UNIX ESSENTIALS FOR THE BEGINNER DBA

John Kanagaraj, DBSoft Inc

INTRODUCTION

Mixed environments containing both NT and UNIX/Linux based Oracle servers are commonplace in almost any organization, and the DBA has to deal with all these operating systems. In this paper, we will deal with the practical essentials of UNIX, with topics including basic as well as slightly advanced topics such as interpreting Operating System (OS) performance statistics. Quick tips include a repertoire of commands that are put together to form useful snippets of code that can be plugged into more complex shell scripts as well as stand alone by themselves. Though the paper is generic to UNIX, it will deal with some of the essential specifics of the more popular UNIX/Linux variants. Although aimed primarily for the DBA who is transitioning to UNIX, we hope that DBAs experienced in UNIX will also benefit from some of the tips presented here.

A BRIEF HISTORY

UNIX and its variants have had an interesting history. Rising from the ashes of a failed project called MULTICS (Multiplexed Information and Computing Service), UNIX was conceived by a group of scientists from Bell Labs sometime in early 1969. Dennis Ritchie and Ken Thompson performed the first port to a PDP-11 computer in 1970 for three Bell Labs patent department typists who performed text processing - the first 'commercial' application of UNIX. Initially written in assembler, it was later rewritten in 'C' that was also developed by the same team. When Ken took a six-month sabbatical at the University of California, Berkeley, he naturally taught the UNIX system. It was an instant hit, and the staff and students at Berkeley continued to enhance it - this finally evolved into what was known as the Berkeley Software Distribution (BSD) UNIX. The rest is history - UNIX and particularly the BSD versions gained strength and gradually crept into the commercial world in the late '70s and early '80s. By the '90s, it was the de-facto Operating system for most minicomputer hardware vendors and had already played a significant role in the development of both TCP/IP and the Internet. Due to its various strengths, it is now the premier platform of choice for most IT processing needs.

And since we are talking history, we will have to tackle that UNIX clone here – namely Linux. Although Linux does not include any UNIX code, it does mimic many of the features specific to UNIX. Developed by Linus Torvalds in 1991 under the "GNU Manifesto", Linux is an Open Source version of UNIX. Similar to UNIX, it rose from the ashes of the now almost forgotten MINIX. Incidentally, "GNU" is a recursive acronym for "GNU is Not Unix"!

A QUICK OVERVIEW

As mentioned before, UNIX has a number of variants, popular among them being

- Solaris (or SunOS) from Sun Microsystems
- HP-UX (or 'h-pucks' as it is affectionately known) from Hewlett-Packard
- AIX from IBM
- Tru64 from Hewlett-Packard (Originally from DEC, purchased by Compaq and merged with HP!)
- IRIX from SGI
- Dynix/ptx from Sequent (now IBM)
- SCO UNIX from UNISYS (previously XENIX)

All these variants evolved from the same or similar code base and thus exhibit the same generic behavior. However, you will find that there are some slight differences in functionality, capability and behavior, some of which can be quite irritating. For reasons of sanity, we will refer to the generic functionality as being that of 'UNIX' and make specific references when a variant needs to be mentioned. We will also classify Linux under this term as far as functionality and command structure goes. ['UNIX' and Linux gurus - please do not take exception to this!]

Did we mention before that UNIX is a versatile Operating system? It is! The power of UNIX is based on features such as Multitasking, Multi-user support, Portability and Flexibility that have been in-built into the OS and the commands since its inception. UNIX however, has the reputation of being somewhat unfriendly to beginners. (Experienced users would however restate this as 'UNIX is not unfriendly – it is just particular whom it is friendly with!') That is what we would like to address in this paper - how a DBA can start on the path of becoming a friend of UNIX. If you are already well along this path, you might be able to pick up some new tricks and techniques.

UNIX ARCHITECTURE

UNIX is organized in three layers:

- The kernel, which schedules tasks and manages storage
- The shell, which connects the OS and the user, interprets users' commands and executes them
- Other tools and applications that add functionality to the operating system

The kernel is the heart of the Operating System. It controls the hardware and responds to 'OS Calls' from the Shell and the various tools and applications it supports. The Kernel is configurable, and it is essential that a DBA have some basic understanding of some of the concepts therein.

The shell forms a layer around the Kernel and comes in many varieties. All are command driven (read: unfriendly to many that are accustomed to a GUI-only interface). You have the choice of Bourne, Korn, C or Bash shells. All of them have their faithful followings, although Bourne and C are less frequently used nowadays. The Korn shell is widely used, and Bash is popular as well. If you are starting anew, the Korn shell is a good one to start with. A large number of users use it, and it is upward compatible with the Bourne shell. Bash is also becoming popular, especially in the Linux community. Whatever your choice, the function of the shell remains the same: it is the interface between the kernel and the user. Commands can be executed interactively or be submitted non-interactively. Commands organized into batch files are known as 'shell scripts'. These batch files can also be executed interactively as well as be scheduled to run non-interactively.

No OS is complete without it's repertoire of commands, and UNIX has a rich collection of them. The only issue that still can be painful at times is the complete absence of logic in naming these commands. For example, you 'cat' a file to read it and 'vi' a file to edit it. [Actually 'cat' is a shorter version of 'concatenate' the specified files.] 'Biff' is a utility that lets you know you have mail in your UNIX account and was named after the author's dog that used to fetch snail mail and the paper. This inconsistency in naming is the result of too many programmers developing and extending the OS, but we aren't really complaining as they have made a good job of it. The point we want to make is that when you start off, you might find these inconsistencies bothering you. Don't give up though - it will all become second nature soon!

As a DBA, you will use the Shell and the commands most of the time. We will explain the key commands using examples. Please note our conventions - Entries that need to be made by the user are in **`bold`** and our comments are in *`italic`*. The output of commands is shown using normal `courier` font.

GUI (YES!)

UNIX is not entirely unfriendly. Most variants support a GUI (Graphical User Interface) usually via a windowing technology known as X-Windows. This has been packaged in many formats and versions. A few years ago though, most of the vendors settled on the CDE (Common Desktop Environment). Although this has created some level of standardization, the 'grand scheme' of CDE displacing MS Windows on the Desktop was never realized. X-Windows is still in use today though, and is required when installing Oracle software. Hence, the DBA does need to know of and use X-Windows. You will need an X-Terminal emulator such as Exceed or Reflection-X to perform this installation if you don't have access to a graphical workstation. Note that the DISPLAY variable needs to be set for any X-Windows operation. In addition, you will also have to run the 'xhost' command on the Server for you to be able to open and use an X-Windows screen. Certain other tools also require X-Windows. The GUI version of administrative tools such as SAM on HP-UX, SMIT on AIX and AdminTool on Solaris require X-Windows. We would certainly not recommend using the CDE File manager for day to day operations - the shell is quick, flexible and extremely lightweight.

FEATURE HIGHLIGHTS

It is impossible to detail or even list all the features of UNIX in this short article. Hence, we will highlight only those features that have direct relevance to an Oracle DBA. Further details are available from a variety of sources including books, web-sites and UNIX Listserv archives. Before we launch into the details, we do need to mention that many of the operations described here need some level of proficiency with an editor. The 'vi' editor is the de-facto standard editor in all UNIX systems and usage is unavoidable. 'Vi' is very powerful and extremely flexible, but new users find it a little difficult to adapt to. The best way of dealing with 'vi' is to remember that a session is always in one of two modes:

- Command mode, when the editor responds to movement, mass delete or change, and other file commands
- Insert mode when you can enter text in a line.

Switching from Command to Insert requires entering the 'a' or 'i' character and switching back to Command mode requires the 'ESC' key. If you are comfortable with switching back and forth, half the 'vi' battle is won! 'Vi' has very powerful search and replace and other text manipulation capabilities as well. It is best to start using 'vi' and learn along the way. The author is still learning new stuff in 'vi' after 18 years!

