87

AIX update

January 2003

In this issue

AIX Update

Published by

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

North American office

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

Subscriptions and back-issues

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

AIX Update on-line

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

Editors

Trevor Eddolls

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.

Determining your networking configuration

Some Unix systems manufacturers make it easier than others to configure and maintain network configuration. For example Sun has an ASCII file, */etc/defaultrouters*, which can be edited to specify the default network routes for the system; Digital's Tru64 uses a file called */etc/rc.config* for network interface configuration and one called */etc/routes* for network routing.

IBM has SMIT. However, the information you supply through the networking panels is stored in the ODM and is therefore not easily readable. Checking what interfaces are currently up and running on a system is relatively simple, as is checking what network routes exist. But which ones will survive a system reboot? And how do you know whether any of them have been changed on the fly? Unless you have made alterations through the **ifconfig** command or the SMIT panels, these changes will be lost at reboot time.

The following program is designed to help in finding out what is stored in the ODM with regards to networking. It queries the *Customized Attributes* part of the database to find out what network interfaces are defined on the system, which are configured and which not, and what, if any, routes are defined.

There is one anomaly, though, that I know of. The SP switch interface (CSS) is not defined in the same way as other network interfaces – it has a 'parent' of sysplanar (system board) and not inet (network). Hence the program does not pick it up.

FIND_ROUTE.PL

```
#!/usr/bin/perl -w
#
# Program to check interfaces & static routes in the ODM on AIX
#
use strict;
use Data::Dumper;
my %defined;
my @undefined;
my @undefpddv;
my $if_name;
my $attr;
my @arrayoffields;
my %routes;
my $route;
my $mesg;
my $oldhandle;
my $setno;
my $msgno;
my $catalog;
$ENV{ODMDIR}="/etc/objrepos";
# Make sure we are looking at the root ODM

# Get the interfaces first : both defined ones and undefined
print "\n\n         ***************************\n";
print "         *       Interfaces        *\n";
print "         ***************************\n\n";
open(CM3,"odmget -q\"parent=inetØ and status=Ø\" CuDv|");
WHILE:while (<CM3>) {
  s/^\s+//;
  s/\s+$//;
  if ( /^name/ ) { push(@undefined,(split /\"/)[1]); }
  if ( /^PdDvLn/ ) { push(@undefpddv,(split /\"/)[1]); }
  }
close CM3;
open(CM1,"odmget -q\"parent=inetØ and status=1\" CuDv|");
WHILE:while (<CM1>) {
  s/^\s+//;
  s/\s+$//;
  CASE: {
    /^name/ and do {
      $if_name=(split /\"/)[1];
      next WHILE if ($if_name=~ /loØ/);
      open(CM2,"odmget -q\"name=$if_name\" CuAt|");
        while (<CM2>) {
          s/^\s+//;
          s/\s+$//;
          IFDET: {
            /^attribute/ and do {
              $attr=(split /\"/)[1];
              last IFDET;
            };
            /^value/ and do {
              $defined{$if_name}{$attr}=(split /\"/)[1];
              last IFDET;
            };
          }
        }
      close CM2;
      last CASE;
    };
  }
}
close CM1;
print " I/f     IP Address         Netmask       Status \n";
print "=====  ===============  ===============  =========\n";
$~="INTERFACES";
foreach (keys %defined) {
  write;
}
$~="STDOUT";
print "\n";
my $temp;
foreach $if_name (@undefined) {
  $attr=shift(@undefpddv);
  open(CM4,"odmget -q\"uniquetype=$attr\" PdDv|");
  while (<CM4>) {
    s/^\s+//;
    s/\s+$//;
    if ( /^setno/ ) { $setno=(split /=/)[1]; }
    if ( /^msgno/ ) { $msgno=(split /=/)[1]; }
    if ( /^catalog/ ) { $catalog=(split /\"/)[1]; }
  }
  close CM4;
  open(CM5,"dspmsg $catalog -s $setno $msgno|");
  $attr=<CM5>;
  close CM5;
  $~="UNDEFIF";
  write;
  $~="STDOUT";
}
print "\n";

# Get the static routes

print "         ***************************\n";
print "         *         Routes         *\n";
print "         ***************************\n";
open(CMD,"odmget -q\"attribute=route\" CuAt|");
while (<CMD>) {
  s/^\s+//;
  s/\s+$//;
  SWITCH: {
    /^value/ and do {
      push(@arrayoffields, split (/,/, (split /\"/)[1]));
      $route=$arrayoffields[scalar(@arrayoffields)-2];
      $routes{$route}{"type"}=$arrayoffields[0];
      $routes{$route}{"type"}.="work" if $arrayoffields[0]=~/^net/;
      $routes{$route}{"via"}=$arrayoffields[scalar(@arrayoffields)-1];
      if ($arrayoffields[1] =~ /-hopcount/)  {
        $routes{$route}{"hopc"}=$arrayoffields[2];
      } else {
        $routes{$route}{"hopc"}="1";
      }
      $routes{$route}{"netm"}=$arrayoffields[4] if ($arrayoffields[3] =~
/-netmask/);
      if ($arrayoffields[3] =~ /-netmask/) {
        $routes{$route}{"netm"}=$arrayoffields[4]
      } else {
        if ( (split(/\./, $routes{$route}{"via"}))[Ø] < 32 ) {
          $routes{$route}{"netm"}="255.Ø.Ø.Ø";
        } elsif ( (split(/\./, $routes{$route}{"via"}))[Ø] < 128 ) {
          $routes{$route}{"netm"}="255.255.Ø.Ø";
        } else {
          $routes{$route}{"netm"}="255.255.255.Ø";
        }
      }
      undef @arrayoffields;
      last SWITCH;
    };
  }
}
close CMD;
if ( ! %routes ) {
  print "***  No system default route has been defined  ***\n";
} else {
  print "\n***  System default route is via gateway
".$routes{"Ø"}{"via"}."  ***\n\n";
  delete($routes{"Ø"});
  print " Type         Host            Gateway           Netmask
Hopcount\n";
  print "=======  ===============  ===============  ===============
========\n";
  $~="ROUTES";
  foreach (keys %routes) {
    write;
  }
  $~="STDOUT";
}
print "\n";
format ROUTES =
@||||||  @||||||||||||||  @||||||||||||||  @||||||||||||||  @||||||||
$routes{$route}{"type"},$_,$routes{$route}{"via"},
                        $routes{$route}{"netm"},$routes{$route}{"hopc"}
.
format INTERFACES =
@<<<<  @||||||||||||||  @||||||||||||||  @||||||||
$_,$defined{$_}{"netaddr"},$defined{$_}{"netmask"},$defined{$_}{"state"}
.
```

```
format UNDEFIF =
Undefined interface : @>>>>
@<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$if_name,$attr
.
# End of program
```

FIND_ROUTE.OUT

```
         ***************************
         *       Interfaces        *
         ***************************

 I/f     IP Address        Netmask       Status
=====  ===============  ===============  =========
trØ     166.133.6.1Ø6    255.255.255.Ø      up

Undefined interface :   enØ   Standard Ethernet Network Interface
Undefined interface :   etØ   IEEE 8Ø2.3 Ethernet Network Interface

         ***************************
         *         Routes          *
         ***************************

***  System default route is via gateway 166.133.6.1  ***

 Type         Host             Gateway           Netmask      Hopcount
=======  ===============  ===============  ===============  ========
 host     166.133.9.64      166.133.6.1     255.255.255.Ø       1
```

Phil Pollard
Unix and TSM Administrator (UK) © Xephon 2003

Performance Toolbox for AIX

IBM Performance Toolbox for AIX is a licensed product providing a set of programs that enable to you to monitor the use of resources on IBM RISC System/6000 computer systems and other systems that are capable of running a data supplier daemon.

The xmservd data-supplier daemon can provide consumers of performance statistics with a stream of data. Frequency and content of each packet of performance data are determined by the consumer program. Any consumer program can access performance data from the local host and one or more remote hosts. Monitoring of performance data via the network is important and extremely useful if you know when and what to monitor. Unfortunately, that is not always, or even normally, the case. Quite often, performance problems arise and are experienced by end users without the system administrator knowing about them until it's too late to start a monitoring session. Additionally, for us, having so many remote servers, monitoring of performance data via the network causes a lot of network traffic.

