CHAPTER 3

Essential UNIX (and Linux) for the Oracle DBA

If the only thing you needed to learn about were Oracle database administration, your life would be so much easier. However, to ensure that your database performs efficiently, you'll also need to understand the operating system. In this chapter, you'll examine UNIX.

The first part of the chapter covers the most important UNIX/Linux commands for you to know. Most of the UNIX and Linux operating system commands are identical, but I'll show you the differences where they exist. You'll learn about files and directories and how to manage them, as well as UNIX processes and how to monitor them. You'll then learn how to edit files using the vi text editor and how to write shell scripts.

As an Oracle DBA, you'll need to know how to use UNIX services such as the File Transfer Protocol (FTP), which enables you to easily exchange files between computers; telnet, a program that lets you enter commands on a remote computer from a local computer; and the remote login and remote copy services. This chapter provides you with an introduction to these useful features. You'll also learn the key UNIX administrative tools for performing system backups and monitoring system performance. There's also some discussion of the basics of RAID systems and the use of the Logical Volume Manager (LVM) to manage disk systems. Toward the end of the chapter, you'll find some coverage of data storage arrays and new techniques to enhance availability and performance.

Overview of UNIX and Linux Operating Systems

The UNIX and Linux operating systems are similar in many ways, and users can transition easily from one to the other. From the DBA's point of view, there are few differences in commands and utilities when you migrate from one operating system to the other, since they all share common roots.

UNIX

UNIX became the leading operating system for commercial enterprises during the 1980s and 1990s. Although IBM mainframes still perform well for extremely large (multiterabyte) databases, most medium to large firms have moved to UNIX for its economy, versatility, power, and stability.

UNIX has a rich history, progressing through several versions before reaching its current popular place in the operating system market. I could spend quite a bit of time discussing the history and variants of the UNIX system, but I'll simplify the discussion by stating that, in reality, the particular UNIX system variant that a DBA uses doesn't make much difference. UNIX has become well known as a multitasking, multiuser system and it is currently the most popular platform for major Oracle implementations. The most popular UNIX flavors on the market as of this writing are Sun Solaris, HP-UX, and the IBM AIX versions. The basic commands don't vary much between the UNIX variants, and the different flavors mainly distinguish themselves on the basis of the utilities that come packaged with them.

Contrary to what newcomers to the field might imagine, UNIX is an easy operating system to learn and use. What might put off many developers and others who were weaned on the graphical Windows framework are the terse and cryptic commands commonly associated with the UNIX operating system. Take heart, though, in the knowledge that the essential commands are limited in number, and you can become proficient in a very short time.

Sun Microsystems (Sun), Hewlett-Packard (HP), and IBM sell the leading UNIX servers—the machines that run each firm's variation of the Berkeley UNIX system V. IBM is also a big UNIX supplier with its AIX server. Sun and HP currently run the vast majority of UNIX-based Oracle installations.

Linux

Developed by Linus Torvalds, Linux is constantly under development because it is released under an open source license and is freely available for download from the Internet. Many users prefer to use Linux because more programs and drivers are available, it's free (or close to free, as the commercial versions are fairly cheap), and bug fixes are released very quickly. A version of Oracle Database 10*g* for Linux is available for download on the OTN web site. Oracle has certified and supports Red Hat Enterprise Linux AS and ES (either the 3.0 or the 2.1 version), SUSE LINUX Enterprise Server, and Asianux 1.0. Oracle will also continue to provide customer support for UnitedLinux 1.0 throughout its life cycle for existing Oracle products.

■ **Note** I used a Linux 3.0 distribution from Red Hat to run Oracle Database 10*g* on my Windows XP desktop. I used the VMware virtual operating system tool (http://www.vmware.com) to run the Linux operating system alongside Windows.

Oracle was the first company to offer a commercially available database for the Linux operating system. Oracle even offers a cluster file system for Linux, which makes it possible to use Oracle's Real Application Clusters (RAC) on Linux without the more costly and complex raw file systems.

Do all these moves toward the Linux operating system foreshadow the demise of the UNIX operating system? Although the market for UNIX systems has dropped in recent years, you have to interpret this fact cautiously; most of the movement toward the Linux operating system is intended for low-end machines that serve network and other desktop applications. For the foreseeable future, UNIX-based systems will continue to rule the roost when it comes to large, company-wide servers that run large and complex databases such as Oracle Database 10*g*.

IT organizations are moving to Linux and open source software to solve a wide variety of business problems. The Linux platform often plays the central role in establishing a low-cost computing infrastructure. Oracle's grid initiative relies on using massive numbers of cheap commodity servers based on the Linux platform. Although Linux is growing very fast as a viable operating system for Oracle databases, the consensus among the IT industry is still that Linux is mainly useful for services, and not for mission-critical databases. This leaves UNIX and Windows as the two leading operating systems for Oracle databases. Oracle provides support to the Linux community by offering code for key products and itself uses the Linux platform extensively. Oracle's clustered file systems link a number of separate servers into a single system and low-cost Linux servers are an inexpensive choice for these file systems.

Midrange Systems

Even just four or five years ago, you had to invest in behemoths like the Sun E10K, with its hard partitions and multiple processors, if you wanted a system to support heavy workloads. Today, much smaller midrange UNIX servers come with features like soft partitioning, high amounts of memory, hot-spare processors, and capacity-on-demand features that were once the exclusive preserve of the high-end systems.

The main competition among the midrange servers is between Intel-based servers like the Windows Server 2003 and RISC-based (reduced instructor set computer) servers using the UNIX or the Linux operating systems. The choice of the particular operating system will depend on the workload you plan on supporting as well as on the availability, reliability, and response time requirements.

The rest of the chapter, while formally oriented toward UNIX-based systems, applies almost verbatim to any Linux-based operating system as well.

Understanding the UNIX Shell(s)

In UNIX systems, any commands you issue to the operating system are passed through a command interpreter layer around the kernel called the *shell*. When you initially log in, you are communicating with this shell. The *kernel* is the part of UNIX that actually interacts with the hardware to complete tasks such as writing data to disk or printing to a printer. The shell translates your simple commands into a form the kernel can understand and returns the results to you. Therefore, any commands you issue as a user are *shell commands*, and any scripts (small programs of grouped commands) that you write are *shell scripts*.

The UNIX shell has many variants, but they are fundamentally the same, and you can easily migrate from one to another. Here's a list of the main UNIX and Linux shell commands and the shells they run:

- `sh`: The Bourne shell, which was written by Steven Bourne. It is the original UNIX shell, and is quite simple in the range of its features.
- `csh`: The C shell, which uses syntax somewhat similar to the C programming language. It contains advanced job control, aliasing, and file-naming features.
- `ksh`: The Korn shell, which is considered a superset of the Bourne shell. It adds several sophisticated capabilities to the basic Bourne shell.
- `bash`: The "Bourne Again Shell," which includes features of both the Bourne and the C shell.

For the sake of consistency, I use the Korn shell throughout this book, although I show a couple of important C shell variations. Most UNIX systems can run several shells; that is, you can choose to run your session or your programs in a particular shell, and you can easily switch among the shells.

The Linux default shell is BASH, the Bourne Again Shell, which includes features of the Bourne shell as well as the Korn, C, and TCSH shells.

■ **Note** Most of the basic commands I discuss in the following sections are the same in all the shells, but some commands may not work, or may work differently, in different shells. You need to remember this when you switch among shells.

Shells act as both command interpreters and high-level UNIX programming languages. As a command interpreter, the Korn shell processes interactive user commands; as a programming language, the Korn shell processes commands in shell scripts.

It is possible to invoke any available shell from within another shell. To start a new shell, you simply type the name of the shell you want to run, `ksh`, `csh`, or `sh`. It is also possible to set the default startup shell for all your future sessions. The default shell for your account is stored in the system database /etc/passwd, along with the other information about your account. To change your default shell, use the `chsh` command.