DIRECTORIES AND FILES

The UNIX kernel supports a hierarchical structure with the Root directory and subdirectories underneath. All devices are addressed as files, which makes it very flexible. Thus terminals are files - the input to and output from a terminal is treated as if it comes from or goes into a file. This coupled with the fact that UNIX supports file redirection provides a powerful method of handling screen I/O and other file operations. This is also known as I/O redirection. For example, the following command segment automates the collection and preservation of SGA allocations: Instead of typing in the SQL*Plus commands one after another, we have setup a 'command script' that simulates the user's keystrokes. The comments are sufficient explanation of what occurs.

```
$ cat > cmd.in          -- Create a file by con'cat'enating the terminal file to 'cmd.in'
connect interal         -- This is the sequence of commands that a DBA would have entered
show sga
exit
<Control-d>     -- This marks the end of file (similar to <Control-Z> in MS, the character is not stored though
$ sqlplus /nolog < cmd.in > cmd.out   -- Replace the terminal input by the file 'cmd.in'
                                      -- and re-direct the output to the file 'cmd.out'
$ cat cmd.out     -- Con'cat'enate the previously captured information from the physical file to the terminal

SQL*Plus: Release 8.1.7.0.0 - Production on Tue Feb 11 15:07:36 2003
```

```
(c) Copyright 2000 Oracle Corporation.  All rights reserved.

SQL> Connected.
Total System Global Area 1357615264 bytes
Fixed Size                    73888 bytes
Variable Size             701116416 bytes
Database Buffers          655360000 bytes
Redo Buffers                1064960 bytes
SQL> Disconnected from Oracle8i Enterprise Edition Release 8.1.7.3.0 - Production
With the Partitioning option
JServer Release 8.1.7.3.0 - Production
$
```

The terminal files have special numbers - 0 for standard input, 1 for standard output and 2 for standard error. If the program had written an error to the standard error, we could have captured it with the following segment:

```
$ sqlplus /nolog < cmd.in > cmd.out 2> cmd.err
```

Most of the directories hanging off the '/' (root) directory have special significance. For example, the '/etc' directory contains most of the OS and network configuration files; '/lib' contains shared code and object libraries; '/tmp' contains user's temporary files; '/dev' files contains a list of all devices; /'var' contains some configuration and spool files.

Directories and files are also organized into 'filesystems' - these are essentially units of storage that can be addressed as individual chunks where directories and files can be created, and are loosely equivalent to Windows 'Drives'. These filesystems can be 'mounted' under the 'root' directory or it's sub-directories, which now makes them accessible. Note that files cannot expand across filesystems, although directories and files can be 'soft-linked' across to other filesystems, thus creating the illusion of having very large file systems.

ACCESS PERMISSIONS

Files and directories are protected by access permissions. A sample list is shown below. We will use some of the files (marked in ***bold and italic***) to highlight some of the security features of UNIX.

```
$ pwd          -- Display the current working directory
/u01/app/oracle/product/8.1.7
$ ls -l -- List all files in a directory
total 70
drwxr-xr-x   3 oracle   dba    96 Nov  4 11:30 admin
drwxrwxr-x   3 oracle   dba    96 Nov  4 11:23 Apache
drwxrwxr-x   5 oracle   dba    96 Nov  4 11:24 assistants
drwxr-xr-x   2 oracle   dba  4096 Nov  4 11:28 bin
-rw-r--r--   1 oracle   dba  1454 Nov  4 11:30 MYDB.env
<snipped for brevity>
lrwxrwxrwx   1 oracle   dba    25 Nov  4 11:21 JRE -> /u01/app/oracle/admin/MYDB/jre/1.1.8
drwx------   6 oracle   dba    96 Nov  4 11:23 jsp
<snipped for brevity>
drwxrwxr-x   6 oracle   dba  1024 Nov  4 11:25 relnotes
-rwxr-xr-x   1 oracle   dba  8012 Nov  4 11:28 root.sh
<snipped for brevity>
```

```
$ ls -l bin/oracle  -- List the file 'oracle' in the 'bin' subdirectory
-rwsr-s--x   1 oracle   dba      33712648 Nov  4 11:28 bin/oracle
```

This is our very familiar Oracle home directory. There are three levels of permissions - 'r' for 'read', 'w' for 'write' (update/delete/insert/rename) and 'x'. The latter allows execute permission for a binary executable file or a shell script. On the other hand, it allows 'change directory' privileges when placed on a directory (Confusing, right!?). There are three sets of such permissions - one each (of r, w and x) for the owner of the file ('oracle' in this case), the group that is allowed that particular access (all users in the 'dba' group in this case), and all others in the system. Many UNIX variants (AIX and Solaris for example) also support Access Control Lists (ACL), or extended security features, but we will not deal with them here.

The 'admin' directory will thus allow all users to change directory to it and list the files therein. This is by virtue of the 'r' and 'x' permission for all three sets. However, it disallows changes by any of them except the owner - users in the 'dba' group are disallowed changes as well. The 'assistants' directory on the other hand, will allow users in the 'dba' group to change the directory contents - i.e. it will allow such users to rename, delete or create files in the directory. Note also that the presence of a 'd' in the first character denotes this to be a 'directory' file rather than an ordinary file. In addition, the file 'JRE' is a link file (denoted by the 'l' in the first character) that points to another directory. None other than 'oracle' will be allowed any operations in the 'jsp' directory, while the MYDB.env' is an 'ordinary' file. The 'root.sh' is an executable file - actually a script - that can be executed by any user.

Finally, the 'oracle' binary executable file in the 'bin' directory has the special 's' (Set User ID or SUID) permission. This feature, rather unique among Operating systems, allows the executor to assume the rights and permissions of the owner, albeit only during the execution of that program. Thus, any user executing the 'oracle' binary will be seen by all other entities as being executed by 'oracle' rather than that user. If you have wondered how a normal user updates Oracle structures writeable only by 'oracle', then you have your answer above! Reads and writes on the directory specified by UTL_FILE_DIR as well as to the Shared Memory area (SGA) both of which are owned and protected by 'oracle' are examples of this type of special access.

Note also that UNIX is case-sensitive - 'JRE' is not the same as 'jre'. Also, any files that begin with a '.' are 'hidden' - they are not normally listed in the output of an 'ls' command unless the '-a' option is added.

[Before we wind up the file type discussion, I have to mention that there are seven types of files. I will not reveal the answer though, as I use this as a 'test-the-waters' question while conducting DBA and SysAdmin interviews!]

CONFIGURING THE OS

Most variants of the UNIX OS are highly configurable, and it is important that the DBA understand some of the relevant parameters. Oracle uses UNIX's Interprocess Communication Process (IPC) to perform some essential tasks such as creation of and access to the SGA that resides in Shared Memory. This includes the create/attach to Shared Memory segments (SHM parameters) as well as Set and Test on Semaphores (SEM parameters). The parameters in the following table are the minimum values required to host a single Oracle 9i database instance - as noted in the 'Oracle 9i Quick Installation Procedure for SPARC Solaris' manual:

Kernel Parameter	Setting	Purpose
SEMMNI	100	Defines the maximum number of semaphore sets in the entire system.
SEMMNS	256	Defines the maximum semaphores on the system. This setting is a minimum recommended value, for initial installation only. The SEMMNS parameter should be set to the sum of the PROCESSES parameter for each Oracle database, adding the largest one twice, and then

		adding an additional 10 for each database.
SEMMSL	256	Defines the minimum recommended value, for initial installation only.
SHMMAX	4294967295	Defines the maximum allowable size of one shared memory segment. 4 GB = 4294967295
SHMMIN	1	Defines the minimum allowable size of a single shared memory segment.
SHMMNI	100	Defines the maximum number of shared memory segments in the entire system.
SHMSEG	10	Defines the maximum number of shared memory segments one process can attach.