So I decided to record performance data on a local disk file on remote servers, and then, with my script (ftp_perf.sh), ftp this daily performance data file from all remote servers to the dedicated performance server during the night. This enables us to collect all performance data on one server and draw the graphics every day to analyse performance problems after they occurred. One of the things that makes this product unique is that it is not hardcoded to monitor a fixed set of resources. It's dynamic in the sense that a system administrator can customize it to focus on exactly the resources that are critical for each host that must be monitored. Outstanding features for analysing a recording of performance data are provided by the azizo program. The xmservd daemon permits any system with the agent component installed to record the activity on the system at all or selected times and for any set of performance statistics. This capability is called the xmservd recording facility and is controlled through the xmservd recording configuration file.

Sample recording configuration file */etc/perf/xmservd.cf*:

```
# Configuration file by Akbas , 1Ø.Ø4.2ØØ2

# Keep files at least 2 days and let each file contain
# one days recordings

retain 2 1

# Set default sampling interval to 1 minute
```

```
frequency 6ØØØØ

# CPU statistics:

CPU/cpuØ/idle
CPU/cpuØ/kern
CPU/cpuØ/user
CPU/cpuØ/wait
CPU/cpu1/idle
CPU/cpu1/kern
CPU/cpu1/user
CPU/cpu1/wait

# System-wide CPU statistics:

CPU/glidle
CPU/glkern
CPU/gluser
CPU/glwait

# Disk statistics:

Disk/hdiskØ/busy
Disk/hdiskØ/rblk
Disk/hdiskØ/wblk
Disk/hdiskØ/xfer
Disk/hdisk1/busy
Disk/hdisk1/rblk
Disk/hdisk1/wblk
Disk/hdisk1/xfer

# Internet Protocol statistics :

IP/NetIF/enØ/ioctet_kb
IP/NetIF/enØ/ipacket
IP/NetIF/enØ/ooctet_kb
IP/NetIF/enØ/opacket

# LAN Interfaces:

LAN/fcsØ/kbytesin
LAN/fcsØ/kbytesout

# Memory statistics:

Mem/Real/%free
Mem/Real/%pinned
Mem/Real/%comp
Mem/Real/%noncomp
Mem/Real/%local
```

```
Mem/Real/%clnt

# Paging space statistics:

PagSp/%totalfree
PagSp/%totalused

# Virtual memory management statistics:

Mem/Virt/pagein
Mem/Virt/pageout
Mem/Virt/pgspgin
Mem/Virt/pgspgout
Mem/Virt/sio
Mem/Virt/steal
Mem/Virt/pagexct

# System call statistics:

Syscall/total
Syscall/read
Syscall/write
Syscall/fork
Syscall/exec

# Process statistics:

Proc/pswitch
Proc/runque
Proc/swpque

# Record Sunday through Saturday - 24 hours per day

start Ø-6 ØØ ØØ Ø-6 ØØ ØØ
```

The command **xmpeek –l** displays all available statistics that can be recorded.

All recording files created by xmservd are placed in the directory */etc/perf*. Recording filenames are azizo.yymmdd, where the part after the period is built from the day the first record was written to the file. A recording on 26 March 2002 would thus be called */etc/perf/azizo.020326*. Our conf file are configured so that two days of files are kept and each file contains one day’s recordings, which gives us time for our ftp script to transfer the file to our performance server. Additionally, in the ftp script, I convert the recordings to an ASCII file and upload it to my PC so that my Visual Basic script inserts the data into an Excel sheet and draws the graphics for each server.

Here is the script that collects azizo recordings from remote servers:

```
#!/bin/ksh
#
# Adnan Akbas , 15.Ø4.2ØØ2

# The script gets recorded performance files (azizo) from the remote
servers and ftps to a dedicated performance server.

# Function that makes ftp get for the given Perf-client to the Perf-
server

function ftp_get {

# Variables

local_dir=/perf/${perf_client}

# Getting the standard output and error output to logfile

logfile=${local_dir}/perf_ftpget_'date +%y%m%d'.log

# Error messages due to return codes

retstrØ="INFO:  FILE RECEIVED for ${perf_client} !!!"
retstr1="ERROR: Cannot change to ${local_dir} ... FILE NOT RECEIVED for
${perf_client} !!!"
retstr2="ERROR: Cannot ping  ${perf_client} ... FILE NOT RECEIVED for
${perf_client} !!!"
retstr3="ERROR: Target file does not exist. ... FILE NOT RECEIVED for
${perf_client} !!!"
retstr4="ERROR: Login failed, check user and password ... FILE NOT
RECEIVED for ${perf_client} !!!"

# Check if we are in the right directory in Perf-Server

cd ${local_dir} > $logfile 2>&1
if [ $(pwd) != ${local_dir} ]
then
   echo "$retstr1" > $logfile 2>&1
   return
fi

# Check whether the Perf-client is pingable
```

```
ping -qc3 ${perf_client} > /dev/null 2>&1
if [ "$?" != "Ø" ]
then
   echo "$retstr2" >> $logfile 2>&1
   return
fi

# Start FTP Job

print  >> $logfile 2>&1
echo "$(date)" >> $logfile 2>&1
print >> $logfile 2>&1
print >> $logfile 2>&1
echo "STARTING FTP (GET) ..... " >> $logfile 2>&1
print >> $logfile 2>&1

ftp -v -n ${perf_client} << ! >> $logfile 2>&1
user $user $password
prompt
bin
cd ${client_dir}
ls azizo.${prev_day}
get azizo.${prev_day}
bye
!

print >> $logfile 2>&1
echo "FINISHED: $(date)" >> $logfile 2>&1
print >> $logfile 2>&1

# Checking the output of ftp and
#  determining whether it is successful or not

# Checking whether the user could log in.

cat $logfile | grep "Login failed"  > /dev/null 2>&1
if [ $? -eq Ø ]
then
     echo "$retstr4" >> $logfile 2>&1
     return
fi

# Checking whether the target directory and file exists.

cat $logfile | grep "does not exist"  > /dev/null 2>&1
if [ $? -eq Ø ]
then
     echo "$retstr3" >> $logfile 2>&1
     return
fi
```

```
# Checking whether ftp is successful

cat $logfile | grep "bytes received in" > /dev/null 2>&1
if [ $? -eq Ø ]
then
     echo "$retstrØ" >> $logfile 2>&1
     return
fi

# if the script did not exit till here

echo "ERROR: $(date) ... Unknown failure ... FILE NOT RECEIVED !!!"  >>
$logfile 2>&1
return

}

# Function that makes ftp put for the given Perf-client to PC.

function ftp_put {

# Variables

local_dir=/perf/${perf_client}
target_dir=d:/perf/${perf_client}

# Getting the standard output and error output to logfile

logfile=${local_dir}/perf_ftpput_'date +%y%m%d'.log

# Error messages due to return codes

retstrØ="INFO:  FILE SENT for ${perf_client} !!!"
retstr1="ERROR: Cannot change to ${local_dir} ... FILE NOT SENT for
${perf_client} !!!"
retstr2="ERROR: Cannot ping  ${target_pc} ... FILE NOT SENT for
${perf_client} !!!"
retstr3="ERROR: Target file does not exist. ... FILE NOT SENT for
${perf_client} !!!"
retstr4="ERROR: Login failed, check user and password ... FILE NOT SENT
for ${perf_client} !!!"


# Check whether we are in the right directory in Perf-Server

cd ${local_dir} > $logfile 2>&1
if [ $(pwd) != ${local_dir} ]
then
   echo "$retstr1" > $logfile 2>&1
   return
fi

# Check whether the target PC is pingable

ping -qc3 ${target_pc} > /dev/null 2>&1
if [ "$?" != "0" ]
then
   echo "$retstr2" >> $logfile 2>&1
   return
fi

# Start FTP Job

print  >> $logfile 2>&1
echo "$(date)" >> $logfile 2>&1
print >> $logfile 2>&1
print >> $logfile 2>&1
echo "STARTING FTP (PUT) ..... " >> $logfile 2>&1
print >> $logfile 2>&1

ftp -v -n ${target_pc} << ! >> $logfile 2>&1
user $winuser $winpassword
prompt
ascii
cd ${target_dir}
put azizo.${newformat}.txt
bye
!

print >> $logfile 2>&1
echo "FINISHED: $(date)" >> $logfile 2>&1
print >> $logfile 2>&1

# Checking the output of ftp and determining if it is successful or not

# Checking whether the user could log in.

cat $logfile | grep "Login failed"  > /dev/null 2>&1
if [ $? -eq 0 ]
then
     echo "$retstr4" >> $logfile 2>&1
     return
fi

# Checking whether the target directory and file exists.

cat $logfile | grep "does not exist"  > /dev/null 2>&1
if [ $? -eq 0 ]
then
```

```
     echo "$retstr3" >> $logfile 2>&1
     return
fi

# Checking whether ftp put is successful

cat $logfile | grep "bytes sent in" > /dev/null 2>&1
if [ $? -eq Ø ]
then
     echo "$retstrØ" >> $logfile 2>&1
     return
fi

# if the script did not exit till here

echo "ERROR: $(date) ... Unknown failure ... FILE NOT SENT !!!"  >>
$logfile 2>&1
return

}

# Variables:
# The hostnames of remote hosts

client_list="spc111e1 spc122e1 rscØ23eØ spcØØ7eØ spc888e1 spc912eØ"
client_dir=/etc/perf

# the user/password info of the perf-clients.

user=perf
password=xxxxx

# the user/password info to upload the file into windows environment.

winuser=ftpuser
winpassword=xxxxx
target_pc=mypc

# Find yesterday's time stamp

prev_day='/usr/local/sbin/yesterday'
newformat="'/usr/local/sbin/yesterday| cut -c5-6'.'/usr/local/sbin/
yesterday| cut -c3-4'.2Ø'/usr/local/sbin/yesterday| cut -c1-2'"

# MAIN ##############################################

for perf_client in $client_list
do
  ftp_get
  # converting azizo file to the ascii format.

  /usr/bin/ptxtab -s ${local_dir}/azizo.${prev_day} > /dev/null 2>&1
  if [ $? -eq Ø ]
  then

     # Getting the headings out for excel insertion

     /usr/bin/grep -v "#Monitor" ${local_dir}/azizo.${prev_day}_Ø1 | /
usr/bin/sed "s/Timestamp/Date Time/g" | /usr/bin/sed "s/\./,/g" >
azizo.${newformat}.txt

      ftp_put
  else
     echo "ERROR: Cannot convert the file into ascii format ..." >>
$logfile 2>&1
  fi
done

####################################################
```