Accessing the UNIX System

You can manage the Oracle databases that run on UNIX systems in several ways:

- Directly from the server hosting the database
- Via a UNIX workstation
- Through a Windows NT Server front end

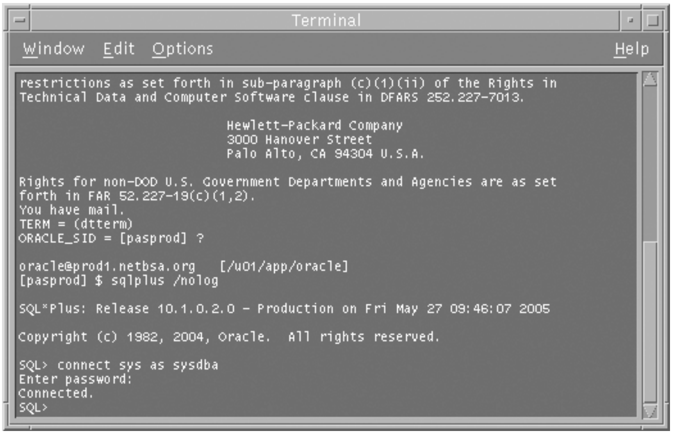
Most DBAs use the last approach, preferring to use their regular PCs to manage their databases. If that's what you choose, you again have several choices as to how exactly you interact with the databases running on the remote server:

- Log directly into the server through the telnet service.
- Log into the server through a display framework such as Reflections X-Client, which provides an X Window System that emulates the look and feel of a UNIX workstation.
- Connect through a GUI-based management console, such as the Oracle-supplied Oracle Enterprise Manager (OEM) or through a tool from a third-party supplier, such as BMC Software (http://www.bmc.com/) or Quest Software (http://www.quest.com/).

Regardless of whether you choose to log into the UNIX box through the server or another interface, the first thing you will need is an account and the appropriate privileges to enable you to log in and actually get something done. The UNIX system administrator, with whom you should become very friendly, is the person who will perform this task and give you your password. The system administrator will also assign you a *home directory,* which is where you will land inside the UNIX file system when you initially log in.

You can log into a UNIX machine in several ways. You can always log into the server directly by using the terminal attached to the machine itself. However, this is not a commonly used option for day-to-day work. You can also use telnet to connect to the UNIX server, and you'll learn about this in the "Using Telnet" section later in this chapter. One of the most common ways to work with UNIX, though, is through your own PC by using what's called a *terminal emulator*—a program that will enable your PC to mimic a UNIX terminal. Several vendors, including Hummingbird (http://www.hummingbird.com/) and WRQ (http://www.wrq.com/), produce the popular Hummingbird and Reflections emulators, respectively. These emulators, also called X Window emulators, emulate the X Window System, which is the standard graphical user interface (GUI) for UNIX systems. The emulators use special display protocols that will let you use your Windows terminal as an X terminal to access a UNIX server.

The general idea behind many of these interfaces is to try and make working with UNIX as easy as possible by providing a familiar GUI. Figure 3-1 shows a basic X session connected to the UNIX operating system.



```
restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause in DFARS 252.227-7013.

                        Hewlett-Packard Company
                        3000 Hanover Street
                        Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set
forth in FAR 52.227-19(c)(1,2).
You have mail.
TERM = (dtterm)
ORACLE_SID = [pasprod] ?

oracle@prod1.netbsa.org   [/u01/app/oracle]
[pasprod] $ sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Fri May 27 09:46:07 2005

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect sys as sysdba
Enter password:
Connected.
SQL>
```

Figure 3-1. *An X session*

For now, let's assume you are equipped with a terminal emulator. You need to know a couple of things before you can log in and use the system. First, you need to know the machine name, which can be in either symbolic or numerical form.

■**Note** All UNIX machines (also called also called UNIX *boxes* or UNIX *servers*) have an Internet Protocol (IP) address, usually in a form like this: 162.15.155.17. Each IP address is guaranteed to be unique. By using a special system file (/etc/hosts), the UNIX administrator can give what's called a *symbolic name* to the machine. For example, the machine with the IP address 162.15.155.17 can be called prod1, for simplicity. In this case, you can connect by using either the IP address or the symbolic name.

Next, the system will ask you for your password. A shell prompt indicates a successful login, as shown here:

```
$
```

The shell prompt will be a dollar sign ($) if you are using the Bourne shell or the Korn shell. The C shell uses the percent sign (%) as its command prompt.

Once you log into the system, you are said to be working in a UNIX *session*; you are automatically working in what's known as your home directory (more on this later on). You type your commands at the shell prompt, and the shell interprets these commands and hands them over to the underlying operating system.

The UNIX directory structure is hierarchical, starting with the root directory at the top, which is owned by the UNIX system administrator. From the root directory, the other directories branch out and the files are underneath them. Let's say you are in the /u01/app/oracle directory when you log in, and you want to refer to or execute a program file located in the directory /u01/app/oracle/admin/dba/script. To specify this location in the hierarchy to the UNIX system, you must give it a *path.* If you want, you can give the complete path from the root directory: /u01/app/oracle/admin/dba/script. This is called the *absolute path,* because it starts with the root directory itself. You can also specify a *relative path,* which is a path that starts from your current location. In this example, the relative path for the file you need is admin/dba/script.

Note Included among these directories and files are the system files, which are static, and user files. As a DBA, your main concern will be the Oracle software files and database files.

You end your UNIX or Linux session by typing the word **exit** at the prompt, as follows:

```
$ exit
```

Overview of Basic UNIX Commands

You can execute hundreds of commands at the command prompt. Don't get overwhelmed just yet, though: of the many commands available to you, you'll find that you'll only use a handful on a day-to-day basis. This section covers the basic commands you'll need to operate in the UNIX environment.

Note If you need help using a command, you can type **man** at the command prompt, along with the name of the topic you're trying to get help with. For example, if you type in the expression **man date**, you'll receive information about the `date` command, examples of its use, and a lot of other good stuff. For more details, see the "Help and Info: The man Command" section later in this chapter.

The UNIX shell has a few simple, built-in commands. The other commands are all in the form of executable files that are stored in a special directory called *bin* (short for "binary"). Table 3-1 presents some of the more important UNIX commands that you'll need to know. The UNIX commands tend to be cryptic, but some should be familiar to Windows users. The commands `cd` and `mkdir` in Windows, for example, have the same meaning in UNIX. Many UNIX commands have additional options or switches (just like their MS-DOS counterparts) that extend the basic functionality of the command, and Table 3-1 shows the most useful command switches.

Table 3-1. *Basic UNIX Commands*

Command	Description	Example

`cd`	The `cd` command enables you to change directories. The format is `cd new-location`. The example shown here takes you to the directory /tmp directory, from your current working directory.	`$ cd /tmp`
		`$`

`date`	The `date` command gives you the time and date.	`$ date`
		Sat Mar 26 16:08:54 CST 2005
		`$`

`echo`	With the `echo` command, you can display text on your screen.	`$ echo Buenos Dias`
		`Buenos Dias`
		`$`

Command	Description	Example

`grep`	The `grep` command is a pattern-recognition command. It enables you to see if a certain word or set of words occurs in a file or the output of any other command. In the example shown here, the `grep` command is checking whether the word "alapati" occurs anywhere in the file test.txt. (The answer is yes.) The `grep` command is very useful when you need to search large file structures to see if they contain specific information. If the grepped word or words aren't in the file, you'll simply get the UNIX prompt back, as shown in the second example.	`$ grep alapati test.txt`
		`alapati`

`history`	The `history` command gives you the commands entered previously by you or other users. To see the last three commands, type **history -3**. The default number of commands shown depends on the specific operating system, but it is usually between 15 and 20. Each command is preceded in the output by a number, indicating how far back it was used.	`$ history -3`
		`4       vi trig.txt`
		`5       grep alapati test.txt`
		`6       date`
		`7       history -3`
		`[pasx] $`