In the case of Solaris, these parameters are set in the '/etc/system' file, followed by a Server restart. In the case of HP-UX, this will need to be set using the SAM (SysAdMin) utility. AIX does not need this to be set explicitly. Note that you will need to login as the 'root' administrative account to perform this change. Alternately, you can advise your UNIX System Administrator to perform this change. If you are using multiple databases or plan to add databases to that box, you might want to scale up the SEM parameter numbers, as explained above. One myth related to the SHMMNI and SHMMAX that has been floating around is that multiple SHM segments should be avoided at all costs - this is not true. Multi-GB SGAs will require multiple SHMMAX sized memory segments, with the total number of segments not exceeding SHMMNI. Also, note that these values are upper limits, and OS memory slots are usually not allocated until actually demanded by the Oracle software. With this in mind, err on the upper side when setting these values.

The 'ipcs' command is used to display IPC related parameters, and the -a option displays all Shared Memory, Semaphore and Message units used. The accompanying command 'ipcrm' can be used to remove erroneous Shared memory segments left behind by Oracle instance crashes. Use caution though while using ipcrm - you may be killing another live SGA!

STARTUP AND SHUTDOWN

As a DBA, you will have to supply startup and shutdown scripts to the SysAdmin. These will normally be installed in the '/etc/rc' directories, although AIX uses calls from the /etc/inittab file to perform shutdown/startup. The /etc/inittab file is worth understanding on any variant, as this file controls the sequence of startup and actions per run levels on a UNIX box.

The '/etc/inittab' file is read by the 'init' command - the OS process spawner which is the first process to startup on server restart. Based on the 'run level', init would read inittab and start the process specified for that level. This in turn launches all other scripts that either start (at run level 2,3 and 4) or shut (at run level 0) all other processes. You can use the 'dbstart' and 'dbshut' scripts that reside in the $ORACLE_HOME/bin directory as templates for the so-called rc (startup/shutdown) scripts that startup and shutdown the database.

MISCELLANEOUS OS FEATURES

Each variant provides additional tuning parameters that may help in getting the best performance from the Server. This includes Disk tuning parameters such as Read-ahead and Virtual Memory Management (VMM) settings that will influence the paging, swapping and free memory level of the OS. The 'Oracle Installation and User Guide' for that particular UNIX variant is usually a good place to start with in this case.

Just as Oracle binaries need to be patched to correct errors, the UNIX kernel also needs patches. We will later table some commands that can be used to display current patch levels as well as to install them. Most of them require root privilege

though, so your SysAdmin needs to install them. Always check the Oracle Installation and User Guide for that database version and platform for minimum OS patch and revision levels.

UNIX also provides Virtual Memory Management whereby the total memory requirement for programs and data that are currently loaded and executing can exceed the physical memory available. UNIX performs paging - older, unused or infrequently accessed 'pages' of executable code, user memory or OS disk buffers are written out to a special part of the disk known as the Swap area, thus creating space in RAM for new programs being loaded. This is called 'paging out'. If required, these pages of memory can then be 'paged in'. When a severe memory shortage occurs, then the system starts swapping, wherein whole processes (rather than pages of it) are written out to the Swap area. Paging is normal, whereas Swapping is not. We will see how to determine this later. It is important that the DBA understands the difference between the two and can identify the occurrence of swapping.

All UNIX servers preserve System level error logs. The location and tool used to access this varies widely – this is tabled later. You should look at these logs in case an OS error is returned by Oracle or you suspect an OS problem is causing an issue.

Finally, most UNIX variants provide Disk Volume managers to ease storage management, either using in-built functionality such as in AIX or external add-ons such as the Veritas Volume Manager. Volume Managers provides additional functionality such as Journalled File systems (JFS), online expansion of filesystem sizes, volume group and volume management, I/O enhancements (e.g. Quick I/O in Veritas) and Filesystem mirroring. The DBA has to be aware of such technologies, understand and appreciate their use. Storage Management also now includes new technologies such as SAN (Storage Area Networks), NAS (Network Attached Storage). In such cases, the traditional 'disk' no longer is one physical device anymore, and understanding the underlying architecture will become important to the DBA for design and performance considerations.

The UNIX SysAdmin should become a DBA's best friend. Develop close ties, share the same view on performance and attack problems jointly with the SysAdmins. Our advice is to keep your UNIX SysAdmins regularly supplied with candy, treats and other consumables in exchange for privileges, disk space and other goodies!

'CRON'Y - YOUR BEST FRIEND

UNIX supports periodic job scheduling using the 'cron' facility. This per-user text file configures a list of jobs run by the 'cron' utility. Cron executes the specified commands or shell scripts at the specified time and interval on behalf of the user in the background. This file is edited using the 'crontab' command that invokes the 'vi' editor on that user's crontab file. The format of the entry in this file is somewhat convoluted, but it consists of the time in minutes, hours, day of the month, month of the year and the day of the week. A day and/or time range as well as selected points can be specified for the execution of each of these commands, as shown in the example below:

```
# Perstat snapshot
# Executes the specified Korn shell script on the hour (00) every
# four hours (00,04,08,12,16,20) every workday (1-5 specifies Mon – Fri)
00 00,04,08,12,16,20 * * 1-5 /u01/app/oracle/local/scripts/stats_snap.ksh MYDB >
/u01/app/oracle/local/logs/stats_snap_MYDB.out 2>&1
```

Note that the whole command is contained in one single line. A line break may be introduced using the '\' (backslash - escape) character at the end of the line. You need to be careful while editing the crontab file. On many variants, one single syntactical or semantic error may cause cron to silently ignore all the commands, and that can be disastrous. The best way to overcome this is by create a copy, edit it using 'vi' and replace it back as shown below. This way, a copy is also retained for your records. You should preface all comment lines in this file using a '#' in the first character of a comment line. Be liberal with your

comments – they will help you when you come back to figure out why you did what you did! 'Cron' can be your 'crony' (friend) when used properly!

```
$ crontab –l > crontab.Jan_30_2003  -- Create a copy of your cron table with the –l
option
$ vi crontab.Jan_30_2003            -- Edit the file using the standard 'vi' editor
$ crontab –e < crontab.Jan_30_2003  -- Replace your cron table using the –e option
```

When a one-time submission is required, you can always use the 'at' command to run the command or script once at a specified time. Both 'at' and 'cron' require that the UNIX SysAdmin setup privileges for the specified user using the 'at.allow' and 'cron.allow' files (or 'at.deny' and 'cron.deny' files as the case may be).

PIPES (NOT THE SCOTTISH KIND!)