And here is my small Perl script to find yesterday's date (*/usr/local/sbin/yesterday*) used in the script above:

```
#!/usr/bin/perl -w
#
# Adnan Akbas , Ø6.Ø4.2ØØ2

use POSIX qw(strftime);
$yes_string = strftime "%y%m%d", localtime(time-864ØØ);
print "$yes_string\n";
```

The command **ptxtab -s azizo.yymmdd** converts recordings to ASCII format.

Here is an example of ptxtab spreadsheet output format (with limited statistics):

```
# Monitor: Nice Monitor – hostname: spc888eØ
Timestamp PagSp/%totalused PagSp/%totalfree Mem/Virt/pagein Mem/Virt/
pageout
"2ØØ2/Ø1/Ø7  15:36:Ø3"       27.8            72.2            8        2Ø
"2ØØ2/Ø1/Ø7  15:36:Ø7"       27.8            72.2            7        17
"2ØØ2/Ø1/Ø7  15:36:11"       27.8            72.2            3       283
"2ØØ2/Ø1/Ø7  15:36:15"       27.8            72.2            28       48
"2ØØ2/Ø1/Ø7  15:36:19"       28.2            71.8            56       41
"2ØØ2/Ø1/Ø7  15:36:23"       29.5            7Ø.5            29       38
"2ØØ2/Ø1/Ø7  15:36:27"       31.5            68.5            Ø        62
"2ØØ2/Ø1/Ø7  15:36:31"       32.4            67.6            7Ø        1
```

```
"2ØØ2/Ø1/Ø7  15:36:35"       32.6            67.4            73       32
"2ØØ2/Ø1/Ø7  15:36:39"       32.8            67.2            156       Ø
"2ØØ2/Ø1/Ø7  15:36:43"       34.5            65.5            167       4
"2ØØ2/Ø1/Ø7  15:36:47"       34.4            65.6            163       Ø
"2ØØ2/Ø1/Ø7  15:36:51"       31.1            68.9            12       57
"2ØØ2/Ø1/Ø7  15:36:55"       3Ø.2            69.8            35       34
"2ØØ2/Ø1/Ø7  15:36:59"       28.Ø            72.Ø            15        Ø
"2ØØ2/Ø1/Ø7  15:37:Ø4"       28.Ø            72.Ø            15        Ø
```

Adnan Akbas
System Administartor
Turkcell (Germany)

The for loop

The **for** loop is one of the looping constructs that cause the shell to execute a command, or sequence of commands, repeatedly. They alter the flow of control in a shell script, causing the shell to loop back and repeat commands within the loop a number of times; the **for** loop instructs the shell to repeat the sequence of operations once for each item contained within a list supplied to the loop.

THE FORMAT OF THE FOR LOOP

The format of the loop is:

```
for variable in word_list
do
      loop_body
done
```

The *variable* can be any name, provided that it does not conflict with an existing variable which you may refer to elsewhere in your script, and by convention this loop variable is often the letter **i**; you can have several consecutive loops in your script all using the same *variable* name, if you wish. In simple scripts of only a few lines this naming convention is usually satisfactory, but when loops extend over a large number of lines, or when they are nested, it is less likely to confuse a casual reader of the script if *variable* has been given a meaningful name to indicate the usage to which it is being put.

The *word_list* may be any list of words separated by blanks, such as **for file in file1 file2 file3** for example, where the word list consists of three filenames, or it can be a list of words that are generated from the output of a command, such as **for vg in $(lsvg -o)**, where the word list generated is now a list of all volume groups that are currently varied on.

The *loop_body* may be a single command, or pipeline, or a list of commands and/or pipelines. Commands are placed either one to a line or separated by command terminators. When execution of the loop is complete, sequential control of flow is resumed, and the shell executes the command that follows the loop, or, if none exists, it terminates.

When the commands in *loop_body* are first executed, *variable* takes on the value of the first word in *word_list*. Each time the commands in *loop_body* are executed, *variable* takes on the next value in *word_list*. The commands in the body of the loop are repeated until all the words in *word_list* have been used.

For example, if *word_list* is:

```
file1 file2 file3
```

then *variable* will first be assigned the string **file1**, and the commands in the body of the loop will be executed. Then *variable* will be assigned the value **file2**, and the commands will again be repeated. Finally, *variable* will be assigned the value **file3** and the body of the loop executed again. The value of the variable, $*variable*, is usually, but not necessarily, referenced within the body of the **for** loop.

The words **for**, **in**, **do**, and **done** are not commands, but are merely keywords. The keywords **for**, **do**, and **done** must appear as the first word on a new line, or as the first word following a command terminator if they are to be recognized by the shell.

Most of the time, **for** loops are formatted as shown in the preceding example and the key words **do** and **done** appear on a line by themselves, although some programmers like to put the **do** on the same line as the **for**, separated from the word list by a semi-colon. The body of the loop is usually indented so that it is possible to see at a glance which commands make up the body of the loop. This is especially important when you are using nested loops so that it is much easier to see which **do** is matched with which **done**.

SIMPLE FOR LOOP EXAMPLE

You can create and run the shell script shown in this section, but, to reduce the risk of unintentionally renaming and modifying existing files, create a new directory to practise in. Make the new directory your current directory and create three empty files with:

```
touch file1 file2 file3
```

The following script, **mvold**, renames the files by appending the suffix .old to each of them:

```
$ vi mvold

# optional cd command on this line
ls *
for i in file1 file2 file3
do
      print $i
      mv $i $i.old
done
ls *
```

If the files to be moved are not in the directory from which you intend to run **mvold**, the first command in the script should contain a **cd** command to change to the directory containing the files. The two **ls** commands are there just to show you before and after versions of the filenames in the directory.

In our example above we have specified the names of the files that we want to rename, but if all the files in the current directory need renaming you can replace the current word list with the metacharacter *****. This will expand the list to include all files in the current directory (excluding *dot* files of course), so that the body of the loop is executed once for each file.

Give the present .old files their previous names (do this manually, or, even better, create a script to do it) and add a few more files to the directory using **touch**. Then substitute the first line of **mvold** with:

```
for i in *
```

and run the command again.

