74

AIX

December 2001

update

In this issue

AIX Update

Published by

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

North American office

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

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1998 issue, are available separately to subscribers for £16.00 ($24.00) each including postage.

AIX Update on-line

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

Editors

Trevor Eddolls and Richard Watson

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.

Shell functions

The Unix shell in its various incarnations is a powerful programming tool. There's the Bourne Shell (sh), the Korn Shell (ksh), the C Shell (csh), and the Bourne Again Shell (bash).

Most people are familiar with the ability of the shell to interpret commands as they are typed in at the terminal. Fewer are familiar with its ability to execute programs or shell scripts. Fewer still are familiar with the function creation and execution features of shell scripts.

The ability to define and execute functions within a shell script is actually built into the shell – whichever one you are using. Rarely do you use single command lines that are typed in at the terminal to define or execute a function, and it is much more common to see them inside scripts. But the ability to define and execute functions directly on the command line provides an interesting way to learn more about how they work.

Functions are a powerful feature of shell programming for the Bourne Shell and its relatives, the Korn Shell and the Bourne Again Shell.

If you are working in an AIX environment, you are probably already using the Korn Shell, but, to check, type ksh to start another Korn shell and press *Enter*.

A function is a single command that can be defined to stand in for one or more Unix commands, including calls to other defined functions. Using a function involves two steps:

1 Defining the function.

2 Invoking or executing the function.

The first step, to ensure the safe creation of a function, is to make sure that the function name you intend to use is not currently in use. In this case you are going to create a function named 'greetings', so we need to make sure this doesn't already exist. Type the command, **greetings**, and press *Enter*. You should receive a message saying that greetings could not be found:

```
greetings
ksh: greetings: not found
```

If this is not the case, then select another command such as **hi**, **hello**, or **salutations** and try each one out until you hit one that is not found.

To define a function, give the function a standard Unix-type name (in this case greetings) followed by the definition of the function. There are two different versions of the syntax for defining a function. In the example below, either command will define a function named greetings that prints 'Hello there' on the screen. Note that the body of the function is enclosed in curly braces:

```
$ function greetings { echo Hello there ; }

$ greetings () { echo Hello there ; }
```

Enter either command, and you will notice that nothing appears to happen. So far, you have defined the function, but you have not yet executed it. To execute the function, type its name just as you would a standard Unix command. The function will execute:

```
$ greetings
Hello there
$
```

Expand on this a bit and define the function to include displaying the environment variable containing the user's ID:

```
$ greetings () { echo Hello there $LOGNAME ; }
```

This time, when you type greetings, the message is more personalized:

```
$ greetings
Hello there mjb
$
```

A function may be defined to execute multiple commands, by separating the commands with a semi-colon or a newline.

The examples below both define a function named 'dowhat' that will list processes started by the current user. The **ps -ef** command is piped into **grep** to search for lines containing the user id (*$LOGNAME*) and the result is piped through **more**. At the end of the display 'Press

Enter' is displayed and a dummy variable is read from the keyboard until the user presses *Enter*. These examples use both types of syntax to define the function. The second version replaces each semi-colon with a newline. After each newline the continuation prompt (>) is displayed.

```
$ dowhat () {ps -ef|grep $LOGNAME|more ; echo Press Enter ; read x ; }
```

or:

```
$ function dowhat {
> ps -ef|grep $LOGNAME|more
> echo Press Enter
> read x
}
$
```

Now type **dowhat** and you will see a display of your activities:

```
$ dowhat
   mjb   260     1  7  Jul 31   Ø1      Ø:Ø4 -ksh
   mjb   261     1  Ø  Jul 31   Ø2      Ø:Ø1 -ksh
   mjb  2293     1  Ø  1Ø:31:37 Ø3      Ø:Ø1 -ksh
   mjb  2313   26Ø  7  1Ø:45:Ø6 Ø1      Ø:ØØ  ps -ef
   mjb  2314   26Ø  7  1Ø:45:Ø6 Ø1      Ø:ØØ  grep mjb
   mjb  2315   26Ø  7  1Ø:45:Ø6 Ø1      Ø:ØØ  more
Press Enter
$
```

While it is fine to use the command line to learn about functions, their true power comes about in shell scripts.

First let's cover some simple rules about shell scripts. Ensure that any shell script you create will run using */bin/sh* or */bin/ksh* by starting the shell script in one of two ways:

1 Always start with a blank line.

2 Always start with the line:

```
#! /bin/sh
```

or:

```
#! /bin/ksh
```

The first method will work on older versions of Unix. The second

method has been supported on many versions of Unix for several years now.

In the following examples I will use the second version.

So far I have shown you examples of functions used to create a sort of alias for one or more other commands. Within shell scripts, functions are used to tidy up the script so that it is more readable and to avoid having to repeat the same logic over and over again. It frequently makes the logic of a long shell script easier to follow.

In Listing 1 I have added line numbers to simplify the explanation. You can type this in as simpmenu using vi or any other editor. Change the mode to executable with:

```
chmod a+x simpmenu
```

At lines 4 through 20 a long function named *amenu()* is defined. This function clears the screen, and displays a set of prompts in a menu format. Lines 24 through 27 define a *PressEnter()* function that asks the user to press *Enter* and then waits for a key press.

Lines 29 through 44 define functions that are to be used or called in response to each menu pick. Note that each of these three functions uses a call to *PressEnter* within the body of the function. This ability to use a function inside another function makes shell programming much easier. Without this ability, the logic of the *PressEnter* function would have to be fully typed within the body of the three main functions that are executed by the menu.

Up to line 63, no actual program has been run. If the shell script stopped at this point, nothing effective would have happened. At line 63 the program begins with a permanent loop (*while true*) that runs to line 85. The menu is displayed by calling the function *amenu* at line 66. The read command is used to read the variable answer at line 69. At line 74, a case statement tests the value in *$answer* and executes one of the three defined functions based on the user's choice, or issues a break command that breaks out of the permanent loop.

Listing 1 – a simple demonstration of functions:

```
1    #! /bin/ksh
2    # simpmenu, A very simple menu
3
4    # a function to display a simple menu
5    amenu () {
6    # clear the screen and display the main menu
7    clear
8    echo 'date'
9    echo
10   echo "\t\t\tMy Personal Menu"
11   echo
12   echo "\t\tPlease Select:"
13   echo
14   echo "\t\t\t 1. Directory display"
15   echo "\t\t\t 2. Current Activity"
16   echo "\t\t\t 3. Who is logged on"
17   echo "\t\t\t"
18   echo "\t\t\t 0. Exit"
19   echo Select by pressing a number and then ENTER ;
20   }
21
22   # Ask the user to press enter
23   # and wait for the ENTER Key
24   PressEnter () {
25   echo Press Enter
26   read x
27   }
28
29   # Three functions for three menu picks
30   DirectoryDisplay () {
31   ls -l|more
32   PressEnter
33   }
34
35   CurrentActivity ()
36   {
37   ps -ef|more
38   PressEnter
39   }
40
41   WhoIsLoggedOn () {
42   who|more
43   PressEnter
44   }
45
46   # The main logic for the program displays the menu,
47   # gets the user's entry and initiates the activity
48   #-----------------------------------------------
49   # MAIN LOGIC
```

```
50  #-----------------------------------------------
51  # Every thing up to this point has
52  # just defined functions.
53  # The program main logic actually starts running here.
54
55  # Repeat the menu over and over
56  # by:
57  # 1. Display the menu
58  # 2. 'read' a line of input from the keyboard
59  # 3. Check the answer for 1, 2, 3, or 0 and dispatch
60  #    to the appropriate function or exit
61  # 4  Repeat until 0 is entered
62
63  while  true
64  do
65  # 1. display the menu
66    amenu
67
68  # 2. read a line of input from the keyboard
69    read answer
70
71  # 3. Execute one of the defined functions based on the
72  #    number entered by the user.
73
74    case $answer in
75        1) DirectoryDisplay ;;
76        2) CurrentActivity ;;
77        3) WhoIsLoggedOn ;;
78
79  #      If the user selects 0 to exit then break out
80  #      of this loop
81        0) break ;;
82    esac
83
84  #     Do it again until the user enters 0.
85  done
86
87  # Clear the screen on the way out
88  clear
```

Listing 2 is an example of an attempt to create a similar simple menu, but without using functions. It is much harder to read and contains a lot of duplicate code.