All the shells support piping. Using what is known as unnamed pipes, the standard output of one process can be sent to the standard input of another, subsequent process. This can form an extremely powerful and flexible chain as shown below. In this example, we use the 'df' command to determine the free space in all the mounted filesystems and pipe the output to the 'sort' command to display filesystems ordered by free space.

```
$ df -k           -- df displays free space in mounted filesystems in Kb (-k option)
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/vx/dsk/rootvol  2033103 1439129  532981    73%    /
/proc                      0       0       0     0%    /proc
fd                         0       0       0     0%    /dev/fd
mnttab                     0       0       0     0%    /etc/mnttab
/dev/vx/dsk/var       986131   62945  864019     7%    /var
swap                 2527384       8 2527376     1%    /var/run
swap                 2558256   30880 2527376     2%    /tmp
/dev/vx/dsk/data1    2457600 1037183 1333382    44%    /data1
/dev/vx/dsk/data2    2883584 2324263  526185    82%    /data2
/dev/vx/dsk/ftp       540608  375809  154506    71%    /ftp
/dev/vx/dsk/data 104857600 52340122 49272808    52%    /data

$ df -k | sort -nr +3   -- The output of 'df -k' is piped to 'sort' which sorts its
                        -- input in reverse numeric sequence by the 3rd field as its key
/dev/vx/dsk/data 104857600 52340122 49272808    52%    /data
swap                 2558256   30880 2527376     2%    /tmp
swap                 2527384       8 2527376     1%    /var/run
/dev/vx/dsk/data1    2457600 1037183 1333382    44%    /data1
/dev/vx/dsk/var       986131   62945  864019     7%    /var
/dev/vx/dsk/rootvol  2033103 1439129  532981    73%    /
/dev/vx/dsk/data2    2883584 2324263  526185    82%    /data2
/dev/vx/dsk/ftp       540608  375809  154506    71%    /ftp
mnttab                     0       0       0     0%    /etc/mnttab
```

```
fd                         0       0       0     0%    /dev/fd
/proc                      0       0       0     0%    /proc
Filesystem             kbytes    used   avail capacity  Mounted on
```

The flexibility is readily apparent: a simple combination of commands such as these can easily show up the top filesystems with free space when you are desperately searching for space for your Oracle datafiles. A note of caution though: Don't consider locating datafiles in the standard UNIX directories such as'/', '/var or '/tmp'! The OS needs adequate free space in these directories - your SysAdmin may also inadvertly 'clean up' these directories.

Commands can also work with named pipes. These are special 'pipe' files created by the 'mknod' command and used as if they were normal files operating on redirected I/O. This is useful in cases where you want to trick a command or utility that will only write to a named file that needs to be used in a piped stream. An example of this is the use of named pipes to perform Oracle exports that go over the 2Gb limit. Metalink provides a script that uses named pipes for this very purpose.

[Apologies - We need to get back to the basics after the little magic show above!]

MISCELLANEOUS SHELL FEATURES

UNIX provides powerful job control facilities as well. Commands and scripts can be executed in the background by simply adding an '&' (ampersand character) at the end of the command. They can also be brought back to and sent away from the foreground by using <Control-Z> and the 'fg' and 'bg' commands. You also need to know of the 'nohup' command - prefixing this to other commands that are launched in the background detaches the process completely ('No hangup' mode) - these jobs don't abort when the user logs off from that interactive session. This is shown in the examples below:

```
$ ls -lR / > /tmp/list_of_all_files 2>/tmp/error_files &    -- The '&' launches this process in the background
$ fg           -- This brings it back to the foreground
ls -lR / > /tmp/list_of_all_files 2>/tmp/error_files
$ <Control-Z>  -- Signal the job to 'suspend'
$ ^Z[1] + Stopped (SIGTSTP)        ls -lR / > /tmp/list_of_all_files 2>/tmp/error_files -- Job suspended
$ bg           -- Send it back to the background
$ [1]     ls -lR / > /dev/null 2>/dev/null&
$ jobs         -- List all background jobs
[1] +  Running                 ls -lR / > /dev/null 2>/dev/null
$ kill %1      -- Kill the specified background process (note the job number)
[1] + Terminated               ls -lR / > /tmp/list_of_all_files 2>/tmp/error_files
$ nohup ls -lR / > /tmp/list_of_all_files 2>/tmp/error_files &  -- Start the job in No hangup mode
```

All shells provide wildcard substitution, also known as 'globbing'. There are five basic constructs

*	Match any number of any character
?	Match one of anything
[...]	Match one character in the brackets
~	Your home directory
~user	User's home directory
!	Negates the meaning of the subsequent character

Thus, '*.dbf' would match all files in the specified directory that end in '.dbf', while 'abc??.dbf' would match any file that starts with 'abc' followed by any two characters and ending in '.dbf'. When surrounded by brackets, either a list or a range is matched. For example, 'abc[01][0-9].dbf' would match all files starting with 'abc', have either '0' or '1' as the next character, have any number between 0 and 9 (both included) as the next character and end in '.dbf'. Finally, the string '![a-c]*' would match all files that do not start with a, b or c. [Whew! You could really pick out specific files with such powerful pattern matching!]

No shell is complete without the ability to set and use Environment variables. Both built-in variables as well as user-defined variables are an essential part of UNIX shells. Setting and using variables differ with shells and we will consider only the Korn shell in this case: Setting up and accessing a variable is simple as shown below.

```
$ MYVAR="This is a variable"          -- Sets the MYVAR variable
$ YOURVAR="This is yours"             -- Another variable
$ export MYVAR                        -- Makes MYVAR available to subshells
$ export NEWVAR="This is another variable" -- Set and export the variable [Only in Korn]
$ echo $MYVAR                         -- Display the MYVAR variable, note the $
This is a variable
$ ksh                                 -- Create and enter a subshell
$ echo $NEWVAR                        -- Display the previously set and exported variable
This is another variable
$ echo $YOURVAR              -- Echoes a blank line as this variable is not accessible in the subshell
```

Note the 'export' command above. A shell can spawn a subshell or a subprocess – all exported variables are made available to these subshells and subprocesses as shown above. You need to use a '.' while launching a command that sets up your variables for that parent shell. This is required when using a shell script to set your Database Environment variables. You will need to setup the PATH, SHLIB or LIB_PATH, ORACLE_SID (or TWO_TASK), TNS_ADMIN and other NLS variables to be able to connect to a local or remote database. The shell script that launches this should be invoked with a '.' so that the parent shell obtains these variables.

You would have noticed the quoting above – quoting disables a character's special meaning and allows it to be used literally. The single quote (') and the double quote (") are used to perform quoting. Both provide similar functionality, except that the single quote suppresses the meaning of all other characters except itself. The back slash (\) suppresses the meaning of the subsequent character. Finally, the back quote (`) is used for command substitution - we will see examples of this later.

The following table shows some of the built-in variables that are accessible, some within scripts.

PWD	Current working directory; set by 'cd'
PS1	Prompt for the first command line; settable
PS2	Prompt for the continuation line; settable
$#	Total number of arguments passed into and available from a shell
$n (as in $1, $2)	Positional parameter; $0 is a special one - the shell script's name
$*, $@	All command line arguments - as one string and individually quoted

There are many other functions and features of the shells, but the ones above are an essential start. With this under our belt, we will start to put them all to good use in the next few sections.

The 'C', Korn and Bash shells also support command recall and editing. The saves on keystrokes and allows experienced users to escape finger-fatigue. Korn for example enters the command mode of 'vi' and allows single line edits using standard 'vi' commands. For example, <Esc> - K will show up the previous command line in its entirety for editing and resubmission. When in command mode, a 'v' will open full screen 'vi' on that command.

OFTEN USED COMMANDS

Now that we have some basic understanding of the shell, it is time to create a table of often-used commands, grouped by function. Most of these commands have multiple options that provide extremely rich functionality. However, as with all other aspects of UNIX, there is little consistency in option naming and usage. All these commands are documented in detail and are accessible from the command line using the 'man' command. The variant specific details can also be found therein. When combined with the 'apropos' command that lists related commands, the 'man' command is pretty much your starting point when it comes to finding out more about an action, the related commands and their options. An example is shown below: [And don't feel shy about using 'man' - The author still uses 'man' to look up obscure options even after 18 years on the job!]

```
$ apropos mail          -- Read about appropriate commands (lines snipped for brevity)
addresses       aliases (4)     - addresses and aliases for sendmail
aliases         aliases (4)     - addresses and aliases for sendmail
biff            biff (1b)       - give notice of incoming mail messages
from            from (1b)       - display the sender and date of newly-arrived mail
messages
mail            mail (1)        - read mail or send mail to users
mailq           mailq (1)       - print the mail queue
mailstats       mailstats (1)   - print statistics collected by sendmail
mailx           mailx (1)       - interactive message processing system
vacation        vacation (1)    - reply to mail automatically
$ man mailx             -- Now read the specific command

Reformatting page.  Please Wait... done

User Commands                                           mailx(1)

NAME
     mailx, mail - interactive message processing system

SYNOPSIS
     mailx [ -BdeHiInNURvV~ ]  [  -f
      [ file | +folder ]  ]  [ -T file ]  [ -u user ]

     mailx [ -BdFintUv~ ]  [ -b bcc ]  [ -c cc ]  [  -h number  ]
     [ -r address ]  [ -s subject ]  recipient ...

DESCRIPTION
     The mail utilities listed above provide a comfortable, flex-
     ible  environment  for  sending  and receiving mail messages
     electronically.
<snipped for brevity>
```

A list of often-used commands is tabled below. Multiple commands are grouped in the same line when it makes sense. Useful options are noted where required.