`passwd`	When you are first assigned an account, you'll get a username and password combination. You are free to change your password by using the `passwd` command.	`$ passwd`
		`Changing password for salapati`
		`Old password:`
		`New password:`

`pwd`	Use the `pwd` command to find out your present working directory or to simply confirm your current location in the file system.	`$ pwd`
		`/u01/app/oracle`
		`$`

`uname`	In the example shown here, the `uname` command tells you that the machine's symbolic name is prod5 and it's an HP-UX machine. The `-a` option tells UNIX to give all the details of the system. If you omit the `-a` option, UNIX will just respond with HP-UX.	`$ uname  -a`
		`HP-UX  prod5    B.11.00  A    9000/800`
		`190    two-user license`
		`$`

`whereis`	As the name of this command suggests, `whereis` will give you the exact location of the executable file for the utility in question.	`$ whereis who`
		`who:  /usr/bin/who`
		`/usr/share/man/man1.z/who.1`
		`$`

`which`	The `which` command enables you to find out which version (of possibly multiple versions) of a command the shell is using. You should run this command when you run a common command, such as `cat`, and receive somewhat different results than you expect. The `which` command helps you verify whether you are indeed using the correct version of the command.	`$ which cat`
		`/usr/bin/cat`

Continued

Table 3-1. *Continued*

Command	Description	Example

`who`	If you are curious about who else besides you is slogging away on the system, you can find out with the `who` command. This command provides you with a list of all the users currently logged into the system.	`$ who`
		`salapati    pts/0    Nov 8  08:31`
		`rhudson     pts/1    Nov 8  09:04`
		`lthomas     pts/3    Nov 9  15:54`
		`dcampbel    pts/7    Nov 8  16:27`
		`dfarrell    pts/16   Nov 5  07:00`

`whoami`	The `whoami` command indicates who you are logged in as. This may seem trivial, but as a DBA, there will be times when you could be logged into the system using any one of several usernames. It's good to know who exactly you are at a given point in time, in order to prevent the execution of commands that may not be appropriate, such as deleting files or directories. The example shown here indicates that you are logged in as user Oracle, who is the owner of Oracle software running on the UNIX system.	`$ whoami`
		`oracle`
		`$`

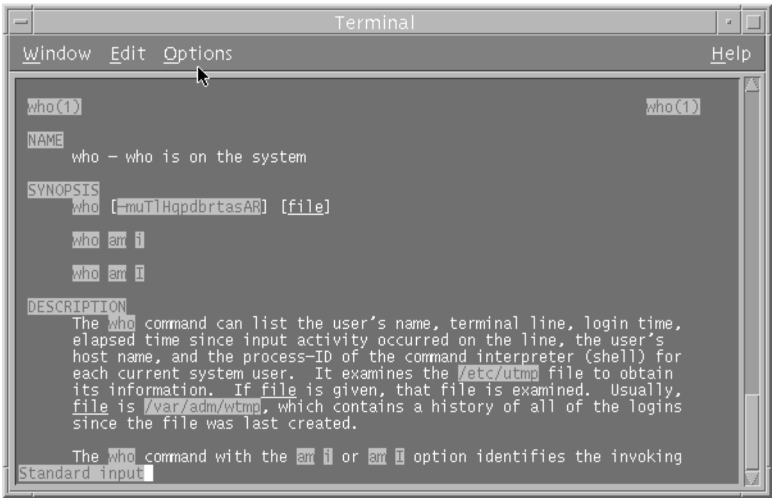
■ **Tip** It is always worthwhile to check that you are at the right place in the file structure before you press the Enter key, to avoid running any destructive commands. The following commands will help you control your input at the command line. Under the Korn shell, to retrieve the previous command all you have to do is press the Esc key followed by the letter **k**. If you want an older command, continue typing the letter **k**, and you'll keep going back in the command sequence. If you have typed a long sequence of commands and wish to edit it, press the Esc key followed by the letter **h** to go back, or press the letter **l** to go forward on the typed command line.

Help and Info: The man Command

There are many operating system commands, most with several options. Therefore, it's convenient to have a sort of help system embedded right within the operating system so you have the necessary information at your fingertips. UNIX and Linux systems both come with a built-in feature called the *man pages,* which provide copious information about all the operating system commands. You can look up any command in more detail by typing the `man` command followed by the command you want information on, as follows:

```
$ man who
```

This command will then display a great deal of information about the `who` command and all its options, as well as several examples (see Figure 3-2).



```
who(1)                                                     who(1)

NAME
     who - who is on the system

SYNOPSIS
     who [-muTlHqpdbrtasAR] [file]

     who am i

     who am I

DESCRIPTION
     The who command can list the user's name, terminal line, login time,
     elapsed time since input activity occurred on the line, the user's
     host name, and the process-ID of the command interpreter (shell) for
     each current system user.  It examines the /etc/utmp file to obtain
     its information.  If file is given, that file is examined.  Usually,
     file is /var/adm/wtmp, which contains a history of all of the logins
     since the file was last created.

     The who command with the am i or am I option identifies the invoking
Standard input
```

Figure 3-2. *Output of the man command*

In Linux-based systems, you can also use the nifty `whatis` command to find out what a certain command does. Like the `man` command, the `whatis` command is followed by the name of the command you want information about. Here's a simple example:

```
$ whatis whereis (1) -locate the binary, source, and manual page files
for a command
```

As you can see, the `whatis` command offers a quicker and easier way to locate summary information about any command than the more elaborate `man` pages.

Changing the Prompt

Every shell has its own default prompt. The default prompt for the Korn shell is the dollar sign (`$`). You can easily change it to something else by changing the value of the PS1 shell variable.

In the following example, I first check the value of the `PS1` variable by issuing the command `echo $PS1`. I then use the `export` command to set the value of the `ORACLE_SID` environment variable to my database name, *finance*. Using the `export` command again, I set the value of the `PS1` environment variable to be the same as the value of the environment variable `ORACLE_SID` (`$ORACLE_SID`). Now the shell prompt is changed to my database name, finance. Since I only exported the `ORACLE_SID` variable value but didn't place it in my environment files, the value I exported is good only for the duration of the current session.

```
$ echo $PS1
$
$ export ORACLE_SID=finance
$ export PS1=[$ORACLE_SID]
[finance]
```

Note If you add the `PS1` variable to your .cshrc file (I explain how to do this later in the "Customizing Your Environment" section), every time you open a new shell, it'll have your customized prompt. The ability to change the prompt is useful if you're managing many different databases via UNIX. You can amend the prompt to reflect the database you're working on at any given time. For example, when you're working in an inventory system, the prompt can display `invent>`. That way, you won't accidentally execute a command in the wrong database.

Finding Files and Directories

Sometimes you want to locate a file, but you aren't sure where it might be located in the file system. The `whereis` command, of course, is of help only if you are locating commands, not files. To find out where a file or a directory is, you can use the `find` command, as shown here:

```
$ pwd
/u01/app/oracle
$ find . -name bill.sql -print
./dba/bill.sql
$
```

In this example, the `find` command informs you that the bill.sql file is located in the /u01/app/oracle/dba directory. Note that there is a dot after the `find` keyword, indicating that a recursive search is made from the present directory—every directory and subdirectory under the present directory will be searched. If you want to search from a specific directory, you need to specify that in the command. In the following example, the `find` command starts its search from the root (`/`) file system and prints the location of the test.txt file to the screen, if it finds it:

```
$ find / -name test.txt  -print
```

Controlling the Output of Commands

Sometimes a command will produce more output than can fit on the screen. You can control the output of a command in a couple of ways.