You could also specify on the command line the files to be renamed, which is preferable to hardcoding the names within the script; in this way you can be selective and leave other files unchanged. To achieve this, the body of the loop must be executed once for each of the command line arguments. For example:

```
$ vi mvold

for i in $1 $2 $3
do
      print $i
      mv $i $i.old
done
```

On running the script, the shell now loops once for each of the first three positional parameters that have been assigned a value. However, this format will not rename more than three files at a time. One solution is to use the shell variable **$***, which expands to the list of all the command line arguments. This will allow **mvold** to operate on all our arguments in turn.

Substitute the first line in **mvold** with:

```
for i in $*
```

and try running the script with a list of arguments. The list of command line arguments is used with **for** loops so often that a shorthand notation is provided. If you substitute the first line with:

```
for i
```

this instructs the shell to use the value of **i** for each of the arguments in the command line in turn.

THE DIFFERENCE BETWEEN $* AND $@

Sometimes the expansion of **$*** is not always exactly the same as the list of command line arguments. For example, consider the **for** loop in the following script:

```
$ vi args

for i in $*
do
      print $i
done
print; print $1; print $2
```

If **args** is run as follows, then the output will be as expected:

```
$ args one two three

one
two
three

one
two
```

Note, though, that when **args** is run with the following command, **$*** appears to consist of three distinct words, even though there are really only two command line arguments:

```
$ args 'one two' three

one
two
three

one two
three
```

Because the shell treats everything enclosed in single quotes as a single word, and hence a single argument, **$1** is set to **one two**. However, when **$*** is expanded, **one two** is broken into two words. Quoting **$*** does not solve the problem because the shell will expand **"$*"** to **"one two three"**, which is a single word, and the **for** loop will be executed only once.

Try this by substituting the first line with:

```
for i in "$*"
```

The solution is supplied in the form of another pre-set shell variable, **$@**. When this variable appears between double quotes, the shell recognizes it as a special case. Although **$*** and **$@** are exactly the same, when they are quoted with double quotes they are not the same.

"$@"** is expanded to exactly the list of command line arguments, even if the arguments contain blank spaces; **"$@" is equivalent to **"$1" "$2"** etc.

If you now substitute the first line of **args** with the following:

```
for i in "$@"
```

and then try running **args** with **args 'one two' three,** the output will be in the required form:

```
one two
three

one two
three
```

SUMMARY OF DIFFERENCES

To summarise:

- Each of **$***, **$@**, and **"$*"** is expanded to the list of command line arguments. Arguments containing spaces or tabs will be broken into separate words as **$*** and **$@** are expanded.
- The expansion of **"$@"** is exactly the same as the list of command line arguments, and arguments containing spaces or tabs are not broken down.
- If in a **for** loop the keyword **in** and *word_list* are omitted, **"$@"** is used.

USING FOR LOOPS INTERACTIVELY

It is also possible to enter a **for** loop when using the shell interactively.

Let us assume that you are a system administrator who is trying to remove hdisk4 to hdisk7 from your system configuration since they are no longer accessible. You could, of course, create a script to do this, but this would be unnecessarily time-consuming for a relatively simple task. Instead you can do this from the command line as follows:

```
for i in 4 5 6 7
> do
> rmdev -dl hdisk$i
> done
```

The shell knows you have not entered a complete command until the final **done** keyword and so it prompts you for further commands with > each time. Sometimes in situations like the one above it may be quicker to run individual **rmdev** commands against each disk, but when you have a large number of disks it will undoubtedly be quicker to use an interactive **for** loop.

THE CONTINUE COMMAND

Suppose we wanted to use our **mvold** script on a number of directories which themselves contained sub-directories that we did not want to rename. To do this the original script must be modified so that we do not rename directories, and this can be achieved by the use of the **continue** command. For example:

```
$ vi mvold

for i in *
do
      if [[ -d $i ]]
      then
            # skip directories
            continue
      fi
      # rename the file
      mv $i $i.old
done
```

The **continue** command causes the shell to continue execution of the loop by returning to loop initialization immediately, thereby skipping the remaining commands in the loop between the **do** and **done** keywords.

In the above example, if the current value of **$**i is the name of a directory, the test succeeds and **continue** is executed. This causes the shell to skip the rest of the commands in the loop body, and to continue with the next loop iteration. If the test fails, **continue** is not executed and the renaming is performed.

MODIFYING THE CHKFS SCRIPT

Now that we are familiar with **for** loops and the **continue** command, let us modify the **chkfs** script so that we can check the space usage of all filesystem entries in our source file without having to specify filesystem names on the command line.

The modified version is shown below. The changes are shown in *italics*.

```
#!/bin/ksh
# Filename: chkfs
# Usage: chkfs
# Source file: fs_limits
#########################################################
# Version History
# Version       Date    Remarks
# 1.0                        Original Version
# 2.0                 Modified to check all filesystems in source file
#########################################################
SOURCE=/usr/local/lib/fs_limits
#------------------------------------------------
# Function: f_get_fss
# Checks that the source file exists and gets the names of all
# filesystems that have entries
#------------------------------------------------
f_get_fss()
{
      # Does the source file exist?
      if [[ ! -f $SOURCE ]]
      then
            print $SOURCE source file does not exist
            exit 1
      fi
      # Get filesystem names from source file
      FSS=$(grep "^[^#].*:" $SOURCE | tr -d ':' | sort | uniq)
}
#------------------------------------------------
# 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 Ø ]]
      then
            print $FS is not a valid filesystem name
            return 1
      fi
      # Is the filesystem mounted?
      df | tr -s ' ' | cut -d ' ' -f 7, | grep $FS >/dev/null 2>&1
      if [[ $? -ne Ø ]]
      then
            print $FS is not mounted
            return 1
      fi
}
#------------------------------------------------
# Function: f_chk_source
# Checks for multiple filesystem entries in source file
#------------------------------------------------
f_chk_source()
{
      NUM=$(grep -c ^${FS}: $SOURCE)
      if [[ $NUM -gt 1 ]]
      then
            print Multiple entries for $FS in $SOURCE
            return 1
      fi
}
#------------------------------------------------
# Function: f_get_limits
# Gets the limits for the filesystem from the source file
#------------------------------------------------
f_get_limits()
{
      CHK_TYPE=$(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"
          return 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"
            return 1
          fi
          if [[ $MAXUSED -lt 1 ]]
          then
            print max_used for $FS in $SOURCE is $MAXUSED
            print "\tValue must be from 1 to 99"
            return 1
          fi
          if [[ $MAXUSED -gt 99 ]]
          then
            print max_used for $FS in $SOURCE is $MAXUSED
            print "\tValue must be from 1 to 99"
            return 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"
            return 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"
            return 1
          fi
      else
          print chk_type for $FS in $SOURCE is $CHK_TYPE
          print "\tIt should be percent or bytes"
          return 1
      fi
}
#------------------------------------------------
# Function: f_get_fs_usage
# Gets the current filesystem 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_get_fss                       # Check source file exists
                                # and get filesystem names
for FS in $FSS
do
    f_chk_source                # Check entry in source file
    [[ $? -ne 0 ]] && continue
    f_chk_valid_fs              # Valid filesystem?
    [[ $? -ne 0 ]] && continue
    f_get_limits                # Get limits from source
    [[ $? -ne 0 ]] && continue
    f_get_fs_usage              # Get current usage
done
```

The **f_chk_args** function no longer exists since we now have no command line arguments and so do not need to check them.

