L1 Inst Cache Line Size  (B) : 128
L1 Data Cache Size   (KB)    : 128
L1 Data Cache Associativity  : 2
L1 Data Cache Block Size (B) : 128
L1 Data Cache Line Size (B)  : 128
L2 Cache Size        (KB)    : 8192
L2 Cache Associativity       : 1
TLB status                   : Present, combined Instruction and Data
Entries in TLB               : 512
Associativity of TLB         : 4
Rtc type                     : RTC_POWER_PC
Virtual aliasing             : unsupported
Kernel width                 : 32bit
SLB                          : unimplemented
```

*Alex Polak*
*System Engineer*
*APS (Israel)*



 

## sysmgt.websm.apps update issues

We have recently come across a problem when updating an existing AIX 4.3 system from the AIX 4.3.3.0 Maintenance Level. In our shop the sysmgt.websm.apps 4.3.3.0 failed to install. After much analysis it became apparent that the failure occurs only if the sysmgt.websm.apps fileset is already installed, because of a requisite to perl.rte, which is a new fileset in 4.3.3. To prevent this problem, it is essential that you install the perl.rte fileset, which is included in the AIX 4.3.3.0, Maintenance Level before updating from the AIX 4.3.3.0 Maintenance Level. We have found this problem to be quite specific to the AIX 4.3.3.0 Maintenance Level and it does not occur when updating from the AIX 4.3.3 Product Media.

*Systems Programmer (UK)* 

## Call for papers

Why not share your expertise and earn money at the same time? *AIX Update* is looking for technical articles and hints and tips about AIX performance, as well as example scripts that experienced AIX users have written to make their life, or the lives of their users, easier.

Articles can be e-mailed to Trevor Eddolls at trevore@xephon.com or sent to any of the addresses shown on page 2. A copy of our *Notes for contributors* is available from www.xephon.com/nfc.

# AIX news

IBM has announced support for AIX 5L Version 5.1 for its TotalStorage Expert Version 2.1.1, which gathers and presents information that can help storage administrators manage storage resources. It consists of Enterprise Storage Server Expert (ESS Expert) and Enterprise Tape Library Expert.

This modification of TotalStorage Expert V2.1.1 includes the base release function announced in V2.1, some integrated APAR fixes, and the capability to run on AIX 5L V5.1.

It works with AIX or Windows 2000.

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

* * *

IBM has announced the Informix Extended Parallel Server (XPS) 8.40 with optimized query performance in complex ad hoc environments, and a parallel load functionality, better data access performance, data skew management tools, and a range of other management tools.

Aimed at very large data warehouse applications, bundled products include Informix Connect Runtime V2.8, Informix Java Database Connectivity (JDBC) V1.5 and V2.21, Informix Server Administrator (ISA) V1.41, and Informix I-Spy V2.0.

Connect Runtime is the runtime libraries for the Client Software Development Kit V2.80, a collection of APIs that help speed development time for applications that work with Informix Dynamic Server (IDS), and Informix XPS.

The Informix JDBC Driver is a Java platform-independent, industry-standard type 4 JDBC driver, enabling multi-tier Java application deployment against existing Informix database servers. It also improves performance and integration with XML and IBM products like WebSphere.

ISA V1.41 is a browser-based cross-platform database server administration tool, providing an interface for the XPS command line.

Finally, I-Spy is a data warehouse monitoring and optimization tool for Informix databases. XPS runs on AIX, HP-UX, Solaris, or TRU-64 UNIX.

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

* * *

IBM has acquired TrelliSoft, which makes storage resource management software, for an undisclosed sum. It will be integrated into IBM Software Group and TrelliSoft products are now available from the Tivoli operation.

TrelliSoft specializes in Java and Web-based storage resource management to support multiple platforms, including AIX, HP-UX, Red Hat Linux, Solaris, and Windows NT/ 2000.

It’s expected to complement SRM products currently under development at IBM.

For further information contact your local IBM representative.
URL: http://www.tivoli.com/products