Function and Command	What it does
File Management	Commands that operate on files and directories
cat	Concatenate files - usually used to display a text file to the display (terminal file)
cd	Change directory - the '~<user>' moves to the specified user's home directory
chmod/chown/chgrp	Change file/directory permission, ownership and group
Cp	Copy one file to another or one or more files to a directory
csplit/split	Split files based on context (csplit) or line/byte count (split)
Find	Locate files specified by name, pattern, modification time , etc.
head/tail	Display the specified portion of the top or bottom of the file; 'tail -f' is useful for monitoring the tail end of a text file that is currently being generated, such as the output of a report
ln	Create a link to an existing file
ls	List the contents of the current or specified directory. Ls probably has the most options, but the '-l' and '-lrt' are the most popular
mkdir	Create directory
more/less/pg	Read a file with the ability to search and move around
mv	Rename a file or directory or move it to another location
pwd	Display current working directory
rm/rmdir	Remove files/directories; 'rm -r' will recursively remove all
wc	Count word, lines and characters; wc -l is the most used
File/text manipulation	Commands that can be used to operate on text within files
awk/nawk	'Awk'ward programming language; used for a variety of pattern matching and text manipulation
cut	Cut up a file along vertical lines by columns or fields
diff/diff3	Display the difference between two (or three for diff3) files
expr	Evaluate arguments as expressions; usually used for simple arithmetic
grep/egrep	Search files for patterns; egrep handles expressions
join	Joins common lines from two sorted files
paste	Paste files vertically
sed	Editor that operates on a stream of characters (as opposed to a line by line editor such as 'vi'); When combined with awk, sed is so powerful that they deserve a paper by themselves!
sort	Sort files; use -n for numeric sorts (default is character); -r for reverse

strings	Display text strings in binary executable files
tee	Replicate the standard output by placing a 'T' after a pipe
xargs	Construct argument lists and execute the succeeding command; useful when a command's arguments exceed specified limits
System status and interaction	Commands that interact with UNIX to set or obtain status
at/crontab	Schedule a command or script to run at a later time - 'one time only' with 'at', periodically with 'cron'
cal	Display the calendar for a month or year (execute 'cal 9 1752' for some interesting results!)
date	Display the current date and time
df	Display the disk free for mounted filesystems; -k specifies Kb
du	Display the disk used for the specified directory; -sk summarizes in Kb
env/set	Display the environment; set also sets in addition to display
kill	Terminate processes; -TERM is a 'safe' kill which may be ignored; -KILL (or -9) is a forced kill; Only processes belonging to the same user can be killed
lp/lpr	Print the specified file via the UNIX spooler
lpstat/cancel	Obtain status of a print job or cancel it
netstat	Display network statistics
ps	Display state of processes; -ef displays details of all processes system-wide; Format and options vary widely in variants
sar	Display System Activity Report - reports on a wide range of OS statistics
stty	Set terminal characteristics; use 'stty erase <Backspace>' to get rid of those irritating '^H' on some terminal types
truss	Trace a process and display its system calls and signals
uname	Display the name and other details of the current system. Use –a to get all the information.
who/w/uptime	Display the details of users logged in and the time system has been up; 'w' displays the number of jobs in the run queue over the last 1, 5 and 15 minutes as well as logged in users; uptime displays only the load
vmstat/iostat	Display CPU, Memory and IO statistics
Storage	
compress/uncompress	Compress/uncompress files
cpio	Copy files/directories in a generic format
gzip	Compress/uncompress files (open source)
tar	Create a tape archive; can also used to consolidate a list of files and directories into one disk file (i.e. not necessarily a tape backup)

Communication	
ftp	File transfer utility; be sure to use binary mode to transfer Data, Export/Import dmp files and executables and use ascii mode for text files
login/telnet	Connect to other UNIX servers
mailx	Send and receive E-mail; may need email gateway enabled to send and receive from external system
Miscellaneous	
banner	Convert words to banners
clear	Clear the screen and position the cursor at the top left hand corner
man	Read the manual page for a specified command
nohup	Launch the command in a 'no hangup on close' mode
passwd	Change or set the password for your account
script	Record the keystrokes and screen output to a file ('typescript' by default)
su	Switch user
sudo	Execute command as someone else; Open source software that requires installation and setup by the SysAdmin (Hint: Needs candy bribes!)
time	Time a particular command

PUTTING IT ALL TOGETHER

Many of you would have seen beautiful patchwork quilts – all of them are made from scraps of cloth of different colors and patterns. These individual scraps, while they exist in their own right, make a wonderful quilt when put together by an artistic and experienced quilter. So also can you weave intricate command structures with UNIX using various techniques such as piping and I/O redirection. We will see some examples below, and use them to smoothly flow into our next topic. These examples have self-explanatory comments – for brevity's sake, the output is not listed unless it is relevant.

```
$ ls -l | grep '^d'                -- List all the directories; the '^' sign specifies that all text lines
                                   that begin with the succeeding characters should be filtered using the
                                   grep command

$ ls –l | grep –v '^d'             -- List all files, excluding directories

$ tail –1000 alert_MYDB.log | more   -- Browse the last 1000 lines

$ find / -type f -print |  xargs grep -il SECRET_PASS > $HOME/pass.out 2> /dev/null
-- List all files in the system (starting from the '/' or root directory) containing the text
'SECRET_PASS' both in upper case or lower and store it in my home directory in a file named  'pass.out'.
Files that are not accessible will be ignored and error messages to that effect are directed to /dev/null
– a special file that is a digital black hole
```

```
$ sar | head -4; sar | tail -1

SunOS mydbserver 5.8 Generic_108528-15 sun4u    01/12/03

00:00:00    %usr    %sys    %wio   %idle
Average       26       9      18      47
-- Display CPU usage averages from 'sar' for the day, including headings; the ';' allows more than one
command to be entered in one line, and output is displayed without a line break subsequent to each other

$ ls –l | tee file_list            -- Display the files in the directory and create the file 'file_list'
                                   using the 'tee' command at the same time

$ grep MYDB *.ksh | wc -l          -- Count the occurrences of 'MYDB' in all the files ending in '.ksh'

$ echo "The size of MYDB is `du -sk /u*/oradata/MYDB/*.dbf | awk '{sum+=$1} END {print sum}'` kb"
-- Summarize the size (in Kb) of all database files for MYDB located in the specified directories using
the summing capability of 'awk'. The 'sum+=$1' command adds up all the occurrences of the first column and
the 'print sum' prints the value when all rows have been displayed. The backquotes enable command
substitution

$ ps -eo pid,pcpu,ruser,time,etime,args | sort -nr +1  | head -20
-- List the top 20 CPU consuming processes (options valid for Solaris only)

$ cal 2 2003 | awk '{print $2}'| grep ^[0-9]|tail -1
-- Determine the last Monday in the specified month (February, 2003 in this example)
```

We will use command structures such as these in the subsequent section.