The `more` command will show you the contents of a file, one screen at a time. Just press Enter to see the next screen of the file:

```
$ more test.txt
```

The pipe command (`|`) enables you to pass the output of one command as input to another command. In the following example, the | operator takes the `ps -ef` command's output (which is the list of all processes that are currently running on your system) and passes it to the `grep` command as a list, to search for all processes that contain the word "Oracle":

```
$ ps -ef | grep Oracle
```

This example also demonstrates the use of multiple commands at once.

Showing the Contents of Files

As you know, you can use the vi editor to read a file as well as write to it. However, in some cases you may want to just read the contents of a file. The `cat` command lets you do so, as shown here:

```
$ cat  test.ksh
#!/bin/ksh
VAR1=1
```

```
while  [ $VAR1 -lt 100 ]
do
     echo "value of VAR1 is : $VAR1"
     ((VAR1=VAR1+1))
done
$
```

■**Note** You can also use the `page` command to peruse files.

Different or Same Files?

The `diff` command compares two files, returns the line(s) that are different, and tells you how to make the files the same. Here's an example:

```
$ diff test.one test.two
0a1
> New Test.
```

This `diff` command output tells you that if you add the line "New Test" to the test.one file, you can make it identical to the test.two file. The first character, "0," is the line number to edit in test.one; the "a" indicates that the line should be added to test.one to match the first line, "1," of test.two.

UNIX Variables

There are two main types of variables in a UNIX or Linux system: user-created variables and shell variables. Let's briefly look at how you use both kinds of variables.

User-Created Variables

A user can create a variable and initialize it by providing a value for it. The variable name must consist of letters and numbers, and it must start with a letter. You can also use the `export` command to export variables, so that any shell you create in your current session can make use of your variables.

Here's an example of a user-created variable (note how echoing the variable itself prints just the variable, not its value—to show the variable's value, you must precede the variable's name with the `$` sign in your `echo` command):

```
$ database=nicko
$ echo database
database
$ echo $database
nicko
$
```

In this example, I first created a new variable called database and assigned it the value of "nicko". I then used the `echo` command to print the value of the database variable, and the `echo` command just prints the string "database". The second time I used the `echo` command, I added the dollar sign ($) in front of the name of the variable ($database). When I did this, the value of the variable database was shown as "nicko".

To remove the value of the database variable, simply set it to null, as shown here:

```
$ database=
$ echo $database
$
```

Shell Variables

Shell variables are variables whose values are set by the shell itself, instead of by a user. Shell variables are also called *keyword variables*, since short keywords are used to represent some of these variables. When you first log into a UNIX system, you must make several bits of information available to the shell, such as the name of your home directory, the type of editor you prefer to use for editing text, and the type of prompt you want the system to display while your session is active. Each of these is determined by values assigned to shell variables. These are some common shell variables:

- HOME: Identifies a user's home directory.
- PATH: Specifies the directories in which the shell should look when it tries to execute any command. It's common to include both the binary (bin) directories for UNIX and Oracle software as part of the PATH variable.

Fortunately, you don't have to manually set up the environment every time you log into the system. There is a file, named .profile or .login, depending on the type of UNIX shell you are using, that automatically sets the environment variables for all users at login time. When you log in, the shell will look in the appropriate file and establish the environment by setting the values of all shell variables.

Using the export and setenv Commands

Both user-defined and shell variables are local to the process that declares them first. If you want these variables to be accessible to a shell script that you want to execute from your login shell, you need to explicitly make the variables available to the calling environment of the child process.

You can make a variable's value available to child processes by using the `export` command in the Korn and BASH shells. In the C shell, you use the `setenv` command to do the same thing. Here's an example that shows how to use the `export` command to make the value of a variable available to a child process:

```
$ export ORACLE_HOME =/u03/app/oracle/product/10.2.0/orcl
```

The following sequence would achieve the same results as the preceding `export` command:

```
$ ORACLE_HOME =/u03/app/oracle/product/10.2.0/orcl
$ export ORACLE_HOME
```

In the C shell, you use the `setenv` command to set a variable's value, as shown here:

```
$ setenv ORACLE_HOME= /u03/app/oracle/product/10.2.0/orcl
```

■**Note** UNIX programs and commands can be run in two entirely different ways: *interactive mode* is when you log in and type your commands directly to the screen; *batch mode* is when you run your commands or an entire program at once, usually by using executable shell scripts in the form of UNIX text files.

Displaying the Environment

Type **env** at the system prompt, and your entire set of environment variables will scroll by on the screen. Here's an example:

```
$ env
 PATH=/usr/bin:/usr/ccs/bin:/user/config/bin
 ORACLE_PATH=/u01/app/oracle/admin/dba/sql
 ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
 ORACLE_SID=prod1
 TNS_ADMIN=/u01/app/oracle/product/network
 TERM=vt100
$
```

To see the value of one specific environment variable, rather than the entire set (which can be a fairly long list in a real-world production system), you can ask the shell to print the variable's value to the screen by using the `echo` command:

```
$ echo $ORACLE_HOME
 /u01/app/oracle/product/10.2.0.0.0/db_1
$
```

Note that in the `echo` command, the `$` precedes the environment name so that the command will print the value of the variable, not the name of the variable itself.

Customizing Your Environment

Both the Bourne shell and the Korn shell use the .profile file to set the values for all shell variables. The .profile file executes when you first log in to the UNIX or Linux system.

The C shell executes the .cshrc file every time you invoke a new C shell. The .cshrc file is a short file with generic C shell commands that should work with any flavor of UNIX with only minor modifications. This means that you could have essentially the same .cshrc file on all UNIX systems you use. Your .cshrc file is executed whenever you open a terminal window in a UNIX or Linux environment, or when you execute a script. You can add commands in the .cshrc file (using a text editor like vi) that will make your work in UNIX more productive. The C shell also executes the contents of the .login file when you log in and start a new session. The .login file is located a user's home directory; for example, `/home/oracle` for the Oracle user on most UNIX systems.

Here's a list of the various scripts executed under each of the main UNIX and Linux shells, to set the shell's environment:

- Bourne shell (`sh`): Only the .profile file is executed when a user logs in. The .profile file is located in the user's home directory.
- C shell (`cshrc`): The shell executes the .login file after it first executes the .cshrc file. When you create a new shell after logging in, the .cshrc script is executed, but not the .login file.
- Korn shell (`ksh`): The .profile file in your home directory is executed.
- BASH shell (`bash`): The .bash_profile is executed at login time, and the .bashrc file is executed when you start a new shell.

To change an environment variable permanently, you can edit the .profile or .login file and insert the necessary values for a variable. For example, for the .login file you would add a line like this:

```
setenv VARIABLENAME value_of_variable
```

For the .profile file, you could add lines like the following:

```
VARIABLE=value_of_variable
EXPORT VARIABLE
```

The changes will come into effect the next time you log in or invoke an instance of the C shell. You can change your environment immediately in the Bourne and Korn shells in order to effect immediate environmental changes, by using the following command:

```
$ . .profile
```

Similarly, you can use the `source` command in the C shell, to put the environment variable changes into immediate effect:

```
$ source .cshrc
```

Input and Output Redirection in UNIX

When using a UNIX window on your PC or a UNIX workstation, the keyboard is the standard way to input a command to the shell, and the terminal is the standard location for the output of the commands. Any resulting errors are called *standard errors* and are usually displayed on the screen.

■**Note** It's common to use the terms *standard input, standard output,* and *standard error* to refer to the standard input and output locations in the UNIX shell.

However, you can also use a previously written file as input, or you can have UNIX send output to a file instead of the screen. This process of routing your input and output through files is called input and output redirection.

You can redirect output to a special location called /dev/null when you want to get rid of the output. When you use /dev/null as the output location, all messages issued during the execution of a program are simply discarded and not recorded anywhere on the file system. The following example shows how redirecting a file's output to /dev/null make its contents disappear.

```
$ cat testfile1
$ This is the first line of testfile1
$ cat testfile1 > /dev/null
$ cat /dev/null
```

In this example, the first `cat` command shows you the output of testfile1. However, after redirecting the `cat` command's output to /dev/null, the output of the `cat` command disappears.