Listing 2 – a duplicate of Listing 1 without functions:

```
1   #! /bin/ksh
2   # simpmenu, A very simple menu
3
```

```
4
5    # The main logic for the program displays the menu
6    # gets the user's entry and initiates the activity
7    #-----------------------------------------------
8    # MAIN LOGIC
9    #-----------------------------------------------
10
11   # Repeat the menu over and over
12   # by:
13   # 1. Display the menu
14   # 2. 'read' a line of input from the keyboard
15   # 3. Check the answer for 1, 2, 3, or Ø and dispatch
16   #    to the appropriate function or exit
17   # 4  Repeat until Ø is entered
18
19   while  true
20   do
21   # 1. display the menu
22   # clear the screen
23   clear
24   echo 'date'
25   echo
26   echo "\t\t\t My personal menu"
27   echo
28   echo "\t\t Please Select:"
29   echo
30   echo "\t\t\t 1. Directory display"
31   echo "\t\t\t 2. Current Activity"
32   echo "\t\t\t 3. Who is logged on"
33   echo "\t\t\t"
34   echo "\t\t\t Ø. Exit"
35   echo Select by pressing a number and then ENTER
36
37   # 2. read a line of input from the keyboard
38     read answer
39
40   # 3. Execute commands based on the
41   #    number entered by the user.
42
43     case $answer in
44         1)
45               ls -l|more
46               echo Press Enter
47               read x ;;
48         2)
49               ps -ef|more
50               echo Press Enter
51               read x ;;
52         3)
```

```
53              who|more
54              echo Press Enter
55              read x ;;
56
57  #      If the user selects 0 to exit then break out
58  #      of this loop
59        0) break ;;
60    esac
61
62  #     Do it until the user enters 0.
63  done
64
65  # Clear the screen on the way out
66  clear
67
```

If you only used functions to tidy up long shell scripts you would be taking advantage of one of the powerful features of the shell programming language. The true beauty of functions is the ability to modularize logic with a script, and create a structured program.

Mo Budlong
President
King Computer Services (USA)

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.

Using EMC Symmetrix's TimeFinder option to create QA-system copies

We have a fairly large number of medium to large database systems (Oracle as well as Informix) in our company, most of them used for SAP systems. Every now and then our QA people, developers, and system designers want to create copies of whole application systems. These copies will then be presented to the same or a different host, which will then operate on the data, eg preparing or testing upgrade strategies, running QA tests, etc.

Being a long term EMC customer for storage systems in connection with RS/6000 AIX hosts, we selected the TimeFinder option for synchronizing and splitting off the mirrors of the production database.

EMC SYMMETRIX BASICS

An EMC Symmetrix (3000 or 8000 family, differences are in internal processing capabilites and layout) connected to an AIX system via SCSI or Fibre Channels is seen from the AIX viewpoint as a (huge) array of hdisks, whose size can be set during Symmetrix configuration time. The most common size is 8,492GB per hdisk. Internally the Symmetrix consists of Front End Directors (SCSI, Fibre), a Cache Memory (2 to 64GB), Disk Directors and HDAs (typically 18GB, 36GB, 72GB, or 181GB in size). The disk directors move the data between cache and HDAs. The microcode (currently 5x66 or 5x67) is responsible for the internal operation such as data moving or mirroring for RAID-1 disk groups. The microcode is also able to synchronize disk data to a remote Symmetrix using ESCON or Fibre Channel, thereby employing EMC's proprietary SRDF (Symmetrix Remote Data Facility) protocol and algorithm. The hdisks are parts of the HDAs and are called Hyper Volume Extensions (HVE).

The operation of a Symmetrix system can be controlled using a GUI or a set of command line programs, the symcli. The symcli communicate with the Symmetrix via special hdisks, which have a size of only 2,88MB.

To be able to use a Symmetrix system one has to install the relevant

components of the Symmetrix-lpp (which can be downloaded from the EMC Web site). Its primary purpose is to include Symmetrix definitions in the PdDv and PdAt preconfigured ODM classes, so that the driver attributes (such as time-out, etc) are correctly configured when running the **cfgmgr** configuration manager command against the Symmetrix. Additionally this **lpp** includes an executable to identify hdisks and their Symmetrix-internal representation (**inq**).

WHAT IS TIMEFINDER?

TimeFinder is an API-controlled interface to the mirroring code of the Symmetrix system. During configuration of the Symmetrix (by the hardware engineer) some HVEs are set aside for TimeFinder operations (provided the licence is purchased). EMC calls these volumes 'Business Continuance Volumes' (BCV). Using the correct symcli commands, these BCVs can be brought in sync with the source volumes and, at a convenient moment, split from the source volumes. A BCV can be mirrored in itself to increase availability (which we do). The source volumes are not affected (thus the term 'business continuance') and the split-off BVCs can then be presented to a host system such as an RS/6000. The RS/6000 will then run the **cfgmgr** command and thus identify the volumes, which are now exact copies of the source volumes.

We feel that one of the most exciting aspects of TimeFinder is that it can operate across Symmetrix boundaries, provided the Symmetrixes are SRDF-connected. Then it is possible to establish and split off copies of the source volumes miles away, with only a handful of command line commands (as we do). These symcli commands, together with the TimeFinder commands, have to be installed from separate EMC media.

WHAT ARE THE PECULIARITIES IN AIX TIMEFINDER OPERATION?

In our shop we implemented the following series of steps to end up with QA-copies of our medium-to-large production databases.

This is how an **lsdev** command will see the normal ('standard') devices which are all RAID-1 protected:

...

```
hdisk40 Available 50-58-00-2,0  EMC Symmetrix SCSI RDF1 Raid1
hdisk41 Available 50-58-00-2,1  EMC Symmetrix SCSI RDF1 Raid1
hdisk42 Available 50-58-00-3,0  EMC Symmetrix SCSI RDF1 Raid1
hdisk43 Available 50-58-00-3,1  EMC Symmetrix SCSI RDF1 Raid1
hdisk44 Available 50-58-00-4,0  EMC Symmetrix SCSI RDF1 Raid1
hdisk45 Available 50-58-00-4,1  EMC Symmetrix SCSI RDF1 Raid1
hdisk46 Available 50-58-00-5,0  EMC Symmetrix SCSI RDF1 Raid1
hdisk47 Available 50-58-00-5,1  EMC Symmetrix SCSI RDF1 Raid1
hdisk48 Available 50-58-00-6,0  EMC Symmetrix SCSI RDF1 Raid1
...
```

First of all we define the Symmetrix device groups; you can name the group anything you like, but we prefer names containing the source and target system name. Because it is fairly ease to create a map of the internal layout of the Symmetrix using the **inq** command, we like to operate on the internal device numbers, but there are other ways of identifying the disks also. We use the command:

- `symdg create group [-type RDF1]` (the type must be specified, if the BVCs are in a remote Symmetrix system).
- `symdg -g group add device devid1 , .....` (until all standard devices are registered with the group).
- `symbcv -g group associate [-rdf] device bcvid1 , ….`(until all BCV devices are registered with the group).

Here you would use **–rdf** if the BCVs are not contained in the same Symmetrix as the standard devices.

Now we doublecheck the groups using the commands **symdg –g group show**.

Now it is time to establish the synchronization process with the command:

```
symmir -g group establish [ -rdf] -full -opt
```

where:

- *-rdf* is necessary if the BCVs are not local to the standard devices.
- *-full* is necessary if the group is synchronized for the first time or if the BCVs or the Standard Devices have been connected to other partners in the past.

- *-opt* tells the microcode to create such pairs of standard and BCV devices that the mircocode operation of copying the data to the partner can take place in the most efficient manner (thereby exploiting the internal parallelism to its maximum).

With the command **symmir –g group [-rdf] –q** one can have a snoop at the state of the copying process. Once the process has completed (synchronizing 300GB of data across an SRDF link requires typically 2 ½ hours in our shop), the BCVs remain in the synchronized state, meaning that each and every write to the standard volumes is reflected on the BCVs (just as with an ordinary mirror).

When a convenient point in time is reached, the BCVs can be split off; this can take a considerable amount of time (some 60 seconds under heavy load and for an extended number of disks), a situation remedied by the newer microcode families. The command would be:

`symmir -g group [ -rdf] -split` (using **–rdf** again as necessary).

After the split has completed, writes to the standard volumes will no longer be reflected on the BCVs, thus leaving the BCVs as intact back-up copies. But there is more to it. The split also tells the microcode to present the BCVs as hdisks on the front-ends designated during configuration time.

Let us now assume that there is another RS/6000 connected to the Symmetrix on those ports where the BCVs are now presented. Running the configuration manager will identify all hdisks, but will leave them in the ‘defined’ state. This is to prevent inconsistent or unwanted results when a ‘BCV-system’ reboots during a split.

To make the defined volumes available, we use the command **mkbcv –a**, which sets all of the BCVs to available.

Thus a device listing on the host where the BCVs are presented would look like:

```
…
hdisk39 Available 40-60-00-4,0  EMC Symmetrix SCSI Disk TimeFinder
hdisk40 Available 40-60-00-4,1  EMC Symmetrix SCSI Disk TimeFinder
hdisk41 Available 40-60-00-5,0  EMC Symmetrix SCSI Disk TimeFinder
hdisk42 Available 40-60-00-5,1  EMC Symmetrix SCSI Disk TimeFinder
hdisk43 Available 40-60-00-6,0  EMC Symmetrix SCSI Disk TimeFinder
hdisk44 Available 40-60-00-6,1  EMC Symmetrix SCSI Disk TimeFinder
```

```
hdisk45 Available 40-60-00-8,0  EMC Symmetrix SCSI Disk TimeFinder
hdisk46 Available 40-60-00-8,1  EMC Symmetrix SCSI Disk TimeFinder
hdisk47 Available 40-60-00-9,0  EMC Symmetrix SCSI Disk TimeFinder
hdisk48 Available 40-60-00-9,1  EMC Symmetrix SCSI Disk TimeFinder
hdisk49 Available 40-60-00-10,0 EMC Symmetrix SCSI Disk TimeFinder
…
```

Then we can readily import the volumegroups on the hdisks with their original name or, as we prefer in our installation, with another volume group name. We also created some scripts to rename all the logical volumes and the filesystems to different names, just to always be able to differentiate the production system from its BCV counterparts.