SHELL SCRIPTING BASICS

Shell scripting is the art and act of putting together UNIX commands and utilities using control structures where required. While the general flow of logic is similar across the different shells variants, the syntax is quite different. Hence, it is necessary that you decide which particular variant to follow and stick to that decision. Based on the availability of scripts and skills, you will find that the Korn shell is a practical choice. Hence, we will deal with Korn shell structures here. To create any shell script, create and edit the file using an editor such as 'vi', and assign read ('r') and execute ('x') permissions on that file using the 'chmod' command. Name the file such that it ends in '.ksh' - this is not a requirement, but is just good programming practice.

Remarks and comments should be preceded by the '#' symbol which informs the shell command interpreter to ignore that line or subsequent parts of that line. The exception to this rule is the first line in a shell script that can be used to force the shell being used to execute that script. This will 'protect' the integrity of your execution environment - even if your default shell is Bourne or C, the script will execute in the Korn environment in the example below. This comes in handy if you want to schedule this script to run via 'cron' - by default, 'cron' executes in the Bourne shell environment. Not introducing this will cause statements such as 'export PATH=$PATH:$HOME/bin' to fail as this syntax is not supported in the Bourne shell.

```
#/bin/ksh
# This forces the Korn shell to be used - the location may change /usr/bin/ksh
# depending on the UNIX variant
```

SHELL CONTROL STRUCTURES

As with all other programming languages, the Korn shell has many control structures. They include the if-else, while and for loops, and case structures. Rather than dealing with the syntax of these structures, we will show off the features using some examples. This section has been taken from a script that checks the alert logs for error messages.

```
#/bin/ksh
#
# trap signals – when any of the following signals are sent via the kill command, trap the signal,
# perform required cleanup and exit gracefully with a specific exit code
trap 'rm -f /tmp/check_alert_log.$$.*; exit 3' 0 1 2 3 15
# This is an example of a 'case' structure
# Obtain no-of-secs to sleep from parameter
case $# in             # The $# inbuilt variable stores the number of parameters passed to a shell script
   2) RUN_COUNT=$1            # If the number of parameters ($#) is 2, then perform these actions
      DB_LIST="$2"            # Set the RUN_COUNT to the value of the first variable  & DB_LIST to the 2nd
      ;;                      # The string ;; signifies the end of one case branch
   *)  echo "\nUsage : $0 <no_of_secs_to_sleep>\n"   # The '*' signifies end of the case and is a catch-all
       exit 2                 # Exit the program with an error code
esac           # The reverse string of 'case' – namely 'esac' signifies the end of the Case section

RUN=0
# Loop around, sleeping no-of-secs (parameter)
while [ $RUN –lt $RUN_COUNT ] # The while command tests the condition at the beginning of the execution
 do                           # The 'do' marks the start of the while loop
RUN=`expr $RUN + 1`           # The expr acts on the RUN variable and increments it – uses command substituion!
# Run through the DB_LIST
 for ORACLE_SID in $DB_LIST   # For processes a list until all variables are exhausted
 do                           # In this case, the $2 parameter should have been passed in within quotes
  . $DB_LIST.env              # Note the '.' which signifies that the subscript should execute within the
                              # context of the source – this is referred to as sourcing the environment
 if [ ! -f "$LAST_CHECK_FILE" ]; then        # This checks for the absence of the file named by the variable
   # if this is the first time we run the script against the database
   OLD_SIZE=0
   FIRST_TIME=yes
 else                         # Else marks the start of the code when the condition is not met
   LAST_CHECK_DATE="since `awk '{print $1}' $LAST_CHECK_FILE`"
   OLD_SIZE=`awk '{print $2}' $LAST_CHECK_FILE`      # The OLD_SIZE variable is set using command substitution
 fi                           # As with esac, the reverse of 'if' marks the end of the structure
 done                         # End of the for loop
done                          # End of the while loop
exit 0                        # Exit gracefully with a status code of 0
```

When used either in a stream of commands or shell scripts, exit codes provide a great way of passing success, failure or other information. The exit code of a previous command can be tested using the special variable '$?', as shown in the example below:

```
# Search for the following successful completion message in an export log file
# If the grep command works successfully, then a '0' exit code is returned, otherwise a '1' (non-zero) is
returned
STR=`grep 'Export terminated successfully without warnings' $EXP_LOG`
if [ $? –ne 0 ]; then
```

```
        # Send mail only if unsuccessful
        mailx –s "Export unsuccessful – please check $EXP_LOG" you@your_email.com < $EXP_LOG
fi
```

When managing multiple variants, you might have to customize your shell scripts so that they work across all the platforms. You can use the 'uname –s' command which will always detect the name of the Operating system. Based on this, you can always set the right version of the command and customize your script.

In the examples above, we have also dealt with some of the best practices – passing of parameters, capitalize variables so that they stand out, add comments liberally, use command substitution and piping for obtaining information elegantly, ensuring portability and use of exit codes. Sometimes, the best way to learn scripting is to jump in with a simple script and add functionality as you go along.

UNIX PERFORMANCE OVERVIEW

Most UNIX variants provide a variety of both standard as well as variant specific commands that expose a large - sometimes dizzying large – amount of OS performance statistics. It is important for the DBA – beginner and advanced – to be able to both understand and process this large amount of information as well as tie it back to the equally large amount of Database performance statistics. The Statspack report for the selected period should be used along with these OS statistics to diagnose performance issues.

We will describe a few commands and the interpretation in some detail here, as this kind of practical information is not readily available. We will also look only at standard commands such as 'sar', 'vmstat' and 'iostat' – the information provided by these commands is mostly uniform across the majority of the variants.

SAR – YOUR STARTING POINT

Sar – an acronym for 'System Activity Report' – reports almost the entire gamut of OS performance statistics, both off current as well as previously recorded snapshots of OS internal counters. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications and paging. On most platforms, it needs to be configured by setting up the '/usr/lib/sadc' collector for an appropriate period and sample interval. This has to be performed by the UNIX System administrator, and consists of a few entries in the root's crontab. Be aware that filenames are recycled and this limits the number of days that this file is retained. As well, depending on the sampling interval and the number of disk partitions, these files can grow to large sizes. By default these files are stored in the '/var' file system and you should check the space availability before implementing this collection.

When invoked with the -f option, sar reports the required statistics from the named file. While using this mode, a start and end time can also be specified to identify the required portion to report from. Sar can also be invoked in the interactive mode with a sampling time interval and number of times to sample.

We will not attempt to assign an absolute warning level for any of the values from the sar reports, as they can vary from system to system and application to application. We will however note ballpark estimates that are generally indicative of problems. The trick is to perform a snapshot of these figures during periods of normal activity when response times are good (or at least acceptable) and use that as a baseline when attempting to find out where the new bottlenecks are. We will deal with a few essential options of sar that deal with the CPU, memory and I/O - this includes the following:

- -u (default) details the CPU Utilization in terms of percentage CPU that executed the user code (%usr), System calls (%sys), Wait for I/O (%wio) and Idle waiting for work (%idle). This can be used to look at an overall picture of the health of the system. Generally, these values should be balanced out and consistent, with the %idle not hovering around zero all the time. If you consistently see large values for %cpu (say greater than 70-80%), be aware that user processes, generally Oracle processes on a Database server, are eating up CPU cycles either spinning on latches or accessing memory pages for LIO (Logical I/O) more frequently than they should. Both indicate that SQL tuning can and should be performed. Large values against %sys (say more than 20-30%) should be investigated for heavy, concentrated file I/O such as backups. Then again, large values in %wio may indicate that the system is actually working hard doing what it is supposed to be doing - although we should cross-check that this is not due to some 'hot' disks.
- -q reports the average CPU queue length and the percentage of time this was occupied. As well, it reports the process queue length for processes that were swapped out and the percentage of time the swap queue was busy. Now the CPU queue statistics are indicate of and supportive of the -u option above. Consistent CPU run queue sizes above the number of CPUs available may indicate that you may benefit from adding CPUs to the system. On the other hand, if the %usr values from -u are high, then it will indicate that SQL tuning needs to be done before you consider throwing more hardware in the mix. This way, you should be able to crosscheck your findings before deciding either way. The Swap Queue is an excellent and foolproof indicator of whether the system is swapping (as opposed to paging). Any positive value under the swpq-sz is an indication of swapping and should trigger an investigation. You should investigate if any parameter - such as large values for SORT_AREA_SIZE, SHARED_POOL_SIZE, DB_BLOCK_BUFFERS - or new databases with large SGAs that consumes memory have been added. If possible, reduce the values of these memory grabbers. If there was no such change, investigate and fix any space leak issues. If after these measures, these values are consistently present and are high, then you may choose to add memory to the server. An example showing sar output for –u and –q along with our interpretation is shown below – we have used figures from our mythical 'mydbserver' that hosts our MYDB database. It is a Solaris 8 based Sun server with 8 CPUs:

```
$ sar -u -f /var/adm/sa/sa30  -- Report CPU statistics from the previous 30th

SunOS mydbserver 5.8 Generic_108528-15 sun4u    01/30/03

00:00:01    %usr    %sys    %wio   %idle
00:20:01      29      11      11      48
00:40:00      26      10       6      58
<snip>
05:40:00      52       8      19      21
06:00:01      70      10      15       5     -- Note the sudden increase in %usr
06:20:01      78      12       9       1     -- from this point on. This was identified
06:40:04      86      13       1       0     -- to be due to a combination of runaway processes and
07:00:04      84      13       2       0     -- a often-running batch program with untuned SQL
07:20:01      69      12      15       4
07:40:01      75      12      12       1
08:00:01      81      14       5       0
08:20:00      77      14       9       0
<snip>
14:40:00      86      13       1       0
15:00:01      85      13       2       0
15:20:00      86      13       1       0
15:40:00      88      11       1       0
<snip>
23:00:00       1       2       0      97     -- Note that Cpu usage has significantly dropped off

Average       67      11       7      15     -- Average for the period

$ sar -q -f /var/adm/sar/sa30         -- Report Run/Swap queue statistics from the same period
```

```
SunOS mydbserver 5.8 Generic_108528-15 sun4u    01/30/03

00:00:01 runq-sz %runocc swpq-sz %swpocc
00:20:01     1.2      12
00:40:00     1.0       7
<snip>
05:40:00     1.9      37               -- Note the run queue size indicated 2 slots (out of the 8 CPUs)
06:00:01     2.7      67               -- continually occupied
06:20:01     4.5      84               -- The %usr CPU was high (from -u) but the queue size was Ok
06:40:04    10.1      99               -- Seems we have reached Cpu saturation (queue size >8)
07:00:04    14.2      98     7.4       97  -- Horrors!! We have started swapping as well!
07:20:01     2.8      58     3.2      100  -- CPU Queue size drops, basically because the processes are
07:40:01     4.1      83     3.0      100  -- waiting to be swapped out or swapped in; as well they don't
08:00:01     7.8      96     3.0      100  -- remain in the CPU queue for long
08:20:00     9.6      89     3.0      100
<snip>
14:40:00     5.7      99     1.0      100
15:00:01     5.4      95     1.0      100
15:20:00     7.2      98     1.0      100
15:40:00     6.1     100     1.0      100
<snip>                                    -- Problem was fixed by shutting down a new Database that came
22:40:00     1.6      40               -- online during the day. SGA sizes of other databases was
23:00:00     1.0       0               -- reduced and runaway processes terminated. Long running SQL
                                       -- from batch jobs were tuned
Average      6.7      64     1.7       46 -- Still high, but looks like Ok now
```

To look further into memory usage, we can then use the –p and –r options. The former reports paging (not swapping) activities while the latter reports free memory and disk swap blocks. We will see an example of just the –r option below as the –p (paging) statistics verified our analysis and conclusions. The swapping was also verified using the –w option.

```
$ sar -r -f /var/adm/sar/sa30        -- Report free memory/swap statistics from the same period

SunOS mydbserver 5.8 Generic_108528-15 sun4u    01/30/03

00:00:01 freemem freeswap
00:20:01  201754 11032764     -- freemem is the average pages (usually 4k) available for user processes
00:40:00  197249 10954564     -- freeswap is the average disk blocks available for paging/swapping
<snip>
02:40:01  168256 10420803
03:00:00  154536 10201938
03:20:01  102044  9311439
03:40:00   47844  8359617     -- we see from this that some processes started consuming memory
04:00:01   22257  7876087     -- from this point onward. This enabled us to trace some runaways
<snip>
06:20:01   23852  3002841
06:40:04   30408  1274760
07:00:04   21387   372592
07:20:01   17037  2966449     -- The SysAdmin notices swap problems and adds some more swap space
07:40:01   24189  7525329     -- And some more again!
```

```
08:00:01   27015  6817265
08:20:00   17622  9839086
08:40:00   18463 23342956     -- And then some more!
09:00:01   27627 22817922
<snip>
19:40:00   74290 31271494
20:00:01  102466 31788681     -- Only at this point does it let up (server restarted!)
20:20:00  131184 32321035
<snip>
23:00:00  554507 42078251

Average    77050 20298454
```

We now look at the –d option. This will help us is looking at individual 'disk' statistics. We emphasize the word disk since the OS sometimes does not really see the physical disks. What it is able to see is a set of addressable units of storage. Depending on your configuration, this may be one actual physical disk in a JBOD (Just-a-Bunch-Of-Disks) configuration, a partition in a larger disk or a LUN (Logical Unit) from a more complex configuration as in a SAN (Storage Area Network). Whatever it may be, your SysAdmin should be able to map this back to a physical device. Using the –d option, you should be able to determine if a particular disk is contributing to the high-points of the I/O waits previous seen in the –u option. We use the term 'disk' loosely here – the OS can report only upto this level of detail, and this may not be a physical disk at all.

Note that unlike the previous sar options, the –d option will display one line per disk per sample, rather than the normal one line per sample. When the number of disks (or LUNS) are very large (and they will be when dealing with SANs) you might be looking at many lines, many of which are mostly zero since the disks they represent were not busy. You can then use the command sequence below to identify only those disks that were at least 'n'% busy – in this case, we have chosen an arbitrary figure of 50%. The %busy is the percentage of time that disk was busy during that period. The avque details the outstanding requests during that time. You could also have detected only those disks that had more than say three requests outstanding on an average using the awk filter – just replace the condition by '($3 > 3)'. The 'r+w/s' is the number of read and write requests per second, and blks/s is the number of blocks transferred. The avwait and avserv columns are the average wait time and service time for the disk in millisecs. It is not unusual to see large %busy but low avwait and avserv times. This is an indication of a busy disk doing what it knows best – serving Read/Write requests. However, watch for disks that exhibit large avque and avwait/avserv times. If your SysAdmin is able to provide the disk to filesystem map, then you should be able to match this to the Statspack File I/O statistics.

```
$ sar -d -f /var/adm/sa/sa30 | awk '{if ($2 > 50) print $0}' – Use awk to process the second column

SunOS mydbserver 5.8 Generic_108528-15 sun4u    01/30/03

00:00:01   device        %busy   avque   r+w/s  blks/s  avwait  avserv
00:20:01   nfs1              0     0.0       0       0     0.0     0.0     -- This is a place-holder
line
           sd217            69     1.0      16    8098     0.0    60.4     -- Note the high figures
           sd217,e          69     1.0      16    8098     0.0    60.4     -- for not only %age busy
           sd561            69     1.0      16    8091     0.0    61.2     -- but also for the avserv
           sd561,e          69     1.0      16    8091     0.0    61.2
00:40:00   nfs1              0     0.0       0       0     0.0     0.0
           sd218            68     0.9      14    7287     0.0    66.2     -- Similar pairs of figures
           sd218,e          68     0.9      14    7287     0.0    66.2     -- indicate that these
           sd562            68     1.0      14    7281     0.0    67.3     -- disks are mirrored
           sd562,e          68     1.0      14    7281     0.0    67.3
01:00:01   nfs1              0     0.0       0       0     0.0     0.0
           sd219            57     0.8      13    6791     0.0    59.5
           sd219,e          57     0.8      13    6791     0.0    59.5
```

```
            sd563            57     0.8      13    6784     0.0    60.4
            sd563,e          57     0.8      13    6784     0.0    60.4
01:20:00    nfs1              0     0.0       0       0     0.0     0.0
            sd220            68     1.0      16    7977     0.0    61.0
            sd220,e          68     1.0      16    7977     0.0    61.0
            sd564            69     1.0      16    7970     0.0    61.8
            sd564,e          69     1.0      16    7970     0.0    61.8
01:40:00    nfs1              0     0.0       0       0     0.0     0.0
02:00:00    nfs1              0     0.0       0       0     0.0     0.0
            sd221            80     1.1      15    7864     0.0    73.0
            sd221,e          80     1.1      15    7864     0.0    73.0
            sd565            80     1.1      15    7860     0.0    74.0
            sd565,e          80     1.1      15    7860     0.0    74.0
02:20:00    nfs1              0     0.0       0       0     0.0     0.0
            sd222            73     1.0      16    8115     0.0    64.1
            sd222,e          73     1.0      16    8115     0.0    64.1
            sd566            73     1.0      16    8108     0.0    65.2
            sd566,e          73     1.0      16    8108     0.0    65.2
```