■**Note** Redirecting the output of the `cat` command tends to defeat the purpose of running the command in the first place, but there will be other situations, such as when running a script, when you don't want to see the output of all the commands.

Table 3-2 summarizes the key redirection operators in most versions of UNIX.

Table 3-2. *Input/Output Redirection in UNIX*

Redirection Operator	Description
`<`	Redirects standard input to a command
`>`	Redirects standard output to a file
`>>`	Appends standard output to a file
`<<`	Appends standard input to a file
`2 >`	Redirects standard error

In the following example, the `date` command's output is stored in file1, and file2 in turn gets the output of file1:

```
$ date > file1
$ file1 < file2
```

You can achieve the same result with the use of the UNIX pipe (`|`):

```
$ date | file2
```

The pipe command, which uses the pipe symbol (`|`), indicates that the shell takes the output of the command *before* the | symbol and makes it the input for the command *after* the | symbol.

The noclobber Shell Variable

You can use the `noclobber` shell variable to avoid accidentally overwriting an existing file when you redirect output to a file. It's a good idea to include this variable in your shell start-up file, such as the .cshrc file, as shown here:

```
set noclobber
```

Navigating Files and Directories in UNIX

As you might have inferred, files and directories in UNIX are pretty much the same as in the Windows system. In this section, you'll learn all about the UNIX file system and directory structure, and you'll learn about the important UNIX directories. You'll also learn some important file-handling commands.

Files in the UNIX System

Files are the basic data storage unit on most computer systems, used to store user lists, shell scripts, and so on. Everything in UNIX/Linux, including hardware devices, is treated as a file. The UNIX file system is hierarchical, with the root directory, denoted by a forward slash (`/`), as the starting point at the top.

■Tip In Oracle, everything is in a *table* somewhere; in UNIX, everything is in a *file* somewhere.

Files in a typical UNIX system can be one of the following three types:

- *Ordinary files*: These files can contain text, data, or programs. A file cannot contain another file.
- *Directories*: Directories contain files. Directories can also contain other directories because of the UNIX tree directory structure.
- *Special files*: These files are not used by ordinary users to input their data or text; rather, they are for the use of input/output devices, such as printers and terminals. The special files are called *character* special files if they contain streams of characters, and they are called *block* special files if they work with large blocks of data.

Linking Files

You can use the `link` command to create a *pointer* to an existing file. When you do this, you aren't actually creating a new file as such; you are creating a virtual copy of the original by pointing a new filename to an existing file. You use symbolic links when you want to conveniently refer to files from a different directory, without having to provide their complete path. There are two types of links: hard links and symbolic links. You can create *hard links* between files in the same directory, whereas you can create *symbolic links* for any file residing in any directory. The previous example shows a symbolic link. A hard link is usually employed to make a copy of a file, while a symbolic link merely points to another file (or directory). When you manage Oracle databases, you often create symbolic links for parameter files, so you can refer to them easily, without having to specify its complete path.

You use the following syntax when creating a symbolic link:

```
$ ln -s <current_filename> <link_name>
```

The following command creates a symbolic link called test.sql, which refers to the original file called monitor.sql:

```
$ ln -s /u01/app/oracle/admin/dba/sql/monitor.sql      /u01/app/oracle/test.sql
```

Once the test.sql symbolic link is created, the status of the new file can be checked from the /u01/app/oracle directory, as shown here:

```
$ cd /u01/app/oracle
$ ls -altr test.sql
lrwxr-xr-x   1 oracle     dba             41 Mar 30 10:13 test.sql ->
/u01/app/oracle/admin/dba/sql/monitor.sql
$
```

Managing Files

You can list files in a directory with the `ls` command. The command `ls -al` provides a long listing of all the files, with permissions and other information. The command `ls -altr` gives you an ordered list of all the files, with the newest or most recently edited files at the bottom. Here are some examples:

```
$ ls
catalog.dbf1     tokill.ksh    consumer
$ ll
total 204818
-rw-rw-r---  1 oracle  dba  104867572  Nov 19  13:23  catalog.dbf1
-rw-r------  1 oracle  dba        279  Jan   04  1999    tokill.ksh
drwrxr-xr-x  1 oracle  dba        1024 Sep  17   11:29     consumer
$ ls -altr
-rw-r------  1 oracle dba  279       Jan 04    1999     tokill.ksh
drwrxr-xr-x  1 oracle dba  1024      Sep 17    11:29     consumer
-rw-rw-r---  1 oracle dba  104867572  Nov 19    13:23    catalog.dbf1
$
```

You can view the contents of a file by using the `cat` command, as shown in the following code snippet. Later on, you'll learn how to use the vi editor to view and modify files.

```
$ cat test.txt
This is a test file.
This file shows how to use the cat command.
Bye!
$
```

But what if the file you want to view is very large? The contents would fly by on the screen in an instant. You can use the `more` command to see the contents of a long file, one page at a time. To advance to the next page, simply press the spacebar.

```
$ cat abc.txt | more
```

You can copy a file to a different location by using the `cp` command. Note that the `cp` command, when used with the `-I` option, will prompt you before it overwrites a previously existing file of the same name.

```
$ pwd
$ /u10/oradata
$ cp test.txt /u09/app/oracle/data
$ cp -i sqlnet.log output.txt
overwrite output.txt? (y/n) y
```

The `mv` command enables you to move the original file to a different location, change the file's name, or both. The following example uses the `mv` command to change the name of the test.txt file to abc.txt:

```
$ ls
$ test.txt
$ mv test.txt abc.txt
$ ls
abc.txt
```

If you want to get rid of a file for whatever reason, you can use the `rm` command. Watch out, though—the `rm` command will completely delete a file. To stay on the safe side, you may want to use the `rm` command with the `-i` option, which gives you a warning before the file is permanently obliterated. Be careful with the `rm` command, as it's easy to inadvertently remove your entire file system with it!

```
$ ls
abc.txt  careful.txt  catalog.txt  sysinfo.txt
$ rm abc.txt
$ rm -i careful.txt
careful.txt: ? (y/n) y
$ ls
$ catalog.txt   sysinfo.txt
```

Permissions: Reading from or Writing to Files in UNIX

A user's ability to read from or write to files on a UNIX system depends on the permissions that have been granted for that file by the owner of the file or directory—the user who creates a file is the owner of that file.

Every file and directory comes with three types of permissions:

- *Read*: Lets you view the contents of the file only.
- *Write*: Lets you change the contents of the file. Write permission on a directory will let you create, modify, or delete files in that directory.
- *Execute*: Lets you execute (run) the file if the file contains an executable program (script).

Read permission is the most basic permission. Having the execute permission without the read permission is of no use—you can't execute a file if you can't read it in the first place.

Determining File Permissions

Use the `ls -al` command to list the file permissions along with the filenames in a directory. For example, look at the (partial) output of the following command:

```
$ ls -al
-rwxrwxrwx  1 oracle     dba  320   Jan 23     09:00     test.ksh
-rw-r---r-  1  oracle  dba   152    Jul  18     13:38     updown.ksh
-rw-r---r-  1  oracle  dba    70     Nov 22    01:30     tokill.ksh
$
```

You'll notice that at the beginning of each line, each file has a combination of ten different letters and the blank sign (`-`).

The first letter could be a blank or the letter `d`. If it is the letter `d`, then it's a directory. If it's a blank, it's a regular file.

The next nine spaces are grouped into three sets of the letters `rwx`. The `rwx` group refers to the read, write, and execute permissions on that file. The first set of `rwx` indicates the permissions assigned to the owner of the file. The second set lists the permissions assigned to the group the user belongs to. The last set lists the permissions on that file granted to all the other users of the system.