The logical volumes and filesystems can now be used in the same way as any other LVM object. Just be sure to unmount any BCV-based filesystem, close the logical volumes (eg stopping a database system), and export the volume groups before starting another sync. This is necessary because resyncing the BCVs will abruptly terminate the accessibility of the volumes by the 'BCV-host'.

SPECIAL CASE: PRESENTING THE BCVS TO THE SAME HOST AS THE STANDARDS

There is a special case when the BCVs are presented to the same host as the standard devices. When the configuration manager is run, it detects the same disks as the standards (with identical PVIDs) once again; thus the LVM cannot import the volumegroup. With AIX 4.3.3 Maintainance Level 06 the command **recreatevg** was introduced. This command can be used to adapt the LVM meta structure (VGDA, VGSA, LVMCB, …) to represent unique disks. The **recreatevg** command in a very simple form would be:

```
recreatevg -y bcvvg -Y bcvlv hdisk5 hdisk6 hdisk7
```

This command would recreate the volume group with unique PVIDs, change the volume group name to *bcvvg*, rename all logical volumes on the volume group by prepending **bcvlv**, and then vary on the volume group.

There is a very informative *man* page available, which can be downloaded from the IBM Web site.

Felix Fibich
Informations-Technologie (Austria)

Using history and command line editing

One of the powerful and useful features of the C Shell and the Korn Shell is history. The ability to recall and reuse commands that you have entered earlier can save many keystrokes of repetitive typing.

Combine history with command line editing and you have a handy pairing of two closely-related features. You may use command line editing to edit the command that you are currently working on after you have typed it in, but you may also edit a command line that has been recalled from command history.

I hope to show you some of the ins and outs of history and command line editing in the Korn Shell.

For history to work, you need to be in the Korn shell, so type:

```
ksh
```

to launch a Korn Shell if you are not already in one. Korn Shell is the default for AIX machines, so you may already be using ksh unless your login script is set up for a different shell.

The Korn Shell allows two major different styles for retrieving command history and editing commands. The primary editors are vi, emacs, and gmacs. In this article vi is used as the command line editor. History and command line editing behave very differently depending on whether you are using vi or emacs/gmacs, so I will also cover some of the differences for emacs/gmacs.

To set vi up as your history editor type:

```
set -o vi
```

Once you have used history, you will want to use it again, and you will need to include this command in your start-up profile.

Here are the equivalent Korn shell **set -o** commands for emacs and gmacs:

```
set -o emacs
```

or:

```
set -o gmacs
```

Assuming that you have **set -o vi** as the history editor, you may now access history. Press the ESCAPE (Esc) key once and release it and press **k**. The last command that you typed will appear on the screen. That last command was the one you just entered, **set –o vi**:

```
$ set -o vi
```

Once a historical command is retrieved, you may press *Enter* and that command will be executed. Try a few more commands such as:

```
ps -ef
who am i
ls -l
```

End with **ls -l** on the current directory. Now press ESCAPE and **k**. The **ls -l** command appears once more. Press **k** again and the **who am i** command appears. Press **k** again and you go back one more command to **ps -ef**. Each time you press **k** you move back one more command in your history. The minus key (-) will work exactly like **k**. I use minus because it is easier for me to remember. Eventually you press **k** or minus enough times and you end up back at the **set -o vi** command.

Now press **j** or **+** and you will move forward through the command history. If you are rapidly running back through commands looking for one you need, and you pass it, it is nice to be able to come forward again.

If you have **set -o emacs** or **gmacs**, the keys are CONTROL-P for the previous command and CONTROL-N for the next command. In vi you hit ESCAPE once and then **k** or **j** will move you backward or forward through the commands. For emacs/gmacs use CONTROL-P or CONTROL-N each time.

To view your history as a list of commands, type history and press *Enter*. A list of your most recent commands is displayed with numbers. If you see the number of the command that you want you may also recall a command by hitting the ESCAPE key, typing the number of the command as displayed by history, and then an upper

case **G**. This will place the command with that number on the command line.

Once you have located the command that you want, you can press *Enter* to execute the command, but you can also edit the command before executing it.

There are several vi commands for moving around the command line and editing it. I will cover only a few here. For further information consult the manual on ksh. If you are familiar with vi you will recognize the commands. The vi editor operates in two main modes. In command mode single keystrokes move the cursor, delete characters, and perform other actions. In input mode single keystrokes result in characters being placed on the screen. To enter command mode with the vi editor, press **Esc** once. Once you are in command mode, the following are some of the commands that can be executed:

- l (el) – move the cursor forward one character.
- h – move the cursor back one character.
- w – move the cursor forward one word.
- b – move the cursor back one word.
- fc – find character c in the line.
- 0 (zero) – cursor to the start of the line.
- $ – cursor to the end of the line.

l, **h**, **w**, **f**, and **b** can be preceded by a number. Thus **2b** moves the cursor back two words, and **3fx** finds the third occurrence of *x* in the line. In emacs/gmacs mode cursor movement is different, ie:

- CONTROL-F – move the cursor forward one character.
- CONTROL-B – move the cursor back one character.
- ESCAPE then f – move the cursor forward one word.
- ESCAPE then b – move the cursor back one word.
- CONTROL-A – cursor to the start of the line.

- CONTROL-E – cursor to the end of the line.

The editing of a command line is not limited to history. You can also edit a command line that you are currently typing in. For example, type the following, but do not press *Enter*:

```
ls -l
```

You should be sitting with the cursor at the end of the command line.

If you have **set -o vi** then hit the ESCAPE key to exit input mode and enter command mode. Now you can type **h** a few times and the cursor will move backward through the command line. If you have **set -o emacs**, there is no need to press ESCAPE, just start by typing CONTROL-b a few times, and the cursor will move backwards.

So now you have the cursor where you want it, but how do you actually edit the command? There are several editing options, and again I am going to describe just some of the most useful for vi:

- x – delete the character over the cursor. Can be preceded by a count.
- X – delete the character behind the cursor. Can be preceded by a count.
- ~ (tilde) – change case of the character over the cursor. Can be preceded by a count.
- u – undo the last change to the line.
- U – undo all changes to the line.
- a – enter input mode and begin inserting after the current character.
- A – enter input mode and begin inserting at the end of the line.
- i – enter input mode and begin inserting before the current character.
- I – enter input mode and begin inserting at the beginning of the line.

Once you are in input mode, which is started by pressing **a**, **A**, **i**, or **I**, you may exit input mode by pressing the ESCAPE key and return to command mode.

You may press *Enter* at any time, and the command as it appears on the command line will be executed.

In emacs/gmacs mode you are always in input mode and normal characters that are typed appear in the command line. Editing commands are given as CONTROL + some key or as an ESCAPE followed by some key:

- CONTROL-D – delete the character over the cursor.
- ESCAPE then d – delete the current word.
- CONTROL-C~ – capitalize current character.
- ESCAPE then l – convert current character to lower case.

In emacs/gmacs mode you may also press *Enter* at any time, and the command as it appears on the command line will be executed.

Remember that command line editing applies to the current command that you are typing, or to any command that you have recovered from your history.

The last piece of command line editing that I want to cover is file name completion. This is usually used in current command lines but of course can be used when editing a command retrieved from history.

Assume a directory and file structure as follows and you are currently in the abc directory:

```
(dir)  topdir
         |
         -------------------------------
         |             |                 |
(dir)    abc           def               ghi
(files) data.txt      john.ltr        news.doc
        old.txt       john.txt        weather.doc
```

Type the following but do not press *Enter*:

```
ls -l d
```

Now hit ESCAPE and then the backslash. This causes the command line editor to attempt to locate a file (or directory) that can be used to complete the name that has been started. The search finds data.txt and that is filled in on the command line:

```
ls -l data.txt
```

The cursor is left after the *txt* of *data.txt* ready for you to type anything else that is needed to complete your command line.

This also works for directories. Assuming again that you are in the abc directory and need to edit news.doc in the ghi directory, you could change to that directory and then start editing, or you could issue a vi command to edit directly from where you are:

```
cd ../ghi
vi news.doc
```

or:

```
vi ../ghi/news.doc
```

If you try the following sequence you will see the backslash at work completing the names for you. First type enough of the command to identify the directory:

```
vi ../g
```

Now type ESCAPE followed by backslash and the directory name is completed for you:

```
vi ../ghi/
```

Now type enough to uniquely identify the file by adding an ‘n’:

```
vi ../ghi/n
```

Press ESCAPE followed by backslash and the file name is completed for you:

```
vi ../ghi/news.doc
```

This feature is especially useful for long directory and file names and saves a lot of typing.