Other sar options worth monitoring are the -b (disk buffer activity), -c (system call), -g (page outs), -m (message and semaphore activities indicative of Database activity), -v (process/I-node activity - look out for overflows).

Although sar provides a lot of OS performance information, certain other utilities such as 'vmstat' (CPU, Queue and Virtual Memory statistics) and 'iostat' (I/O - terminal, disk and tape - statistics as well as CPU information) provide additional supporting information. Depending on your UNIX variant, the format and columns vary significantly. As well, each variant provides additional tools that you may use only on that platform. For example, Sun provides 'mpstat' that can provide CPU statistics for individual CPUs for a period. These utilities have a major weakness in the sense that they do not automatically store performance information as sar does, nor do they cover the wide range that 'sar' reports on. Sample vmstat and iostat outputs are shown below. Note that the first line (or set of lines) from these commands are the averages since the last UNIX restart.

```
$ iostat 10 4  -- Display 4 IO averages, once every 10 seconds
   tty        sd6           sd11          sd12          sd13           cpu
 tin tout kps tps serv  kps tps serv  kps tps serv  kps tps serv   us sy wt id
   1  300   0   0    0   58   7   21   30   6   24    0   0    5   23  7 13 58 -- Average since UNIX reboot
   0  305   0   0    0   22  11    4   21  11    5    0   0    0   22  6 29 44
   0    8   0   0    0   20   2    8   20   2    9    0   0    0   15  6 29 50
   0  290   0   0    0   57  17   46   45  16   57    0   0    0    9  5 28 58
$ vmstat 10 4  -- Display 4 CPU averages, once every 10 seconds
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s6 s1 s1 s1   in   sy   cs us sy id
 0 1 0 17816360 8204424 371 579 2793 5 5 0 0 0 7  6  0 4079 2536 2776 23  7 70 -- Average since UNIX reboot
 0 0 0 17923568 6197136 16 161 0 1 1  0  0  0  8  8  0 3105 3272 1798  5  3 93
 0 0 0 17922472 6196568 13 157 0 0 0  0  0  0  7  7  0 3167 2941 1734  5  3 92
 0 0 0 17921576 6195616 33 488 0 2 2  0  0  0  2  2  0 3508 4291 2020  8  3 89
```

For determining network-related problems, use the 'netstat' command. The format and option varies among variants (for example, netstat –v in AIX displays packet statistics, but on Solaris would display socket and routing table information!). Whatever it may be, you should look at number of overruns and errors on the various interfaces, as well as number of Open connections and their States.

VARIANTS, VARIANTS

Although UNIX commands and usage is pretty standard across the variants, there are some differences that a DBA will have deal with, especially when it comes to administrative type activities. We will deal with the most common ones, and choose the top three variants - namely Solaris, HP-UX and AIX to detail these variations. Although there are minor differences in Linux variants from various vendors, we will add Linux into this mix as one column.

Task	Solaris	HP-UX	AIX	Linux
Administrative GUI	admintool	sam	smit	linuxconf, yast2 (suse)
User management	useradd, userdel, usermod	useradd, userdel, usermod	lsuser, mkuser, chuser rmuser	useradd, userdel
List hardware configuration	prtconf -v /usr/platform/`uname -i`/ sbin/prtdiag -v (sun4u and sun4d only), psrinfo –v, isainfo –v, dmesg iostat –en, prtfru	ioscan, dmesg, model	prtconf, lscfg, lsattr, lsdev	dmesg /proc/cpuinfo /proc/pci /proc/meminfo /proc/interrupts /proc/ioports lspci hwinfo (suse)
Show and Set Kernel parameters	sysdef getconf cat /etc/system ndd adb -k	sam sysdef kmtune (hp-ux11+) getconf	/usr/samples/kernel/ vmtune (installed with the bos adt.samples fileset) /usr/sbin/no (network-related)	/proc/* /proc/sys/*
Check swap space	swap -s swap –l	swapinfo	lsps -a	cat /proc/meminfo free
Software status	Pkginfo prodreg admintool /var/sadm/install/contents	rmfn, what (hp-ux 9) swlist (hp-ux 10+)	lslpp -l	(redhat) rpm -a –i, rpm -qa (debian) dselect , dpkg –l
Show patch level or patches	Showrev -p prodreg (2.6+) patchadd –p	swlist -l product | grep phsomestring	instfix -ivq oslevel –r	(redhat) rpm -q (debian) dpkg -s
Tracing utility	Truss sotruss	trace (freeware) tusc (11+, freeware)	trace syscalls truss	strace ltrace
OS Error reporting	Prtdiag	dmesg sysdiag (9 and early10) stm/cstm/mstm/xstm (10.20+)	errpt	dmesg (deb) reportbug

Most UNIX vendors also place their Documentation on the Web and links from the main page of the vendor will lead you to these documents.

CONCLUSION

We have detailed the most important points about UNIX that you as a DBA will need to know to be able to work effectively. The information presented above barely scratches the surface of what UNIX has and can do. As mentioned before, we are all learning, and it is never too late to start or build upon what we already know. Books, website and Listserv discussion and archives abound, and most of them are well worth the resources you invest on them.

ABOUT THE AUTHOR

John Kanagaraj is a Principal Consultant with DBSoft Inc., and resides in the Bay Area in sunny California. He has been working with various flavors of UNIX since '84 and with Oracle since '88, mostly as a Developer/DBA/Apps DBA and System Administrator. Prior to joining DBSoft, he led small teams of DBAs and UNIX/NT SysAdmins at Shell Petroleum companies in Brunei and Oman. He started his Troubleshooting career as a member (and later became head) of the Database SWAT/Benchmarking team at Wipro Infotech, India. His specialization is UNIX/Oracle Performance management, Backup/recovery and System Availability and he has put out many fires in these areas along the way since '84! John can be reached via email at 'ora_apps_dba_y@yahoo.com'.

REFERENCES

Web sites:

http://www.bell-labs.com/history/unix – The true history of UNIX

http://bhami.com/rosetta.html – A (UNIX) Sysadmin's UNIXersal Translator

Various Vendor's websites containing UNIX documentation

And of course, the Man pages!