You will note that the main section now calls the new **f_get_fss** function that extracts the names of all filesystems that have entries in the source file **fs_limits**. To remind you, stanzas in the source file are similar to the following:

```
/:
    chk_type = percent
    max_used = 90

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

The **f_get_fss** function first checks that the source file exists, and then extracts the filesystem names to the variable **FSS**. The

grep command matches only filesystem names starting from the beginning of a line up to the first colon, and which do not include a # as the first character; this will allow us to comment out filesystems names when we no longer require any usage information. We have also sorted the list and removed multiple entries using **uniq**, since the **f_chk_source** function, which is called later, is used to check for multiple entries of the filesystem name.

The coding which used to exist in **f_chk_source** to determine that the source file existed has now been transferred to the **f_get_fss** function, since we now only need to check this just once. If we left it in **f_chk_source**, its existence would be unnecessarily checked for in every filesystem.

The main section now contains a **for** loop, where each of the functions is called for all filesystems having stanzas in our source file. In the previous version of the script we exited when we found an error, but in this version we are checking a number of filesystems so that, should we find an error, we merely want to display a message stating what the error is, and then continue checking the remaining filesystems.

In order to do this we return the value **1** when we find an error in a function called within the loop, and then check the exit status of each function with the following line in the script:

```
[[ $? -ne Ø ]] && continue
```

If there is an error, a message will be displayed, the exit status of the function will be non-zero, and the **for** loop will start again on the next filesystem name in **$FSS**.

Tonto Kowalski
Guru (UAE)

Carriage returns in DOS and Unix

INTRODUCTION

Everybody seems to want to share data between Unix and PCs. Be it FTP, NFS, or SMB, sooner or later somebody complains about the *^M* at the end of every line of a PC file seen through vi, or the fact that a Unix text file doesn't display properly with Notepad.

These problems are caused by the fact that PCs expect every line to be finished with a carriage-return/linefeed while Unix just uses a linefeed.

Remove carriage-returns by:

```
tr -d "\r" < in-file > out-file
```

Add carriage-returns by:

```
awk '{print $Ø"\r"}' in-file > out-file
```

DLC (UK) © Xephon 2003

A script to save AIX data on a DAT tape with tar

MY PROBLEM

I have a new AIX system without ADSM or any programs like it. I have just the standard AIX tools, and I have to save my production files every day. I have the Sun program Forte, and Oracle8.

On my computer I have only a DAT tape available.

Therefore I have written a script to do the following:

- Copy files in a directory */sauv*.
- Compress them.
- Copy the compressed files to tape.

To ensure that I have copied all the files, I stop Oracle and Forte before copying starts, and I restart Oracle and Forte at the end of the back-up to disk.

THE SOLUTION

ADSM is a network-based back-up system sold by IBM and used by many organizations. There are clients for a large variety of systems (different Unix brands, Windows, Windows NT, Novell, Mac). Unfortunately, at the time of writing, there is no native Unix version.

You will have to use the SCO binary and install the iBCS2-emulator for running ADSM. This description is for ADSM Version 2 Release 1.

Many methods of performing back-ups with Unix exist, such as **dump**, **tar**, **cpio**, as well as **dd**, which are available by default on your Unix system. Also available are text-based utilities, such as **Amanda**, which is designed to add a friendlier user interface to the back-up and restore procedures. Finally, commercial back-up utilities are also available, such as **BRU**.

The procedures for performing a back-up and restore will differ depending on your choice of a back-up solution. For this reason we will discuss methods for performing back-ups with the traditional Unix tools:

- **tar**
- **dump**, which is a command-line back-up tool.

The idea of making a back-up is to copy as many files as possible on your system, but some exceptions do exist, as shown below. It is not logical to include these in your back-up because of time and space. The major exceptions to inclusion in your back-up are:

- The */proc* file system – it contains only data that the kernel generates automatically, it is never a good idea to back it up.
- The */mnt* file system – it is where you mount your removable media like CD-ROM, floppy disk, etc.
- The back-up directory or media where you have placed your back-up files, such as a tape, CD-ROM, NFS mounted file system, remote/local directory, or other kinds of media.
- Software that can be easily reinstalled, though it may have configuration files that it is important to back up, otherwise you will have to do the work of configuring them all over again. I recommend putting the configuration files for software on a floppy disk.

THE TAR BACK-UP PROGRAM

The **tar** back-up program is an archiving program designed to store and extract files from an archive file known as a tarfile. A tarfile may be made on a tape drive; however, it is also common to write a tarfile to a normal file.

A simple back-up is when you decide to make a back-up of files on your system. You must choose a back-up scheme before beginning your back-up procedure. A lot of strategic back-up schemes exist, and which one you use depends on the back-up policies you want to use. In the following, we show you one back-up scheme that you could use, which takes advantage of the **tar** program's capabilities. This scheme is to first back up everything once, then back up everything that has been modified since the previous back-up.

- The first back-up is called a full back-up
- The subsequent ones are incremental back-ups.

With six tapes you can make back-ups every day. The procedure is to use tape 1 for the first full back-up *Friday 1*, and tapes 2 to 5 for the incremental back-ups *Monday through Thursday*. Then, you make a new full back-up on tape 6 *second Friday*, and start doing incremental ones with tapes 2 to 5 again. It's important to keep tape 1 untouched until you've got a new full back-up with tape 6.

In the following example, we assume that we write the back-up to a SCSI tape drive named */dev/st0*, and we back up the home directory */home* of our system. First of all, we must move to the file system */* partition. When creating an archive file, **tar** will strip the leading */* character from file path names. This means that restored files may not end up in the same locations that they were backed up from. Therefore, to solve the problem, change to the */* root directory before making all back-ups and restores.

To move to the */* root directory, use the command:

```
[root@deep]# cd /
```

It is important always to start with a full back-up, say on a Friday, for example:

Friday 1 – use tape 1 for the first full back-up:

```
[root@deep] /# cd /
[root@deep] /# tar cpf /dev/stØ –label=" full-backup created on 'date
'+%d-%B-%Y''." \
      –directory / home
```

Monday – use tape 2 for the incremental back-up:

```
[root@deep] /# cd /
[root@deep] /# tar cpNf /dev/stØ —label=" full-backup created on 'date
'+%d-%B-%Y''." \
      —directory / home
```

Tuesday – use tape 3 for the incremental back-up:

```
[root@deep] /# cd /
[root@deep] /# tar cpNf /dev/stØ —label=" full-backup created on 'date
'+%d-%B-%Y''." \
      —directory / home
```

Wednesday – use tape 4 for the incremental back-up:

```
[root@deep] /# cd /
[root@deep] /# tar cpNf /dev/stØ —label=" full-backup created on 'date
'+%d-%B-%Y''." \
      —directory / home
```

Thursday – use tape 5 for the incremental back-up:

```
[root@deep] /# cd /
[root@deep] /# tar cpNf /dev/stØ —label=" full-backup created on 'date
'+%d-%B-%Y''." \
      —directory / home
```

Friday 2 – use tape 6 for the new full back-up:

```
[root@deep] /# cd /
[root@deep] /# tar  cpf /dev/stØ —label=" full-backup created on 'date
'+%d-%B-%Y''." \
      —directory / home
```

Now, start doing incremental back-ups with tapes 2 to 5 again, and so on.

The options are:

- The **c** option specifies that an archive file is to be created.
- The **p** option preserves permissions – file protection information will be remembered.
- The **N** option does an incremental back-up and stores only files newer than DATE.
- The **f** option states that the very next argument will be the name of the archive file or device being written.

Notice how a filename that contains the current date is derived – simply by enclosing the date command between two back-quote characters. A common naming convention is to add a tar suffix for non-compressed archives, and a tar.gz suffix for compressed ones. Since we aren't able to specify a filename for the back-up set, the **-label** option can be used to write some information about the back-up set into the archive file itself. Finally, only the files contained in the */home* are written to the tape.

Because the tape drive is a character device, it is not possible to specify an actual file name. Therefore, the file name used as an argument to tar is simply the name of the device, */dev/st0*, the first tape device. The */dev/st0* device does not rewind after the back-up set is written. Therefore it is possible to write multiple sets on one tape. You may also refer to the device as */dev/st0*, in which case the tape is automatically rewound after the back-up set is written. When working with tapes you can use the following commands to rewind and eject your tape:

```
[root@deep] /# mt -f  /dev/stØ offline
```

To reduce the space needed on a **tar** archive, the back-ups can be compressed with the **z** option of the **tar** program. Unfortunately, using this option to compress back-ups can cause trouble. Because of the nature of how compression works, if a single bit in the compressed back-up is wrong, all the rest of the compressed data will be lost. So, using compression with the z option is not recommended when making back-ups with the **tar** command.

If your back-up doesn't fit on one tape, you'll need to use the -multi-volume **-M** option:

```
[root@deep] /# cd /
[root@deep] /# tar cMpf /dev/stØ /home
```

Prepare volume #2 for */dev/st0* and hit return.

After you have made a back-up, you should check that it is OK, using the -compare **-d** option as shown below:

```
[root@deep] /# cd /
[root@deep] /# tar dvf /dev/stØ
```

To perform a back-up of your entire system, use the following command:

```
[root@deep] /# cd /
[root@deep] /# tar  cpf /archive/full-backup-'date '+%d-%B-%Y''.tar \
 –directory / –exclude=proc –exclude=mnt –exclude=archive \
 –exclude=cache –exclude=*/lost+found .
```

- The -directory option tells **tar** to first switch to the following directory path, the / directory in this example, prior to starting the back-up.
- The -exclude options tells **tar** not to bother backing up the specified directories or files.
- The **.** character at the end of the command tells **tar** that it should back up everything in the current directory.

When backing up your file systems, do not include the */proc* pseudo-file-system! The files in */proc* are not actually files but are simply file-like links, which describe and point to kernel data structures. Also, do not include the */mnt*, */archive*, and all lost+found directories.

AUTOMATING BACK-UPS WITH TAR

It is always interesting to automate the back-up tasks. Automation offers enormous opportunities for using your Unix server to achieve the goals you set. The following example below is our back-up script, called back-up.cron. This script is designed to run on any computer by changing only four variables:

- COMPUTER
- DIRECTORIES
- BACKUPDIR
- TIMEDIR.

We suggest that you set this script up and run it at the beginning of the month for the first time, and then run it for a month before making major changes. In our example below we do the back-up to a directory on the local server BACKUPDIR, but you could modify this script to do it to a tape on the local server or via an NFS mounted file system.

Create the back-up script back-up.cron file, open */etc/cron.daily/backup.cron* and add the following lines to this back-up file:

```
#!/bin/sh
#Change the 5 variables below to fit your computer/back-up

COMPUTER=deep # name of this computer
DIRECTORIES="/home"         # directories to back-up
BACKUPDIR=/backups          # where to store the back-ups
TIMEDIR=/backups/last-full  # where to store time of full back-up
TAR=/bin/tar                # name and location of tar

#You should not have to change anything below here

PATH=/usr/local/bin:/usr/bin:/bin
DOW='date +%a'     # Day of the week eg Mon
DOM='date +%d'     # Date of the Month eg 27
DM='date +%d%b'    # Date and Month eg 27Sep

# On the 1st of the month a permanent full backup is made
# Every Sunday a full backup is made - overwriting last Sunday's backup
# The rest of the time an incremental backup is made. Each incremental
# backup overwrites last weeks incremental backup of the same name.
#
# if NEWER = "", then tar backs up all files in the directories
# otherwise it backs up files newer than the NEWER date. NEWER
# gets its date from the file written every Sunday.

# Monthly full backup
if [ $DOM = "Ø1" ]; then
        NEWER=""
        $TAR $NEWER -cf $BACKUPDIR/$COMPUTER-$DM.tar $DIRECTORIES
fi

# Weekly full backup
if [ $DOW = "Sun" ]; then
        NEWER=""
        NOW='date +%d-%b'

        # Update full backup date
        echo $NOW > $TIMEDIR/$COMPUTER-full-date
        $TAR $NEWER -cf $BACKUPDIR/$COMPUTER-$DOW.tar $DIRECTORIES

# Make incremental backup - overwrite last week's
else
```

```
        # Get date of last full backup
        NEWER="—newer 'cat $TIMEDIR/$COMPUTER-full-date'"
        $TAR $NEWER -cf $BACKUPDIR/$COMPUTER-$DOW.tar $DIRECTORIES
fi
```

BACK-UP DIRECTORY

Here is an abbreviated look at the back-up directory after one week:

```
[root@deep] /# ls -l /backups/

total 22217
-rw-r—r—    1 root    root  1Ø731288  Feb  7 11:24 deep-Ø1Feb.tar
-rw-r—r—    1 root    root      6879  Feb  7 11:24 deep-Fri.tar
-rw-r—r—    1 root    root      2831  Feb  7 11:24 deep-Mon.tar
-rw-r—r—    1 root    root      7924  Feb  7 11:25 deep-Sat.tar
-rw-r—r—    1 root    root  11923Ø13  Feb  7 11:24 deep-Sun.tar
-rw-r—r—    1 root    root      5643  Feb  7 11:25 deep-Thu.tar
-rw-r—r—    1 root    root      3152  Feb  7 11:25 deep-Tue.tar
-rw-r—r—    1 root    root      4567  Feb  7 11:25 deep-Wed.tar
drwxr-xr-x  2 root    root      1Ø24  Feb  7 11:2Ø last-full
```

The directory to store the back-ups, BACKUPDIR, and the directory to store the time of the full back-up, TIMEDIR, must exist or be created before the back-up script is used, or you will receive an error message.

If you are not running this back-up script from the beginning of the month, the incremental back-ups will need the time of the Sunday back-up to be able to work properly. If you start in the middle of the week, you will need to create the time file in the TIMEDIR directory. To do this, use the following command:

```
[root@deep] /# date +%d%b < /backups/last-full/myserver-full-date
```

where */back-ups/last-full* is our variable TIMEDIR, in which we want to store the time of the full back-up, and *myserver-full-date* is the name of our server, eg deep, and our time file consists of a single line with the present date, eg 15-Feb.

Make this script executable and change its default permissions to be writable only by the super-user root 755:

```
[root@deep] /# chmod 755 /etc/cron.daily/backup.cron
```

Because this script is in the */etc/cron.daily* directory, it will be automatically run as a cron job at one o'clock in the morning every day.

RESTORE FILES WITH TAR

More important than performing regular back-ups is having them available when we need to recover important files! In this section, we will discuss methods for restoring files that have been backed up with the **tar** command.

The following command will restore all files from the full-backup-Day-Month-Year.tar archive, which is an example back-up of our home directory created from the example tar commands shown above:

```
[root@deep] /# cd /
[root@deep] /# tar xpf /dev/stØ/full-backup-Day-Month-Year.tar
```

The above command extracts all files contained in the compressed archive, preserving original file ownership and permissions.

- The **x** option stands for extract.
- The **p** option preserve permissions – file protection information will be remembered.
- The **f** option states that the very next argument will be the name of the archive file or device.

If you do not need to restore all the files contained in the archive, you can specify one or more files that you wish to restore. To do this, use the following command:

```
[root@deep]# cd /
[root@deep]# tar xpf /dev/stØ/full-backup-Day-Month-Year.tar \
 home/wahib/Personal/Contents.doc home/quota.user
```

The above command restores the */home/wahib/Personal/Contents.doc* and */home/quota.user* files from the archive.

If you just want to see what files are in the back-up volume, use the -list or **-t** option:

```
[root@deep] /# tar tf /dev/stØ
```

If you have files on your system set with the immutable bit, using the **chattr** command, these files will not be remembered with the immutable bit from your restored back-up. You must reset it as immutable with the command **chattr +i** after the back-up is completed.

Don't forget to test the ability to recover from back-ups. For many system administrators, recovering a file from a back-up is an uncommon activity. This step assures that if you need to recover a file, the tools and processes will work. Performing this test periodically will help you to discover problems with the back-up procedures so you can correct them before losing data. Some back-up restoration software does not accurately recover the correct file protection and file ownership controls. Check the attributes of restored files to ensure they are being set correctly. Periodically test to ensure that you can perform a full system recovery from your back-ups.

For further documentation, and more details, there is a **man** page you can read.

I hope this will help people to back-up AIX files without having to buy a special program, such as ADSM.

Claude Dunand
Systems Programmer (France)

Network back-up manager

Editor's note: this article continues our look at backing up AIX systems with a different approach from the previous article.

In general, tape devices are used to back up files because storing data is cheaper on tape cartridges than on disk. However, disk prices are getting lower and lower, so disk space can be used to back up files that are relatively small. Most of the back-up operations consist of periodical back-ups. Files or directories are backed up at some interval and kept for some days. Besides, a back-up is often needed when a change is made to a critical file in case an error is encountered after the change. For example, application programmers want to back up programs before they make a change in their source code. Network back-up manager programs are written to utilize disk space for back-up purposes.

The following command is used to back-up files or directories:

```
nbmcopy –p path_name –f file_name –r retention period –s server_name –g
server_group_number –n -c
```

The file or directory specified by *path_name* and *file_name* are copied to a directory according to **-s**, **-g**, and **-n** parameters. Back-up paths are defined in the network. A back-up group number is given to each back-up path. Back-up paths can be grouped by location or some other parameters. If the *-s server_name* parameter is given, the file or directory is copied to a back-up path on the specified server in the network. If the *-g server_group_number* parameter is given, a back-up path in the specified group is used. To back up very critical data, a group in another location can be used for disaster recovery. If the **-n** option is given, it means a back-up path is used where the file or directory does not reside. The **-r** retention period parameter specifies how long the back-up files will be kept in the back-up paths.

If a file is given, the file is copied to the back-up path with a new name, which has a version number at the end. If a directory is given, the directory is archived by the **tar** command, compressed, and copied to a back-up path with the new name. These steps are run on pipes in order to use less temporary space. The *NBM_TEMPDIR* environment variable keeps the directory name that is used for the temporary operations.

The **nbmclean** command runs after a specified interval. It scans all files backed up by **nbmcopy** and deletes the back-up files if they have expired.

To see which files or directories are backed up by **nbmcopy**, the following command is used:

```
nbmlist –p path_name –f file_name
```

In the *path_name* and *file_name* parameters, the characters _ and *%* can be used for any character and any string respectively.

Files and directories are restored using the following command:

```
nbmrestore –p path_name –f file_name –v version –a after_time –b
before_time
```

The **-v** option can be 0, -1, -2, and so on. If 0 is given, the last back-up file is restored. If -1 is given, the previous back-up file is restored. All back-up files after or before a specified time can be displayed, and one of them can be chosen to be restored by using the **-a** and **-b** parameters. Back-up files are restored to the temporary directory specified by the *NBM_TEMPDIR* variable.

The **nbmcopy**, **nbmrestore**, and **nbmlist** programs must be copied to all servers in the network. The database containing the network back-up manager tables must be catalogued on all servers since the tables must be reached by all the servers. The **nbmclean** program can be run on only one server. It will delete all files on all servers.

The following tables are used:

- backup_paths:
  - path_no dec(3),
  - host varchar(10),
  - path varchar(200),
  - group_no dec(2),
  - username char(20),
  - server_type char(5)
- backup_versions:
  - path_name varchar(200),

    - file_name varchar(100),
    - version dec(5)
- backup_files:
    - host varchar(10),
    - path_name varchar(200),
    - file_name varchar(100),
    - backup_path_no dec(3),
    - version dec(5),
    - date_taken timestamp,
    - retpd dec(5),
    - file_type dec(1),
    - compressed dec(1).

NBMCOPY.SQC

```
#include <stdio.h>
#include <sqlca.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdlib.h>
#include <sqlenv.h>
#include <sys/wait.h>
char path_name_with[200],file_to_be_copied[400];
int random_number,not_on_this_host=0,interactive_mode=0;
char pipe1[200],pipe2[200];
char tempfilename[200];
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION ;
  char backup_path_name[200];
  char backup_host[10];
  char username[20];
  char path_name[200];
  char file_name[200];
  char preferred_host_name[10];
  long path_no;
  long file_version;
  short retpd;
  short preferred_group_num;
  char server_type[5];
  char host_name[1Ø];
  short is_dir=Ø;
  short compress=Ø;
EXEC SQL END DECLARE SECTION;
void nbm_help()
{
  printf("Usage:\n");
  printf("  nbmcopy -p path_name -f file_name -n ( not_on_this_path )
\n");
  printf("  -s server_name -g server_group_num -c compress -i (
interactive mode )\n");
}
void exit_program(int exit_code)
{
  char sys_cmd[1ØØØ];
  int rc;
  sprintf(sys_cmd,"rm %s %s %s ./size_of_files.%ld ./dfout.%ld >/dev/
null 2>&1 ",tempfilename,pipe1,pipe2,random_number,random_number);
  rc=system(sys_cmd);
  if ( rc != Ø && rc != 512 ) perror("Remove:");
  exit(exit_code);
}
void sql_error(int err_pl)
{
  printf("SQLCODE=%ld\n",sqlca.sqlcode);
  printf("Place of the error=%d\n",err_pl);
  exec sql rollback;
  exit_program(1);
}
void get_random_number()
{
  struct timeval tv;
  long msec;
  gettimeofday(&tv, NULL);
  msec=tv.tv_usec;
  srand(msec);
  random_number=rand();
}
int get_file_size()
{
  long size_of_files,tot_size=Ø;
  struct stat filestat;
  if ( stat(file_to_be_copied,&filestat) != Ø ) {
    printf("Problem in getting file status...\n");
    exit_program(1Ø1);
  }
  if ( filestat.st_mode & S_IFDIR ) {
    is_dir=1;
    return get_dir_size();
  } else {
    is_dir=Ø;
    return filestat.st_size/1Ø24;
  }
}
int get_dir_size()
{
  FILE *fp1;
  char sys_cmd[3ØØ],temp_file_name[5Ø],dummy_str[3ØØ];
  long size_of_files,tot_dir_size=Ø;
  sprintf(sys_cmd,"du -sk %s > ./
size_of_files.%ld",file_to_be_copied,random_number);
  if ( system(sys_cmd) != Ø ) {
    printf("Problem in du command...\n");
    exit_program(1Ø2);
  }
  sprintf(temp_file_name,"./size_of_files.%ld",random_number);
  if ( ( fp1=fopen(temp_file_name,"r") ) == NULL ) {
    printf("Problem in opening ./size_of_files...\n");
    exit_program(1Ø3);
  }
  while (fscanf(fp1,"%d%s",&size_of_files,&dummy_str)!=-1)
    tot_dir_size+=size_of_files;
  fclose(fp1);
  return tot_dir_size;
}
long get_freespace()
{
  FILE *fp1;
  char sys_cmd[1ØØØ],temp_file_name[1ØØØ];
  long freespace;
  if ( strcmp(backup_host,host_name) == Ø )
    if ( strcmp(server_type,"AIX  ") == Ø )
      sprintf(sys_cmd,"df -k %s | grep -v 'Filesystem    1Ø24-blocks
Free'| awk '{print $3}' > ./dfout.%ld",backup_path_name,random_number);
    else
      sprintf(sys_cmd,"df -k %s | grep -v 'Filesystem    1Ø24-blocks
Free'| awk '{print $4}' > ./dfout.%ld",backup_path_name,random_number);
  else
    if ( strcmp(server_type,"AIX  ") == Ø )
      sprintf(sys_cmd,"rsh %s -l %s 'df -k %s | grep -v Filesystem'| awk
'{print $3}' > ./
dfout.%ld",backup_host,username,backup_path_name,random_number);
    else
      sprintf(sys_cmd,"rsh %s -l %s 'df -k %s | grep -v Filesystem'| awk
'{print $4}' > ./
```

```
dfout.%ld",backup_host,username,backup_path_name,random_number);
  if ( system(sys_cmd) != Ø ) {
    printf("Problem in df command...\n");
    exit_program(1Ø4);
  }
  sprintf(temp_file_name,"./dfout.%ld",random_number);
  if ( ( fp1=fopen(temp_file_name,"r") ) == NULL ) {
    printf("Problem in opening dfout file...\n");
    exit_program(1Ø5);
  }
  fscanf(fp1,"%d",&freespace);
  fclose(fp1);
  return(freespace);
}
void get_a_path(long file_size)
{
  if ( strcmp(preferred_host_name,"") == Ø  ) {
    if ( preferred_group_num != Ø  ) {
      exec sql declare path_crs1 cursor for
               select PATH_NO,PATH,HOST,rtrim(USERNAME),SERVER_TYPE from
BACKUP_PATHS
               where GROUP_NO=:preferred_group_num;
      if ( sqlca.sqlcode < Ø ) sql_error(1);
      exec sql open path_crs1;
      if ( sqlca.sqlcode < Ø ) sql_error(2);
      exec sql fetch path_crs1
               into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
      if ( sqlca.sqlcode < Ø ) sql_error(3);
      while ( sqlca.sqlcode == Ø ) {

        if ( file_size < get_freespace()) break;

        exec sql fetch path_crs1
                 into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
        if ( sqlca.sqlcode < Ø ) sql_error(4);
      }
      if ( sqlca.sqlcode == 1ØØ ) {
        exec sql close path_crs1;
        printf("No path found ( Size=%ld) \n",file_size);
        exit_program(1Ø6);
      }
      exec sql close path_crs1;
      if ( sqlca.sqlcode < Ø ) sql_error(5);
    } else {
      if ( not_on_this_host == 1  ) {
        exec sql declare path_crs2 cursor for
                 select PATH_NO,PATH,HOST,rtrim(USERNAME),SERVER_TYPE
from BACKUP_PATHS
```