If two or more files match what you have typed so far, the file completion option will complete as much of the file as it can and then wait for you to provide more unique input. To edit the file john.txt from one of the adjacent directories type:

```
vi ../def/j
```

Pressing ESCAPE and backslash will fill in as much as it can and then wait for you to indicate whether it is john.txt or john.ltr that you want:

```
vi ../def/john.
```

The sequence that does this for emacs/gmacs is ESCAPE followed by a second ESCAPE.

There is much more to command line editing and history, but I have found that just these few tips greatly speed up my work, and I hope they do the same for you.

If you like the idea of history, ask your system administrator to change your shell to a Korn Shell, and then be sure to add **set -o vi**, **emacs**, or **gmacs** into your *.profile* file.

Mo Budlong
President
King Computer Services (USA)

Create a tnsnames.ora for your PC using the AIX files

INTRODUCTION

Our company has several AIX and NT sites. On some of these servers are one or more Oracle databases. To make it complicated, some of these databases also have database links to databases on other servers. All these databases are mentioned in the tnsnames.ora on that host. Only the databases residing on the host can be found in the oratab of the host. To extract one tnsnames.ora from all AIX sites and all TNS_ADMIN directories to create a tnsnames.ora for your PC is not a simple job.

It would be easy to have just one or two static databases on one or two machines. But other people, such as developers, testers, and application managers, would like to have lots of databases on lots of machines. All development and test databases have a short life, for example two months, but some are years 'in production'.

Some testers are complaining that they do not have the right tnsnames.ora on their PC – some databases are missing. Or they are now using Oracle8 instead of Oracle 7, and so they cannot use their own tnsnames.ora.

The administration of this is a difficult job – you can easily forget one or more entries in different tnsnames.oras in different TNS_ADMIN directories. You can also forget to delete or add entries in the oratabs.

Having to:

- Gather all the entries.
- Sort them.
- Multiply them by adding *.world*.
- Make duplicate entries unique.
- Check whether all the oratabs are clean.

is not a job I like to do. So I made a script to ease my job.

I have set it in my crontab and it runs as standard every week. If people are complaining I will run it at that time and mail the actual tnsnames.ora. If there are errors I will resolve them at that time, so I have always a clean tnsnames.ora and oratab on each machine.

There is just one requisite: the TNS entry name has to be on the same line as the word DESCRIPTION, so I know that this line is the beginning of the entry.

Also note that if one or more hosts are not online, the script continues with warnings. In that case you must have the actual tnsnames.ora of that machine or machines. You need that one to add the lines of the tnsnames on the machines: only the entries called in other tnsnames.oras will be added to the file tnsnames.ora.

For the first few times it is recommended to use the debug parameter. You will get much more information about why some databases are and other databases are not mentioned in the tnsnames.ora. The first time, it will probably generate lots of errors (in my experience!). Edit the files and restart the script. After restarting a couple of times the errors vanish and you will get a usable tnsnames.ora.

NEXT STEPS

To make it easier for NT administrators, you can put the tnsnames.pc in a certain directory on an NT server. If it is shared, everyone can get the tnsnames.ora to update the old one. Another step you can make is to distribute it across the servers and put the file on all PCs using the net logon procedure.