For example, consider the access permissions on the following file:

```
$ -rwxr-x--x 1 oracle dba Nov 11 2001 test.ksh
```

Because the first character is a hyphen (`-`), this is a file, not a directory. The next three characters, `rwx`, indicate that the owner of the file test.ksh has all three permissions (read, write, and execute) on the file. The next three characters, `r-x`, show that all the users who are in the same group as the owner have read and execute permissions, but not write permissions. In other words, they cannot change the contents of the file. The last set of characters, `--x`, indicates that all other users on the system can execute the file, but they cannot modify it.

Setting and Modifying File Permissions

Any file that you create will first have the permissions set to `-rw-r--r--`. That is, everybody has read permissions, and no user has permission to execute the file. If you put an executable program inside the file, you'll want to grant someone permission to execute the file. You can set the permissions on the file by using the `chmod` command in one of two ways.

First, you can use the *symbolic notation,* with the letter `o` standing for owner, `g` for group, and `u` for other users on the system. You grant a group or users specific permissions by first specifying the entity along with a plus sign (`+`) followed by the appropriate symbol for the permission. In the following example, the notation `go+x` means that both the group and others are assigned the execute (`x`) permission on the `test.ksh` shell script:

```
$ chmod go+x   test.ksh
```

The next example shows how you can use symbolic notation to *remove* read and write permissions on a file from the group:

```
$ chmod g-rw  test.ksh
```

Second, you can use the *octal numbers* method to change file permissions. Each permission carries different numeric "weights": *read* carries a weight of 4, *write* a weight of 2, and *execute* a weight of 1. To determine a permission setting, just add the weights for the permissions you want to assign. The highest number that can be associated with each of the three different entities—owner, group, and all others—is 7, which is the same as having read, write, and execute permissions on the file. For example, consider the following:

```
$ ls
$ -rw-r--r--  1  oracle dba     102   Nov  11  15:20  test.txt
$ chmod 777 test.txt
$ ls
$ -rwxrwxrwx  1  oracle dba     102   Nov  11  15:20  test.txt
```

The file test.txt initially had its file permissions set to 644 (rw, r, r.) The command `chmod 777` assigned full permissions (read, write, and execute) to all three entities: owner, group, and all others. If you want to change this so that only the owner has complete rights and the others have no permissions at all, set the octal number to 700 (read, write, and execute permissions for the owner, and no permissions at all for the group or others) and use the `chmod` command as follows:

```
$ chmod 700 test.txt
$ ls -altr test.txt
-rwx------   1 oracle     dba              0 Mar 28 11:23 test.txt
$
```

Table 3-3 provides a short summary of the commands you can use to change file permissions. By default, all files come with read and write privileges assigned, and directories come with read, write, and execute privileges turned on.

Table 3-3. *UNIX Permissions in Symbolic Notation and Octal Numbers*

Symbolic Notation	Octal Number	Privilege Description
`---`	0	No privileges
`--x`	1	Execute only
`-w-`	2	Write only
`-wx`	3	Write and execute, no read
`r--`	4	Read only
`r-x`	5	Read and execute, no write
`rw-`	6	Read and write, no execute
`rwx`	7	Read, write, and execute (full privileges)

The `UMASK` environment variable determines the default file and directory permissions. Issue the following command to see the current defaults on your server:

```
$ umask
022
```

When you create a new file, it'll have the default permissions allowed by the `UMASK` variable. In the preceding example, the `UMASK` is shown to be 022, meaning that the group and others don't have write permissions by default on any new file that you create.

Changing the Group

You can change the group a file belongs to by using the `chgrp` command. You must be the owner of the file to change the group, and you can change the file's group only to a group that you belong to. Here's how you use the chgrp command:

```
$ chgrp groupname filename
```

Directory Management

There are several important directory commands that enable you to create, move, and delete directories.

The mkdir command lets you create a new directory:

```
$ mkdir newdir
```

You can use the mkdir command with the -p option to create any necessary intermediate directories if they don't already exist. The following example creates the directory /u01/, the directory /u01/app, and the directory /u01/app/oracle, all with a single command:

```
$ mkdir  -p  /u01/app/oracle
```

The command for removing directories is not the same as the command for removing files. To remove a directory, you can use the rmdir command, as in the following example (but first make sure you have removed all the files in the directory using the rm command):

```
$ rmdir testdir
```

The rmdir command only removes empty directories. To remove a directory that contains files, use the rm command with the -R (or -r) option. This command will recursively delete the entire contents of a directory before removing the directory itself:

```
$ rmdir -r newdir
```

To move around the UNIX hierarchical directory structure, use the cd command (which stands for "change directory").

```
$ pwd
/u01/app/oracle
$ cd  /u01/app/oracle/admin
$ cd  /u01/app/oracle
$ cd admin
$ pwd
/u01/app/oracle/admin
$
```

Notice that you can use the cd command with the complete absolute path or with the shorter relative path. You can also use it to change to a directory that is indicated by an environment variable. For example, cd $ORACLE_HOME will change your current directory to the directory that happens to be the location for ORACLE_HOME.

Important UNIX Directories

There are several directories that you'll regularly come across when you're using the UNIX system as a DBA:

- */etc*: The /etc directory is where the system administrator keeps the system configuration files. Important files here pertain to passwords (etc/passwd) and information concerning hosts (etc/hosts).
- */dev*: The /dev directory contains device files, such as printer configuration files.
- */tmp*: The /tmp directory is where the system keeps temporary files, possibly including the log files of your programs. Usually you'll have access to write to this directory.
- *home*: The home directory is the directory assigned to you by your UNIX administrator when he or she creates your initial account. This is where you'll land first when you log in. You own this directory and have the right to create any files you want here. To create files in other directories, or even to read files in other directories, you have to be given permission by the owners of those directories.
- *root*: The root directory, denoted simply by a forward slash (/), is owned by the system administrator and is at the very top level of the treelike directory structure.

Writing and Editing Files with the vi Editor

The vi editor is commonly used to write and edit files in the UNIX system. To the novice, the vi editor looks very cryptic and intimidating, but it need not be intimidating. In this section, you'll learn how to use the vi editor to create and save files. You'll find that vi really is a simple text editor, with many interesting and powerful features.

Creating and Modifying Files Using vi

You start vi by typing **vi** or, better yet, by typing **vi *filename*** to start up the vi editor and show the contents of the *filename* file on the screen. If the file doesn't exist, vi allocates a memory buffer for the file, and you can later save the contents into a new file.

Let's assume you want to create and edit a new file called test.txt. When you type the command **vi test.txt**, the file will be created and the cursor will blink, but you can't start to enter any text yet because you aren't in the input mode. All you have to do to switch to input mode is type the letter **i**, which denotes the "insert" or "input" mode. You can start typing now just as you would in a normal text processor.

■**Note** If you need to create a file but don't want to enter any data into it, you can simply create a file with the `touch` command. If you use the `touch` command with a new filename as the argument, `touch` simply creates an empty file where none previously existed (unless you specify the `-c` flag). If you use an existing filename as the argument to the `touch` command, the last-accessed time of the file is changed to the time when the `touch` command was run. Here's an example:

```
touch program.one
```

This command sets the last access and modification times of the program.one file to the current date and time. If the program.one file does not exist, the `touch` command will create a file with that name.

Table 3-4 shows some of the most basic vi navigation commands, which enable you to move around within files.

Table 3-4. *Basic vi Navigation Commands*

Command	Description
h	Move a character to the left.
l	Move a character to the right.
j	Move a line down.
k	Move a line up.
w	Go to the beginning of the next word.
b	Go to the beginning of the previous word.
$	Go to the end of the current line.
^	Go to the start of the current line.
:G	Go to the end of the file.
:1	Go to the top of the file.

In addition to the cursor-movement commands, there are numerous vi text-manipulation commands, but unless you are a full-time system administrator or a UNIX developer, the average DBA can get by nicely with the few text commands summarized in Table 3-5.