```
                  where HOST!=:host_name;
         if ( sqlca.sqlcode < Ø ) sql_error(6);
         exec sql open path_crs2;
         if ( sqlca.sqlcode < Ø ) sql_error(7);
         exec sql fetch path_crs2
                  into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
         if ( sqlca.sqlcode < Ø ) sql_error(8);

         while ( sqlca.sqlcode == Ø ) {

           if ( file_size < get_freespace()) break;

           exec sql fetch path_crs2
                    into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
           if ( sqlca.sqlcode < Ø ) sql_error(9);
         }

         if ( sqlca.sqlcode == 1ØØ ) {
           exec sql close path_crs2;
           printf("No path found ( Size=%ld) \n",file_size);
           exit_program(1Ø7);
         }
         exec sql close path_crs2;
         if ( sqlca.sqlcode < Ø ) sql_error(1Ø);
       } else {
         exec sql declare path_crs3 cursor for
                  select PATH_NO,PATH,HOST,rtrim(USERNAME),SERVER_TYPE
from BACKUP_PATHS;
         if ( sqlca.sqlcode < Ø ) sql_error(11);
         exec sql open path_crs3;
         if ( sqlca.sqlcode < Ø ) sql_error(12);
         exec sql fetch path_crs3
                  into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
         if ( sqlca.sqlcode < Ø ) sql_error(13);

         while ( sqlca.sqlcode == Ø ) {

           if ( file_size < get_freespace()) break;

           exec sql fetch path_crs3
                    into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
           if ( sqlca.sqlcode < Ø ) sql_error(14);
         }

         if ( sqlca.sqlcode == 1ØØ ) {
           exec sql close path_crs3;
```

```
          printf("No path found ( Size=%ld) \n",file_size);
          exit_program(1Ø8);
        }
        exec sql close path_crs3;
        if ( sqlca.sqlcode < Ø ) sql_error(15);
      }
    }
  } else {
    exec sql declare path_crs4 cursor for
             select PATH_NO,PATH,HOST,rtrim(USERNAME),SERVER_TYPE from
BACKUP_PATHS
                    where HOST=:preferred_host_name;
    if ( sqlca.sqlcode < Ø ) sql_error(16);
    exec sql open path_crs4;
    if ( sqlca.sqlcode < Ø ) sql_error(17);
    exec sql fetch path_crs4
             into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
    if ( sqlca.sqlcode < Ø ) sql_error(18);
    while ( sqlca.sqlcode == Ø ) {

      if ( file_size < get_freespace()) break;
      exec sql fetch path_crs4
               into
:path_no,:backup_path_name,:backup_host,:username,:server_type;
      if ( sqlca.sqlcode < Ø ) sql_error(19);
    }
    if ( sqlca.sqlcode == 1ØØ ) {
      exec sql close path_crs4;
      printf("No path found ( Size=%ld) \n",file_size);
      exit_program(1Ø9);
    }
    exec sql close path_crs4;
    if ( sqlca.sqlcode < Ø ) sql_error(2Ø);
  }
}
int get_version_number(char *path_name,char *file_name)
{
  exec sql select VERSION into :file_version
           from backup_versions
           where PATH_NAME=:path_name and
                 FILE_NAME=:file_name;
     if ( sqlca.sqlcode < Ø ) sql_error(21);
  if ( sqlca.sqlcode==1ØØ ) {
    exec sql insert into backup_versions
values(:path_name,:file_name,1);
    if ( sqlca.sqlcode < Ø ) sql_error(22);
    file_version=Ø;
  } else {
    exec sql update backup_versions
```

```
              set VERSION=VERSION+1
              where PATH_NAME=:path_name and
                  FILE_NAME=:file_name;
      if ( sqlca.sqlcode < Ø ) sql_error(23);
  }

  return file_version;
}
void do_copy()
{
  char *getenv(),*tempdir,sys_cmd[1ØØØ];
  pid_t childpid1,childpid2;
  union wait *status;
  int ind,rc;
  tempdir=getenv("NBM_TEMPDIR");
  sprintf(pipe1,"%s/pipe1.%ld",tempdir,random_number);
  sprintf(pipe2,"%s/pipe2.%ld",tempdir,random_number);
  sprintf(tempfilename,"%s/tempfile.%ld",tempdir,random_number);
  for (ind=Ø;ind<=strlen(path_name);ind++)
    if ( path_name[ind] == '/' ) path_name_with[ind]='!'; else
path_name_with[ind]=path_name[ind];
  umask(Ø);
  if ( rc=mknod(pipe1, S_IFIFO|Ø6ØØ,Ø) != Ø ) {
    printf("Problem in creating pipe1. rc=%ld\n",rc);
    exit_program(11Ø);
  }
  if ( rc=mknod(pipe2, S_IFIFO|Ø6ØØ,Ø) != Ø ) {
    printf("Problem in creating pipe1. rc=%ld\n",rc);
    exit_program(111);
  }
  if ( is_dir == 1 ) {
    childpid1=fork();
    if ( childpid1 == -1 ) {
      printf("Problem in fork process.\n");
      exit_program(112);
    }
    if ( childpid1 == Ø ) {
      sprintf(sys_cmd,"tar -cvf %s %s/%s >/dev/null 2>&1
",pipe1,path_name,file_name);
      system(sys_cmd);
      exit(Ø);
    } else {
      if ( strcmp(backup_host,host_name) == Ø ) {
        childpid2=fork();
        if ( childpid2 == -1 ) {
          printf("Problem in fork process.\n");
          exit_program(113);
        }
        if ( childpid2 == Ø ) {
          sprintf(sys_cmd,"compress -c < %s > %s  ",pipe1,pipe2);
          system(sys_cmd);
          exit(Ø);
        } else {
          sprintf(sys_cmd,"cp %s %s/%s!at!%s!%s.V%ld.Z  > /dev/
null",pipe2,backup_path_name,path_name_with,host_name,file_name,file_version);
        }
      } else {
        sprintf(sys_cmd,"compress -c < %s > %s  ",pipe1,tempfilename);
        if ( system(sys_cmd) != Ø ) {
          printf("Problem in compressing file:%s ...\n",pipe1);
          exit_program(114);
        }
        wait(status);
        sprintf(sys_cmd,"rcp %s %s@%s:%s/%s!at!%s!%s.V%ld.Z  > /dev/
null",tempfilename,username,backup_host,backup_path_name,path_name_with,
host_name,file_name,file_version);
      }
    }
```

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

Abdullah Ongul
DBA
Disbank (Turkey)

AIX news

IBM has announced an ultra-dense eServer p655 Unix server targeted at the high performance computing market that's capable of reaching half a trillion operations per second in a single frame at peak processing power. The new eServer packs up to 128 POWER4 processors per frame and is available in four or eight-CPU building blocks. It will run AIX 5L 5.1 and Linux.

The system can be clustered using eServer cluster 1600, combined using a high-performance switch. Also, systems can be defined using logical partitioning. Cluster systems administration from a single control workstation is provided by IBM's cluster management application.

Autonomic computing capabilities include integrated service processor and Chipkill and bit-steering memory.

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

* * *

TenFold has announced a new release of its Universal Application software, aimed at large organizations needing to build complex applications, promising to do so three to ten times faster than with other available technologies. The product runs on AIX 5.1L in 64-bit mode.

Performance improvements include EfficientMessageDispatch, which reduces applications server CPU usage so that Universal Application supports many more end users without additional hardware resources.

The BrowserClient AutoTest runs automated portable regression tests. Improved usability comes via EditableGrids, which let BrowserClient end-users change multiple rows of data without waiting for a browser screen refresh. Also, users can define custom HTML for starting UA transactions using WelcomePage.

For further information contact:
TenFold, 698 West 10000 South, Suite 200, South Jordan, Utah 84095, USA.
Tel: (801) 495 1010.
URL: http://www.10fold.com.

* * *

Opsware has released Version 3.5 of its Opsware System data centre automation software, which has better assimilation capabilities, disaster recovery functionality, and support for new operating systems. The product now runs under AIX.

It automates data centre operations such as server provisioning, application configuration and deployment, and security management.

The new release automates multi-data centre management from a single location and discovery and assimilation of servers already running in an existing data centre environment.

For further information contact your local Opsware, 599 N Mathilda Ave, Sunnyvale, CA 94085, USA.
Tel: (408) 744 7300.
URL: http://www.opswareinc.com/news/releases/11-13-02.htm.

* * *