THE SCRIPT

```
# Name           : gather_tnsnames
# Last change    : 09-04-2001 T.W. Post - creation script
# Description    : grab all tnsnames.ora + oratabs of all aix servers
#                  check if all entries in the tnsnames.ora are valid
#                  and all entries in the oratab are mentioned in the
#                  tnsnames.ora
#-----------------------------------------------------------------
# =================================================================
```

```
#
# company /site dependent variables
#
# company_name used in the host parameter e.g.
#              HOST=aixserver.domain.company
# This variable contains only the last part
company_name=Your_company_name_used_for_tcpip

# ftp_user: user used for ftp and starting this script
ftp_user=ftp_user

# work_dir : Directory where we can find the files we need and
#            we can put the result
#      needed: also_tnsnames.add
#      needed: diff_tnsadmin_sites
#      result: tnsnames.pc
work_dir="/home/workdirectory"

# password_dir      : directory used for password files
# password_filename : file name containing hosts + their
#                     corresponding passwords of hosts that do
#                     not have the general password of the ftp_user
password_dir="/home/secret"
password_filename="pwd_ftpuser"

# general_dir : directory used for general files :
#               all_aix_servers  - containing all aix servers
#               all_test_servers - containing all aix test servers
general_dir="/home/general"

# program_sid: program used as an SID, always last chars of an SID"
# len_prog   : length in characters of the program SID
program_sid="FNDFS"
len_prog=5

# There are two possibilities:
#    - get all entries of the tnsnames.ora set get_all_entries=Y
#    - get only the entries of all started databases set
#          get_all_entries=N
get_all_entries="Y"

# The next variable is created to continue even if there are serious
#  errors.
# If you want to always get a tnsnames.ora: set it to Y
# Initializing it with N means: stop if errors are found.
# This variable is meant to be used in batch mode, starting this script
# on the prompt will ignore this variable
ignore_all_errors="N"

Edit_duplicate_servernames ()
```

```
{
# In the file tnsnames.test exist an entry called database_aixserver4
# in its name the servername is duplicated so you get
# database_aixserver4_aixserver4.
# There is no entry database, so "repair"
# this entry to a conventional name
#
echo "Removing duplicate servernames"
cat my_tnsnames.test | sed '1,$s/_aixserver4_aixserver4//
'>tnsnames.edited
mv tnsnames.edited my_tnsnames.test

# also for the second test-server
cat my_tnsnames.test | sed '1,$s/_aixserver6_aixserver6//
'>tnsnames.edited
mv tnsnames.edited my_tnsnames.test

# end Edit_duplicate_servernames
}

# End of company / site dependent variables
#
# ================================================================

# uppercase variables
typeset -u debug
typeset -u noftp
typeset -u to_start
typeset -u YesNo
typeset -u line_parm1
typeset -u line_parm2
typeset -u help

# lowercase variables
typeset -l lower

Create_Tnslines ()
{
# Procedure to make the lines we were searching for
# It splits the line which contains the tnsnames entry for one database
# The first word: entry name will always be in lower case
# the rest of the entry is copied from the entry in the tnsnames.ora

# We make at most 4 entries of one database:
# tnsline  : tns entry started with databasename - oracle version 7
# tnsline2 : tns entry started with databasename and
#            serverprefix eg db1_server1 - Oracle Version 7
#            this line is only written for databases on testservers
# tnsline3 : tns entry started with databasename and .world
#            eg db1.world - Oracle Version 8
```

```
# tnsline4 : tns entry started with databasename and
#            serverprefix and .world  - Oracle Version 8
#            this line is only written for databases on testservers

# check if description is started in oratab
# if it is started: check if it is an local entry
# local entry: write to my_tnsnames.test
#                         ( test server ) or
#                       my_tnsnames.prod
#                         ( production server )
to_start="N"

if [ -s "oratab.$save_host" ]
then oratab_file=oratab.$save_host
else oratab_file=oratab.$orig_host
     fi

entry_oratab='fgrep -i ${save_sid}: $oratab_file 2>/dev/null | grep -v
"^#" '
to_start='echo $entry_oratab | cut -f3 -d":"'

if [ "$debug" = "DEBUG" ]
then if [ "$to_start" = "" ]
     then echo " $save_desc is not mentioned in the oratab $oratab_file
"
     else echo " $save_desc entry in oratab: \c"
          echo "$entry_oratab, started: $to_start"
          fi
     fi

if [ "$get_all_entries" = "Y" -a "$to_start" = "N" ]
then to_start="Y"
     if [ "$debug" = "DEBUG" ]
     then echo "$save_desc added due to variable get_all_entries"
           fi
     fi

if [ "$to_start" = "Y" ]
then test_server='fgrep $save_host $general_dir/all_test_servers'
     echo "     Adding database $save_desc"
     serverprefix='echo $save_host | cut -f1 -d'.''
     # Set description to lower case
     lower=$save_desc
     rest='echo $line | cut -f2- -d"="'
     tnsline=${lower}" = "${rest}
     tnsline2='echo ${lower}'_$serverprefix" = "${rest}
     tnsline3='echo ${lower}'".world = "${rest}
     tnsline4='echo ${lower}'_$serverprefix".world = "${rest}

     if [ "$test_server" ]
```

```
     then echo $tnsline  >>my_tnsnames.test
          echo $tnsline2 >>my_tnsnames.test
          echo $tnsline3 >>my_tnsnames.test
          echo $tnsline4 >>my_tnsnames.test
          if [ "$debug" = "DEBUG" ]
          then echo "  Server $servername marked as a testserver "
               fi
     else echo $tnsline  >>my_tnsnames.prod
          echo $tnsline3 >>my_tnsnames.prod
          fi
else if [ "$debug" = "DEBUG" ]
     then echo "  Entry $desc not started on \c"
          echo "$servername, entry ignored "
          fi
     fi

# End Create_Tnslines
}

Get_Parameters ()
{
save_desc='echo $line | cut -f1 -d "="'
save_sid=
save_host=
save_prot=
save_line=
len_prog=5
program_sid="FNDFS"
not_online=Ø

# Get parameter: PROTOCOL
prot=$line
while true
     do
     prot='echo $prot | cut -f2- -d"("'
     # remove leading and trailing blanks
     help='echo $prot | cut -c1-8'
     if [ "$help" = "PROTOCOL" ]
     then break
          fi
     done
prot='echo $prot | cut -f2 -d"=" | cut -f1 -d")"'
help='echo $prot | cut -c1-3'

# check only TCP protocol!
if [ "$help" != "TCP" ]
then save_prot="Protocol=$prot"
     if [ $processing -ne 1 ]
     then echo "     $save_desc \tnot checked: $save_prot"
          fi
```

```
else
     # Get parameter : SID
     sid=$line
     while true
           do
           sid='echo $sid | cut -f2- -d"("'
           # get the parameter name
           help='echo $sid | cut -f1 -d"="'
           # remove heading and trailing blanks
           help='echo $help'
           if [ "$help" = "SID" -o "$help" = "SERVICE_NAME" ]
           then break
                fi
           done

     save_sid='echo $sid | cut -f2 -d"=" | cut -f1 -d")"'

     # Get parameter : HOST
     host='echo $line'
     while true
           do
           host='echo $host | cut -f2- -d"("'
           # get the parameter name
           help='echo $host | cut -f1 -d"="'
           # remove heading and trailing blanks
           help='echo $help'
           if [ "$help" = "HOST" ]
           then break
                fi
           done

     save_host='echo $host | cut -f2 -d"=" | cut -f1 -d")"'
     # remove leading and trailing blanks
     save_host='echo $save_host'

     orig_host=$save_host
     # resolve tcp/ip address into hostname
     # The form can be:
     #   - hostX.domain.company
     #   - hostX.domain.company.country e.g. host.dom.company.en
     #   - tcp/ip address               e.g. 123.012.002.234
     # All of these addresses have to be translated to the
     # filename we get in the FTP step.
     teller='echo $save_host | awk -F. '{ print NF}''

     if [ $teller -eq 4 ]
     then int='echo $save_host | cut -f1 -d"." '
          let integer=int 2>/dev/null
          if [ $? -ne 0 ]
          then # full quallified name
```

```
               save_host='echo $save_host | cut -f1-3 -d'.''
          else # resolve TCP/IP address
               save_host='nslookup -retry=1 -timeout=1 \
                         $save_host 2>/dev/null | \
                         grep "Name:" | cut -f2 -d":" | \
                         tr 'A-Z' 'a-z'`
               if [ -z "$save_host" ]
               then # tcp/ip address not resolved :
                    # use original name to make it not to
                    # difficult to find it in the file
                    save_host=$orig_host
               else save_host='echo $save_host'
                    fi
               fi
     else if [ $teller -eq 1 ]
          then # resolve TCP/IP address
               save_host='nslookup -retry=1 -timeout=1 \
                         $save_host 2>/dev/null | \
                         grep "Name:" | cut -f2 -d":" | \
                         tr 'A-Z' 'a-z'`
               if [ -z "$save_host" ]
               then # tcp/ip address not resolved :
                    # use original name to make it not to
                    # difficult to find it in the file
                    save_host=$orig_host
               else save_host='echo $save_host'
                    fi
               fi
          fi
     fi
# End  Get_Parameters
}

Check_Parameters ()
{
if [ "$save_host" -a "$save_sid" ]
then > errors.host
     # check if we have the file
     if [ -s oratab.$save_host ]
     then cat oratab.$save_host 2>errors.host| \
              grep -i $save_sid 1>/dev/null
          retc=$?
     else # save_host is not the complete name :
          #   it is maybe extended by country name,
          #   use variable orig_host
          cat oratab.$orig_host 2>errors.host| \
              grep -i $save_sid 1>/dev/null
          retc=$?
          if [ $retc -eq Ø ]
          then >errors.host
```

```
          fi
     fi

if [ $retc -ne 0 ]
then # Host is maybe not online:
     #      look in not_responding_sites
     fgrep $save_host not_responding_sites 1>/dev/null
     if [ $? -eq 0 ]
     then # Host at last found
          retc=0
          >errors.host
          echo "     $save_host is NOT online: \c"
          echo "$save_desc not checked"
          not_online=1
          fi
     fi

# Check if SID is a program ( programs will not be checked ! )
len='echo $save_sid |wc -c'
len='expr $len - 1'
if [ $len -ge $len_prog ]
then help=$save_sid
     while [ $len -gt $len_prog ]
           do
           help='echo $help | cut -c2-'
           let len=$len-1
           done
     if [ "$help" = "$program_sid" ]
     then retc=0
          fi
     fi
if [ -s errors.host ]
then # if we could not find the hostfile, we
     # generate an error, check if it can be
     # resolved during the extra file
     fgrep $save_host also_tnsnames.add 2>/dev/null | \
           fgrep -i $save_sid >/dev/null
     if [ $? -eq 0 ]
     then # host found in extra file
          retc=0
          >errors.host
          fi
      fi

if [  $retc -ne 0 -o -s errors.host ]
then # We did not find the sid on the host -> ERROR
     echo "Host: $servername" >>errors.tnsnames
     if [ -s errors.host ]
     then echo "   Unknown hostname - $orig_host "\
               >>errors.tnsnames
```

```
               fi
          echo "   $save_sid does not exist on host $orig_host" \
                    >>errors.tnsnames
                    echo "   entry in tnsnames.ora on host $serverprefix
marked as an invalid entry \n">>errors.tnsnames
          error=1
          fi
     if [ $not_online -eq Ø ]
     then echo "     $save_desc \tchecked"
          fi
     fi

# End Check_Parameters
}

# go to the working directory
cd $work_dir

#
# Check whether any parameters are given, and fill the
# corresponding variables
line_parm1=$1
line_parm2=$2
if [ $# -gt 1 ]
then if [ "$line_parm1" = "DEBUG" ]
     then debug=$line_parm1
          noftp=$line_parm2
     else debug=$line_parm2
          noftp=$line_parm1
          fi
else if [ "$line_parm1" = "DEBUG" ]
     then debug=$line_parm1
     else noftp=$line_parm1
          fi
     fi

#
# Check if the program is started online or in batch:
#       in batch we do not have a tty
tty >/dev/null 2>&1
if [ $? -ne Ø ]
then batch="BATCH"
else batch="FOREGROUND"
     clear
     fi

batch_errors="No"

if [ "$debug" = "DEBUG" ]
then echo "\n\n############################################################ "
```

```
     echo "#                                                          # "
     echo "#         Script entering DEBUG mode ...                   # "
     echo "#                                                          # "
     echo "############################################################ \n\n"
     fi

if [ -s tnsnames.pc ]
then echo "\n\n############################################################ "
     echo "#                                                          # "
     echo "#    Previous version renamed to old.tnsnames.pc           # "
     echo "# This one will be used if sites are not responding        # "
     echo "#                                                          # "
     echo "############################################################ \n\n"
     mv tnsnames.pc old.tnsnames.pc
     fi

if [ "$noftp" = "NOFTP" ]
then echo "\n\n############################################################ "
     echo "#                                                          # "
     echo "#                No FTP will be done                       # "
     echo "#                                                          # "
     echo "############################################################ \n\n"
else # get general password of user to ftp
     ftp_password='cat $password_dir/$password_filename 2>/dev/null'

     if [ "$ftp_password" ]
     then echo "Password of ftp user was found, continuing ...\n"
     else if [ "$batch" = "BATCH" ]
          then echo
"\n\n############################################################ "
               echo "#                                                          # "
               echo "#      Password of ftp user was not found.                 # "
               echo "#                Script exiting                            # "
               echo "#                                                          # "
               echo "############################################################
\n\n"
               exit 1
               fi
          # Ask for the password of the ftp-user
          echo "Password $ftp_user: \c"
          OLDCONFIG='stty -g'
          # enter it without echo
          stty -echo
          read ftp_password
          stty $OLDCONFIG
          echo
          fi

     #
     # retrieve all tnsnames.ora and oratab from all Aix servers
```

```
#

# removing old entries just to be sure there is no old stuff
rm oratab.*.$company_name*   2>/dev/null
rm tnsnames.*.$company_name* 2>/dev/null
rm ping_*.*.$company_name*   2>/dev/null
rm ftp_*.*.$company_name*    2>/dev/null
>not_responding_sites

echo "\n\n############################################################ "
echo "#                                                          # "
echo "#       Retrieving tnsnames.ora and oratabs                # "
echo "#                                                          # "
echo "############################################################ \n\n"
for site in 'cat $general_dir/all_aix_servers'
   do
   echo "Retrieving data from site: $site"

   # if a site has not the standard password, use the password
   # mentioned in this file
   if [ -s $password_dir/$password_filename.$site ]
   then password='cat $password_dir/$password_filename.$site'
   else password=$ftp_password
        fi

   # build the .netrc file to get tnsnames.ora and the oratab
   echo "machine $site login $ftp_user password $password" \
         >/tmp/all_ora.netrc
   echo "macdef init"                          >>/tmp/all_ora.netrc
   echo "get /etc/oratab       oratab.$site"   >>/tmp/all_ora.netrc
   echo "get /etc/tnsnames.ora tnsnames.$site" >>/tmp/all_ora.netrc
   echo "quit"                                 >>/tmp/all_ora.netrc
   echo ""                                     >>/tmp/all_ora.netrc
   cp /tmp/all_ora.netrc    $HOME/.netrc
   chmod 600                $HOME/.netrc
   # .netrc file is now ok, we are ready to ftp

   # check if host is online otherwise the FTP is not responding
   ping -c3 $site 1>ping_$site.result 2>&1
   if [ $? -eq 0 ]
   then # site has been reached : ftp can be started
        if [ "$debug" = "DEBUG" ]
        then ftp $site 1>ftp_$site.output 2>&1
        else ftp $site 1>/dev/null 2>&1
             # drop debug file
             rm ping_$site.result
             fi
   else echo $site >>not_responding_sites
      echo "$site did not respond, script continuing with warnings"
        fi
```

```
        done

     # some sites have another $TNS_ADMIN
     # we also need this tnsnames.ora
     save_admin_site="Skip me"
     cat diff_tnsadmin_sites | while read site tns_admin
         do
         if [ "$save_admin_site" != "$site" ]
         then if  [ "$save_admin_site" != "Skip me" ]
              then echo "quit"      >>/tmp/all_ora.netrc
                   echo ""          >>/tmp/all_ora.netrc
                   cp /tmp/all_ora.netrc    $HOME/.netrc
                   chmod 600                $HOME/.netrc

                   # check if host is online otherwise the FTP
                   # is not responding
                   ping -c3 $save_admin_site
1>ping_$save_admin_site.result 2>&1
                   if [ $? -eq 0 ]
                   then # site has been reached : ftp can be started
                        if [ "$debug" = "DEBUG" ]
                       then ftp $save_admin_site 1>ftp_$site.output 2>&1
                        else ftp $save_admin_site 1>/dev/null 2>&1
                             # drop debug file