Table 3-5. *Important vi Text-Manipulation Commands*

Command	Description
i	Start inserting from the current character.
a	Start inserting from the next character.
o	Start inserting from a new line below.
O	Start inserting from a new line above.
x	Delete the character where the cursor is.
dd	Delete the line where the cursor is.
r	Replace the character where the cursor is.
/text	Search for a text string.
:s/old/new/g	Replace (substitute) a text string with a new string.
yy	Yank or move a line.
p	Paste a copied line after the current cursor.
P	Paste a copied line above the current cursor.
:wq	Save and quit.
:q	Exit and discard changes.

For further information on vi navigation and text manipulation commands, you can always look up a good reference, such as *A Practical Guide to the UNIX System* by Mark Sobell (Addison Wesley).

Moving Around with the head and tail Commands

The `head` and `tail` UNIX file commands help you get to the top or bottom of a file. By default, they will show you the first or last ten lines of the file, but you can specify a different number of lines in the output, by specifying a number next to the `head` or `tail` command. The following example shows how you can get the first five lines of a file (the /etc/group file, which shows all the groups on the UNIX server):

```
$ head -5 /etc/group
root::0:root
other::1:root,hpdb
bin::2:root,bin
sys::3:root,uucp
adm::4:root
$
```

The `tail` command works in the same way, but it displays the last few lines of the file. The `tail` command is very useful when you are performing a task like a database software installation, because you can use it to display the progress of the installation process and see what's actually happening.

OTHER EDITORS

In addition to the UNIX vi editor, there are several other alternatives you can use, including pico, sed, and Emacs. Most are simple text editors that you can use in addition to the more popular vi editor. It's worth noting that Emacs works well in graphical mode when you use the X Window System, and there are also specific editors for X, such as dtpad. Some useful information on the various UNIX editors can be found at http://www.helpdesk.umd.edu/systems/wam/general/1235/.

Vim (or Vi improved) is an enhanced clone, if you will, of vi, and it is one of the most popular text editors among Linux administrators. You can download Vim from http://www.vim.org/download.php. For an excellent introduction to the Vim editor and its use with SQL*Plus, see David Kalosi's article "Vimming With SQL*Plus" at http://www.oracle.com/technology/pub/articles/kalosi_vim.html.

Extracting and Sorting Text

The cat and more utilities that you've seen earlier in the "Overview of Basic UNIX Commands" section, dump the entire contents of a text file onto the screen. If you want to see only certain parts of a file, however, you can use text-extraction utilities. Let's look at how you can use some of the important text-extraction tools.

Using grep to Match Patterns

I described the `grep` command briefly earlier in the chapter—you use the `grep` command to find matches for certain patterns in a string, using regular expressions. (For a good introduction to regular expressions, see the tutorial at http://www.regular-expressions.info/tutorial.html.) The word grep is an acronym for "global regular expression print," and it is derived from the following vi command, which prints all lines matching the regular expression *re*.

```
g/re/p
```

You can think of regular expressions as the search criteria used for locating text in a file; `grep` is thus similar to the find command in other operating systems. `grep` searches through each line of the file (or files) for the first occurrence of the given string, and if it finds that string, it prints the line. For example, to output all the lines that contain the expression "oracle database" in the file test.txt, you use the `grep` command in the following way:

```
$ grep 'oracle database' test.txt
```

In order to output all lines in the test.txt file that don't contain the expression "oracle database", you use the `grep` command with the `-v` option, as shown here:

```
$ grep -v 'oracle database' test.txt
```

In addition to the `-v` option, you can use the `grep` command with several other options:

`-c`	Prints a count of matching lines for each input file
`-l`	Prints the name of each input file
`-n`	Supplies the line number for each line of output
`-i`	Ignores the case of the letters in the expression

In addition to `grep`, you can use `fgrep` (fixed grep) to search files. The `fgrep` command doesn't use regular expressions. The command performs direct string comparisons, to find matches for a fixed string, instead of a regular expression.

The egrep version of grep helps deal with complex regular expressions, and is faster than the regular grep command.

Cutting, Pasting, and Joining Text

Often, you need to strip part of a file's text or join text from more than one file. UNIX provides great commands for performing these tasks, as I show in the following sections.

Outputting Columns with the cut Command

The cut command will output specified columns from a text file. Let's say you have a file named example.txt with the following text:

```
one two three
four five six
seven eight nine
ten eleven twelve
```

You can specify the fields you want to extract with the -f option. The following command will return just the second column in the example.txt file:

```
$ cut -f2 example.txt
two
five
eight
eleven
```

You use the -c option with the cut command to specify the specific characters you want to extract from a file. The following two commands extract the tenth character and then characters 10–12 from the password.txt file:

```
$ password.txt | cut -c10
$ password.txt | cut -c10-12
```

You can use the -d option in tandem with the -f option to extract characters up to a specified delimiter. The following example specifies that the cut command extract the first field (f1) of the passwd file, with the -d option specifying that the field is delimited by a colon (:). (The passwd file, located in the /etc directory, is where UNIX and Linux systems keep their user account information.)

```
$ cut -d":" -f1 /etc/passwd
root
daemon
bin
sys
adm
uucp
mail
```

Joining Files with the paste Command

The paste command takes one line from one source and combines it with another line from another source. Let's say you have two files: test1.txt contains the string "one two three" and test2.txt contains "one four five six". You can use the paste command to combine the two files as shown here:

```
$ paste test1.txt test2.txt
one two three    one four five six
```

Joining Files with the join Command

The `join` command will also combine the contents of two files, but it will work only if there is a common field between the files you are joining. In the previous section, test1.txt and test2.txt don't have a common column, so using the `join` command with those two files won't produce any output. However, suppose you have two files, test.one and test.two, with their contents as follows:

```
test.one                                test.two
11111   Dallas              11111      High Tech
22222   Houston             22222      Oil and Energy
```

By default the `join` command looks only at the first fields for matches, so it will give you the following result, based on the common (first) column:

```
$ join test.one test.two
11111       Dallas            High Tech
22222       Houston           Oil and Energy
```

The `-1` option lets you specify which field to use as the matching field in the first file, and the `-2` option lets you specify which field to use as the matching field in the second file. For example, if the second field of the first file matches the third field of the second file, you would use the `join` command as follows:

```
$ join -1 2 -2 3 test.one test.two
```

You use the `-o` option to specify output fields in the following format: *file.field*. Thus, to print the second field of the first file and the third field of the second file on matching lines, you would use the `join` command with the following options:

```
$ join -o 1.2 2.3 test.one test.two
```

Sorting Text with the sort Command

You can sort lines of text files, whether from a pipe or from a file, using the `sort` command. If you use the `-m` option, `sort` simply merges the files without sorting them. Let's say you have a file called test.txt with the following contents:

```
$ cat test.txt
yyyy
bbbb
aaaa
nnnn
```

By using the `sort` command, you can output the contents of the test.txt file in alphabetical order:

```
$ sort test.txt
aaaa
bbbb
nnnn
yyyy
```

By default, `sort` operates on the first column of the text.

Removing Duplicate Lines with the uniq Command

The `uniq` command removes duplicate lines from a sorted file. This command often follows the `sort` command in a pipe. By using the `-c` option, it can be used to count the number of occurrences of a line, or by using the `-d` option, it can report only the duplicate lines.

```
$ sort -m test.one test.two | uniq -c
      1 New test.
      2 Now testing
      1 Only a test.
```

In the preceding example, the `sort` command merges the two files, test.one and test.two, using the `-m` option. The output is piped to the `uniq` command with the `-c` option. What you get is an alphabetized list, with all duplicate lines removed. You also get the frequency of occurrence of each line.

Shell Scripting

Although the preceding commands and features are useful for day-to-day work with UNIX, the real power of this operating system comes from the user's ability to create shell scripts. In this section, you'll start slowly by building a simple shell program, and you'll proceed to build up your confidence and skill level as you move along into branching, looping, and all that good stuff.