                             rm ping_$save_admin_site.result
                             fi
                   # else: Message has already been sent,
                   #       do not generate a second one
                        fi

                   # remove the netrc file
                   rm $HOME/.netrc  2>/dev/null

                   # merge the extra tnsnames with the information
                   # we already have and drop the temporary files
                   i=1
                   while [ $i -le $counter ]
                   do   cat $save_admin_site.$i
>>tnsnames.$save_admin_site
                        if [ "$debug" != "DEBUG" ]
                        then rm  $save_admin_site.$i
                             fi
                        let i=$i+1
                        done
                   fi
              counter=1
              save_admin_site=$site
              echo "Extra data will be retrieved from site: $site"
              if [ -s $password_dir/$password_filename.$site ]
              then password='cat $password_dir/$password_filename.$site'
```

```
          else password=$ftp_password
               fi

          # building .netrc file
          echo "machine $site login root password $password" \
               >/tmp/all_ora.netrc
          echo "macdef init"            >>/tmp/all_ora.netrc
          echo "get $tns_admin/tnsnames.ora $site.$counter"  \
              >>/tmp/all_ora.netrc
     else let counter=$counter+1
          echo "get $tns_admin/tnsnames.ora $site.$counter"  \
              >>/tmp/all_ora.netrc
          fi
     done
echo "quit"   >>/tmp/all_ora.netrc
echo ""       >>/tmp/all_ora.netrc
cp /tmp/all_ora.netrc    $HOME/.netrc
chmod 600                $HOME/.netrc

# check if host is online otherwise the FTP is not responding
ping -c3 $save_admin_site 1>ping_$save_admin_site.result 2>&1
if [ $? -eq 0 ]
then # site has been reached : ftp can be started
     # get the data we need
     if [ "$debug" = "DEBUG" ]
     then ftp $save_admin_site 1>ftp_$save_admin_site.output 2>&1
     else ftp $save_admin_site 1>/dev/null 2>&1
          # drop debug file
          rm ping_$save_admin_site.result
          fi
else echo $save_admin_site >>not_responding_sites
     echo "$save_admin_site did not respond, \c"
     echo "script continuing with warnings"
     fi

# remove the netrc file + the help file
rm $HOME/.netrc               2>/dev/null
rm /tmp/all_ora.netrc
# merge the extra tnsnames with the information we already have
i=1
while [ $i -le $counter ]
do   cat $save_admin_site.$i >>tnsnames.$save_admin_site
     if [ "$debug" != "DEBUG" ]
     then # we only destroy data if we are not debugging!
          rm  $save_admin_site.$i
          fi
     let i=$i+1
     done
fi
```

```
# At this point we have all the files we need.
# Now check, check and check again if all entries in the files
# are valid

#
# Check if all the entries - mentioned in the oratab - are present
# in the tnsnames.ora
#

echo
"\n\n############################################################################ "
echo "#                                                                          # "
echo "# Checking if all oratab entries are mentioned in                          # "
echo "# the tnsnames.ora                                                         # "
echo "#                                                                          # "
echo
"############################################################################
\n\n"

error=Ø
for file in 'ls oratab*'
    do
    servername='echo $file | cut -f2- -d"."'
    echo "Checking tnsnames <-> oratab of $servername"
    cat $file | grep -v "^#" | while read line
         do
         # Is this database started or just an entry which is
         # always skipped. If it is skipped, skip again
         to_start='echo $line | cut -f3 -d":"'
         if [ "$to_start" = "Y" ]
         then # database is started:
              # check if it is in the tnsnames.ora
              entry='echo $line | cut -f1 -d":"'
              check_it='fgrep $entry tnsnames.$servername'
              if [ ! "$check_it" ]
              then echo "-> Entry $entry at the site $servername \c"
                   echo "is not found in tnsnames.ora"
                   error=1
                   fi
              fi
         done
    done

if [ $error -eq 1 ]
then
     echo "\n\n############################################################ "
     echo "#                                                            # "
     echo "# Error(s) found in tnsnames.ora: please repair it           # "
     if [ "$batch" = "BATCH" ]
     then echo "#     script continuing ( only ) in batch mode      # "
```

```
          echo "#          to detect other errors ...               # "
          batch_errors="Yes"
     else echo "#                  exiting script ....              # "
          echo "#                                                   # "
          echo "#############################################################
\n\n"
          exit 1
          fi
     echo "#                                                        # "
     echo "############################################################ \n\n"
     fi

echo "\n\nAll entries of started databases in tnsnames.ora found, \c"
echo "continuing script .....\n\n"

#
# Check if all the SIDs mentioned in the tnsnames.ora can be found
# in the oratab.
# This step will consume the most of your time to execute
#
echo "\n\n############################################################ "
echo "#                                                          # "
echo "#    Checking tnsnames.ora of all TCP servers     # "
echo "#                                                          # "
echo "############################################################ \n\n"

typeset -u prot
error=Ø
>errors.tnsnames
>errors.host
typeset -u parm
typeset -i integer
typeset -u prot
processing=Ø

for file in 'ls tnsnames.*.$company_name*'
     do
     servername='echo $file | cut -f2- -d"."'
     serverprefix='echo $servername | cut -f1 -d'.''
     echo "Checking tnsnames.ora from $servername"
     cat $file | grep -v "^#" | while read me
          do
          while true
                do
                desc='echo $me | grep DESCRIPTION'
                if [ -z "$desc" ]
                then line="$line $me"
                     break
                else line='echo $line | tr -d " "'
                     if [ "$line" ]
```

```
                    then # We do have the whole string: now process it
                         Get_Parameters
                         Check_Parameters
                         fi
                    line="$me"
                    fi
               read me
               done
          done
     #  Do not forget the last description!
     line='echo $line | tr -d " "'
     Get_Parameters
     Check_Parameters
     line="$me"
     done
#
# Check if there are errors in the tnsnames.ora
if [ -s errors.tnsnames ]
then if [ "$batch" = "BATCH" ]
     then batch_errors="Yes"
     else clear
          pg errors.tnsnames
          fi
     echo
"\n\n################################################################## "
     echo "#                                                              # "
     echo "# xRef Error(s) found in tnsnames.ora: please repair it  # "
     if [ "$batch" = "FOREGROUND" ]
     then echo "#                  exiting script ....               # "
          fi
     echo "#                                                              # "
     echo "################################################################
\n\n"

     if [ "$batch" = "FOREGROUND" ]
     then exit 1
     else cat errors.tnsnames
          fi

else rm errors.tnsnames
     fi

forced_continue="N"

if [ "$batch_errors" = "Yes" ]
then echo
"\n\n################################################################## "
     echo "#                                                               # "

     if [ "$ignore_all_errors" = "Y" ]
```

```
     then echo "#  tnsnames.ora file(s) containing errors            # "
          echo "#  script continuing only due to variable           # "
          echo "#  ignore_all_errors                                # "
          echo "#                                                   # "
          echo
"#####################################################################\n\n"
          forced_continue="Y"
     else echo "#  script exiting due to errors in                  # "
     else echo "#   tnsnames.ora file(s)                            # "
          echo "#                                                   # "
          echo
"#####################################################################\n\n"
          exit  1
          fi
else rm errors.host 2>/dev/null
     fi

# At this point we've checked all entries :
#          oratab -> tnsnames.ora and
#          tnsnames.ora -> oratab
# There are no errors found, so extract the tnsnames.ora files
# to get all databases on that server get rid of duplicates

# initialize workfiles to be sure we do not have duplicate entries
>my_tnsnames.test
>my_tnsnames.prod
>tnsnames.lines
>tnsnames.pc
for file in 'ls tns*$company_name*'
     do
     processing=1
     # init vars
     servername='echo $file | cut -f2- -d"."'
     echo "Extracting tnsnames from $servername"
     cat $file | grep -v "^#" | while read me
          do
          while true
               do
               desc='echo $me | grep DESCRIPTION'
               if [ -z "$desc" ]
               then line="$line $me"
                    break
               else line='echo $line | tr -d " "'
                    if [ "$line" ]
                    then # We do have the whole string: now process it
                         Get_Parameters
                         Create_Tnslines
                         fi
                    line="$me"
                    fi
```

```
                read me
                done
          done
     #  Do not forget the last description!
     line='echo $line | tr -d " "'
     Get_Parameters
     Create_Tnslines
     line="$me"
     done

# Get rid of duplicate servernames in the tnsnames.ora
Edit_duplicate_servernames

# if there is an old version ( the previous one ) and not all
# sites did respond: extract lines from an old version of
# tnsnames.pc and add them to the "production file" because
# the entries called in tnsnames.pc are the right - and leading - ones

if [ -s not_responding_sites -a -s old.tnsnames.pc ]
then echo "\n\n################################################## "
     echo "#                                                # "
     echo "#  Adding tns entries from old version           # "
     echo "#                                                # "
     echo "################################################## \n\n"

     cat not_responding_sites | while read site
          do
          echo extracting tns entries of $site from old version
          fgrep $site old.tnsnames.pc >>my_tnsnames.prod
          done
     fi
#
# At this point we have all tnsnames.
# Merge the descriptions of the testservers with the descriptions
# of the production servers. Do not add duplicate entries!
#
echo "\n\n############################################################ "
echo "#                                                          # "
echo "#      Merging production and test tnsnames.ora            # "
echo "#                                                          # "
echo "############################################################ \n\n"

cat my_tnsnames.test | while read line
     do
     prev_desc='echo $line | cut -f1 -d"="'
     double='fgrep "$prev_desc="  my_tnsnames.prod'
     if [ $? -eq Ø ]
     then
          if [ "$debug" = "DEBUG" ]
          then echo "$prev_desc not added to tnsnames.pc : \c"
```

```
               echo "duplicate entry"
               echo "entry                 : $line "
               echo "entry previously added: $double"
               fi
     else echo "$line \n " >>my_tnsnames.prod
          fi
     done

#
# We have also databases on NT servers, merge them too
#
cat also_tnsnames.add >>my_tnsnames.prod

echo "\n\n############################################################ "
echo "#                                                          # "
echo "#   Building the tnsnames.ora for the PC platform          # "
echo "#                                                          # "
echo "############################################################ \n\n"

echo "########################################################## "  >tnsnames.pc
echo "#                                                        # " >>tnsnames.pc
echo "# date     : `date`        # "                             >>tnsnames.pc
echo "# filename : tnsnames.ora                                # " >>tnsnames.pc
echo "# file generated by gather_tnsnames                      # " >>tnsnames.pc
if [ "$forced_continue" = "Y" ]
then echo "#                                                        # "
>>tnsnames.pc
     echo "# ERRORS FOUND in tnsnames.ora / oratab                  # "
>>tnsnames.pc
     echo "# file STRAINED generated, please repair errors  # "
>>tnsnames.pc
     echo "# in original files.                                     # "
>>tnsnames.pc
     fi
echo "#                                                        # " >>tnsnames.pc
echo "########################################################## " >>tnsnames.pc

# Get rid of duplicates -u and - if there are upper and lowercase parts:
# set all to upper before sorting
sort -u -f my_tnsnames.prod >tnsnames.lines

# Add a blank line to make the file more readeble
cat tnsnames.lines | while read line
     do
     echo "$line\n" >>tnsnames.pc
     done

# Generate a message when a site has not responded
if [ -s not_responding_sites ]
then echo
```

```
"\n\n############################################################################ "
     echo "#                                                                    # "
     echo "#   Some sites did not respond, tnsnames.pc may                      # "
     echo "#   be incomplete                                                    # "
     if [ -s old.tnsnames.pc ]
     then echo "#         Lines copied from previous version!                   # "
          fi
     echo "#                                                                    # "
     echo "# Please check your old version with this one on the                 # "
     echo "#  next sites:                                                       # "
     cat not_responding_sites | while read site
          do
          i='expr "$site" : ".*"'
          while [ $i -lt 5Ø ]
                do
                site=$site' '
                let i=$i+1
                done
          echo "# $site             # "
          done
     echo "#                                                                    # "
     echo
"############################################################################
\n\n"
     fi

# Do not remove files in debug mode or strained file generation
if [ "$debug" != "DEBUG" -a "$forced_continue" != "Y" ]
then rm oratab.*.$company_name*
     rm  my_tnsnames.*
     rm tnsnames.*.$company_name*
     rm tnsnames.lines
     rm not_responding_sites
     fi

echo "\n\n############################################################ "
echo "#                                                            # "
echo "# The tnsnames.ora for the PC is called tnsnames.pc          # "
echo "#                                                            # "
echo "############################################################ \n"

echo "\n\n############################################################## "
echo "#                                                              # "
echo "#   Building the tnsnames.ora for the AIX platform             # "
echo "#                                                              # "
echo "############################################################## \n\n"

if [ "$forced_continue" = "Y" ]
then echo
"\n\n############################################################################ "
```

```
     echo "#                                                              # "
     echo "#  tnsnames.ora file(s) / oratabs containing error(s)          # "
     echo "#                                                              # "
     echo "#  SCRIPT CONTINUED ONLY DUE TO VARIABLE                       # "
     echo "#     ignore_all_errors                                        # "
     echo "#                                                              # "
     echo "#  STRAINED GENERATION OF tnsnames.pc AND tnsnames.aix         # "
     echo "#                                                              # "
     echo "#        PLEASE REPAIR ERRORS, FILES COULD BE IN ERROR         # "
     echo "#                                                              # "
     echo "################################################################
\n\n"
     fi

echo "######################################################## "
>tnsnames.aix
echo "#                                                      # "
>>tnsnames.aix
echo "# date     : `date`        # "
>>tnsnames.aix
echo "# filename : tnsnames.ora                              # "
>>tnsnames.aix
echo "# file generated by gather_tnsnames                    # "
>>tnsnames.aix
if [ "$forced_continue" = "Y" ]
then echo "#                                                      # "
>>tnsnames.aix
     echo "# ERRORS FOUND in tnsnames.ora / oratab                # "
>>tnsnames.aix
     echo "# file STRAINED generated, please repair errors        # "
>>tnsnames.aix
     echo "# in original files.                                   # "
>>tnsnames.aix
     fi
echo "#                                                      # "
>>tnsnames.aix
echo "######################################################## "
>>tnsnames.aix

cat tnsnames.pc | cut -f1 -d "=" | grep -v -E ".world|_|#" | sort -u | \
while read sid
      do
      if [ "$sid" ]
      then fgrep "$sid = " tnsnames.pc >>tnsnames.aix
           fi
      done

# end of script
```

EXAMPLE 1: LOGGING

```
#############################################################
#                                                           #
#    Previous version renamed to old.tnsnames.pc    #
# This one will be used if sites are not responding #
#                                                           #
#############################################################


Password of the ftp user was found, continuing ...


#############################################################
#                                                           #
#       Retrieving tnsnames.ora and oratabs         #
#                                                           #
#############################################################


Retrieving data from site: aix1.dom1.schuitema
Retrieving data from site: aix2.dom1.schuitema
aix2.dom1.schuitema did not respond, script continuing with warnings
Retrieving data from site: aix1.dom2.schuitema.nl
.
.
Retrieving data from site: aix5.dom99.schuitema
Extra data will be retrieved from site: aix1.dom2.schuitema.nl
Extra data will be retrieved from site: aix4.dom8.schuitema


#############################################################################
#                                                                           #
# Checking if all oratab entries are mentioned in the tnsnames.ora  #
#                                                                           #
#############################################################################


Checking tnsnames <-> oratab of aix1.dom1.schuitema
Checking tnsnames <-> oratab of aix2.dom1.schuitema
Checking tnsnames <-> oratab of aix1.dom2.schuitema.nl
.
.
Checking tnsnames <-> oratab of aix5.dom99.schuitema


All entries of started databases in tnsnames.ora found, continuing .....
```

```
##########################################################
#                                                        #
#    Checking tnsnames.ora of all TCP servers            #
#                                                        #
##########################################################


Checking tnsnames.ora from aix1.dom1.schuitema
     database1   checked
     database2   checked
     dbtest     checked
.
.
Checking tnsnames.ora from aix5.dom99.schuitema
     dbtest1    checked
     dbtest99   checked
     extproc_connection_data    not checked: Protocol=IPC
     <oracle_sid>_BEQ   not checked: Protocol=BEQ
Extracting tnsnames from aix1.dom1.schuitema
     Adding database database1
     Adding database database2
     Adding database dbtest
.
.
Extracting tnsnames from aix5.dom99.schuitema
     Adding database dbtest1
     aix2.dom1.schuitema is NOT online: dbtest88  not checked
     Adding database dbtest99

Removing duplicate servernames

##########################################################
#                                                        #
#      Merging production and test tnsnames.ora          #
#                                                        #
##########################################################



##########################################################
#                                                        #
#   Building the tnsnames.ora for the PC platform        #
#                                                        #
##########################################################


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

```
#                                                                      #
#   Some sites did not respond, tnsnames.pc may be incomplete          #
#           Lines copied from previous version!                        #
#                                                                      #
# Please check your old version with this one on the next sites:       #
# aix2.dom1.schuitema                                                  #
# aix11.dom33.schuitema                                                #
#                                                                      #
########################################################################


#########################################################
#                                                       #
# The tnsnames.ora for the PC is called tnsnames.pc     #
#                                                       #
#########################################################
```

EXAMPLE 2: A PART OF THE LOG WITH ERRORS

Error1

```
.
.
Checking tnsnames.ora from aix5.dom99.schuitema
     dbtest1    checked

#####################################################################
#                                                                   #
# xRef Error(s) found in tnsnames.ora: please repair it             #
#                                                                   #
#####################################################################


Host: aix2.dom1.schuitema
   production_1 does not exist on host aix2.dom1.schuitema
   entry in tnsnames.ora on host aix2 marked as an invalid entry

Host: aix3.dom1.schuitema
   dbtest2 does not exist on host aix3.dom1.schuitema
   entry in tnsnames.ora on host aix3 marked as an invalid entry



#####################################################################
#                                                                   #
#  tnsnames.ora file(s) containing errors                           #
#  script continuing only due to variable ignore_all_errors         #
```

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


Extracting tnsnames from aix1.dom1.schuitema
.
.
.

############################################################
#                                                          #
#   Building the tnsnames.ora for the AIX platform         #
#                                                          #
############################################################




######################################################################
#                                                                    #
#  tnsnames.ora file(s) / oratabs containing error(s)                #
#                                                                    #
#  SCRIPT CONTINUED ONLY DUE TO VARIABLE ignore_all_errors           #
#                                                                    #
#  STRAINED GENERATION OF tnsnames.pc AND tnsnames.aix               #
#                                                                    #
#        PLEASE REPAIR ERRORS, FILES COULD BE IN ERROR               #
#                                                                    #
######################################################################
```

You can add the entry to the oratab to resolve this error, or delete this entry in the tnsnames.ora.

Error2

```
############################################################################
#                                                                          #
# Checking if all oratab entries are mentioned in the tnsnames.ora         #
#                                                                          #
############################################################################


.
.
.
Checking tnsnames <-> oratab of aix5.dom2.schuitema
-> Entry test_dtb2 at the site aix5.dom2.schuitema is not found in
tnsnames.ora

.
```

```
.
.

#########################################################
#                                                       #
# Error(s) found in tnsnames.ora: please repair it  #
#                  exiting script ....                  #
#                                                       #
#########################################################
```

To resolve this error, add the entry of test_dtb2 in the tnsnames.ora of aix5.dom2.schuitema, or remove the line test_dtb2 in the oratab on this server.

EXAMPLE 3: TNSNAMES.PC

This file can be transferred to your PC as tnsnames.ora.

Notes:

- Aix1 is a production server; it contains two entries for each database.
- Aix5 is a test server; it contains:
  - Four entries for each database if the databasename is not the same as a production database.
  - Two entries for each database with the same name as a production database.

```
#####################################################
#                                                   #
# date     : Wed May  9 10:49:56 DFT 2001        #
# filename : tnsnames.ora                           #
# file generated by gather_tnsnames                 #
#                                                   #
#####################################################


database1 =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix1.dom1.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=database1)))

database1.world =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix1.dom1.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=database1)))
```

```
.
.
dbtest1 =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=dbtest1)))

dbtest1.world =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=dbtest1)))

dbtest1_aix5 =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=dbtest1)))

dbtest1_aix5.world =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=dbtest1)))

database1_aix5 =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=database1)))

database1_aix5.world =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=aix5.dom99.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=database1)))

.

.
```

EXAMPLE 4: REQUIRED FILES

```
File : also_tnsnames.add
Nt_database1 =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=nt_host1.dom1.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=oracle)))
Nt_database1.world =
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=nt_host1.dom1.schuitema)
(PORT=1526))(CONNECT_DATA=(SID=oracle)))

File : diff_tnsadmin_sites
aix1.dom1.schuitema /u01/tns_admin1/network/admin
aix3.dom1.schuitema /u01/tns_admin2/network/admin
aix3.dom1.schuitema /u01/tns_admin2/test_network/admin
aix3.dom1.schuitema /u01/tns_admin2/another_network/admin

File : $password_dir/$password_filename:
Secret_password

File : $general_dir/all_aix_servers
```

```
aix1.dom1.schuitema
aix2.dom1.schuitema
aix3.som1.schuitema
aix5.dom4.schuitema
aix1.dom6.schuitema
aix6.dom6.schuitema
aix5.dom99.schuitema

File : $general_dir/all_test_servers
aix5.dom4.schuitema
aix1.dom6.schuitema
```

EXAMPLE 5: CRONTAB ENTRY

To start collecting the data on each Sunday at 15:00 hours using the parameter ‘debug’:

```
00 15 * * 0 /oracle_admin/tnsnames/gather_tnsnames Debug 1>/
oracle_admin/tnsnames/gather_tnsnames.log 2>&1
```

The directory is a separate one so you know for certain that no files are destroyed that you or someone else might want to keep.

Redirection of the output in a file is more satisfying than reading the large amount of mail **- debug** generates; in my case, about 1200 lines of output.

Teun Post
Unix/Oracle Specialist
Schuitema NV (The Netherlands)

Have you come across any undocumented features in AIX 5L? Why not share your discovery with others? Send your findings to us at any of the addresses shown on page 2.

AIX news

IBM has announced Version 3.8 of Tivoli Risk Manager, which provides a security console to monitor, view, and manage alert events, and which runs on AIX, as well as NT and Solaris.

It enables sites to identify and manage threats and vulnerabilities, enforce security policies, and enable system administrators to identify different types of threat and attack. Besides historical reporting, it has several predefined reaction tasks to help resolve urgent security issues or unauthorized accesses.

It integrates with multi-vendor security products to provide comprehensive security management, and integrates with Tivoli's network, system, and security management products.

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

* * *

Micro Focus International has announced Server Express 2.0.11, an environment for integrating, developing, and deploying COBOL applications on Web-enabled open platforms.

This latest version offers expanded Java and COBOL interoperability, allowing Web application deployment created using the wizards in Micro Focus Net Express.

This Java/COBOL support is currently provided on AIX, HP-UX, Solaris on SPARC and Intel, and UnixWare 7.1.1 UDK.

For further information contact:
Micro Focus, 9420 Key West Avenue, Rockville, MD 20850, USA.
Tel: (301) 838 5000.
URL: http://www.microfocus.com/products/microfocus/server_express/.

* * *

IBM has announced Version 4.0 of its WebSphere Site Analyzer for Web site visitor statistics and analysis. It has a new Web Tracker for near real-time data capture and reporting, support for Windows NT/2000, Linux, AIX, and Solaris, and Web log analysis.

It's shipped with DB2 and supports Oracle, provides shopping cart analysis for WebSphere Commerce Suite users, and provides proxy traffic analysis for WebSphere Edge Server users, as well as campaign analysis and rule effectiveness.

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

* * *

Firstlogic and Torrent Systems have announced availability of a joint product enabling AIX sites to perform information quality processing on massive data volumes.

For further information contact:
Firstlogic, 100 Harborview Plaza, La Crosse, WI 54601-4071, USA.
Tel: (608) 782 5000.
URL: http://www.firstlogic.com
URL: http://www.torrent.com.