What Is a Shell Program?

A shell script (or shell program) is simply a file containing a set of commands. The shell script looks just like any regular UNIX file, but it contains commands that can be executed by the shell. Although you'll learn mostly about Korn shell programming here, Bourne and C shell programming are similar in many ways. If you want to make the Korn shell your default shell, ask your system administrator to set it up by changing the shell entry for your username in the /etc/passwd file.

Before you begin creating a shell program, you should understand that shell programs don't contain any special commands that you can't use at the command prompt. In fact, you can type any command in any shell script at the command prompt to achieve the same result. All the shell program does is eliminate the drudgery involved in retyping the same commands every time you need to perform a set of commands together. Shell programs are also easy to schedule on a regular basis.

Using Shell Variables

You learned earlier in this chapter how shell variables are used to set up your UNIX environment. It's common to set variables within shell programs, so that these variables will hold their values for as long as the shell program executes.

If you're running the shell program manually, you can set the shell variables in the session you're using, and there's really no need for separate specification of shell variables in the shell program. However, you won't always run a shell program manually—that defeats the whole purpose of using shell programs in the first place. Shell programs are often run as part of the cron job, and they could be run from a session that doesn't have all the environmental variables set correctly. By setting shell variables in the program, you can make sure you're using the right values for key variables such as `PATH`, `ORACLE_SID`, and `ORACLE_HOME`.

Evaluating Expressions with the test Command

In order to write good shell scripts, you must understand how to use the `test` command. Most scripts involve conditional (if-then, while-do, until-do) statements. The `test` command helps in determining whether a certain condition is satisfied or not.

The `test` command evaluates an expression and returns a 0 value if the condition is true; otherwise it returns a value greater than 0, usually 1.

The syntax for the test command is as follows:

```
test expression
```

You can use the test command in conjunction with the if, while, or until constructs or use it by itself to evaluate any expression you like. Here is an example:

```
$ test "ONE" = "one"
```

This statement asks the test command to determine whether the string "ONE" is the same as the string "one".

You can use the test command in the implicit form (with an alias), by using square brackets instead of the test command, as shown here:

```
$ [ "ONE" = "one" ]
```

To find out whether the test command (or its equivalent, the square brackets) evaluated the expression "ONE" = "one" to be true or false, remember that if the result code (same as exit code) is 0, the expression is true, and otherwise it is false. To find the result code, all you have to do is use the special variable ?$, which will show you the exit code for any UNIX or Linux command. In our case, here is the exit code:

```
$ test "ONE" = "one"
$ echo $?
0
```

You can use exit codes in your shell scripts to check the execution status of any commands you use in the script.

You can use the following relations with the test command while comparing integers:

-ne: not equal

-eq: equal

-lt: less than

-gt: greater than

-ge: greater than or equal to

-le: less than or equal to

Executing Shell Programs with Command-Line Arguments

It's common to use arguments to specify parameters to shell programs. For example, you can run the shell program example.ksh as follows:

```
$ example.ksh prod1 system
```

In this case, example.ksh is your shell script, and the command-line arguments are prod1, the database name, and system, the username in the database. There are two arguments inside the shell script referred to as $1 and $2, and these arguments correspond to prod1 and system.

UNIX uses a *positional system,* meaning that the first argument after the shell script's name is the variable $1, the second argument is the value of the variable $2, and so on. Thus, whenever there's a reference to $1 inside the shell script, you know the variable is referring to the first argument (prod1, the database name, in this example).

By using command-line arguments, the script can be reused for several database and username combinations—you don't have to change the script.

Analyzing a Shell Script

Let's look at a simple database-monitoring shell script, example.ksh. This script looks for a certain file and lets you know if it fails to find it. The script uses one command-line argument to specify the name of the database. You therefore will expect to find a `$1` variable in the script.

When the shell program is created, UNIX has no way of knowing it's an executable program. You make your little program an executable shell script by using the `chmod` command:

```
$ ll example.ksh
-rw-rw-rw-  1   salapati    dba    439    feb  02    16:51  example.ksh
$ chmod 766 example.ksh
$ ll example.ksh
4-rwxrw-rw-  1   salapati   dba    439    feb  02    16:52  example.ksh
$
```

You can see that when the script was first created, it wasn't executable, because it didn't have the execution permissions set for anyone. By using the `chmod` command, the execution permission is granted to the owner of the program, salapati, and now the program is an executable shell script.

Here is the example.ksh shell script, which looks for a certain file in a directory and sends out an e-mail to the DBA if the file is not found there:

```
#!/bin/ksh
ORACLE_SID=$1Export ORACLE_SID
PATH=/usr/bin:/usr/local/bin:/usr/contrib./bin:$PATH
export PATH
ORACLE_BASE=${ORACLE_HOME}/../..;
export ORACLE_BASE
export CURRDATE='date +%m%dY_%H%M'
export LOGFILE=/tmp/dba/dba.log
test -s $ORACLE_HOME/dbs/test${ORACLE_SID}.dbf
if  [   'echo  $?'  -ne  0  ]
then
    echo   "File not found!"
mailx  -s  "Critical: Test file not found!"   dba@bankone.com   <  $LOGFILE
fi
```

Let's analyze the example.txt shell script briefly. The first line in the program announces that this is a program that will use the Korn shell—that's what `#!/bin/ksh` at the top of the script indicates. This is a standard line in all Korn shell programs (and programs for other shells have equivalent lines).

In the next line, you see `ORACLE_SID` being assigned the value of the `$1` variable. Thus, $1 will be assigned the value of the first parameter you pass with the shell program at the time of execution, and that value will be given to `ORACLE_SID`. The script also exports the value for the `ORACLE_BASE` environment variable.

Next, the program exports the values of three environmental variables: `PATH`, `CURRDATE`, and `LOGFILE`.

Then the script uses the file-testing command, `test`, to check for the existence of the file test*prod1*.dbf (where *prod1* is the value of `ORACLE_SID`) in a specific location. In UNIX, the success of a command is indicated by a value of 0 and failure is indicated by 1; you'll also recall that `echo $?`*`variable_name`* will print the value of the variable on the screen. Therefore, the next line, `if [ 'echo $? ' -ne 0]`, literally means "if the result of the `test` command is negative" (which is the same as saying, "if the file doesn't exist"). If that's the case, the `then` statement will write "File not found" in the log file.

The `then` statement also uses the `mail` program to e-mail a message to the DBA saying that the required file is missing. The mail program lets you send mail to user accounts on another UNIX server or to a person's e-mail address.

All you have to do to run or execute this shell script is simply type the name of the script at the command prompt, followed by the name of the database. For this simple method to work, however, you must be in the Korn shell when you run the script.

Now that you've learned the basics of creating shell scripts, let's move on to some powerful but still easy techniques that will help you write more powerful shell programs.

Flow-Control Structures in Korn Shell Programming

The Korn shell provides several flow-control structures similar to the ones found in regular programming languages, such as C or Java. These include the conditional structures that use `if` statements and the iterative structures that use `while` and `for` statements to loop through several steps based on certain conditions being satisfied. Besides these flow-control structures, you can use special commands to interrupt or get out of loops when necessary.

Conditional Branching

Branching constructs let you instruct the shell program to perform alternative tasks based on whether a certain condition is true or not. For example, you can tell the program to execute a particular command if a certain file exists and to issue an error message if it doesn't. You can also use the case structure to branch to different statements in the program depending on the value a variable holds.

In the following sections, you'll look at an example that shows the use of a simple conditional branching expression, and you'll look at another example that uses the `case` command.

The if-then-else Control Structure

The most common form of conditional branching in all types of programming is the `if-then-else-fi` structure. This conditional structure will perform one of two or more actions, depending on the results of a test.

The syntax for the `if-then-else-fi` structure is as follows:

```
if  condition
then
        Action a
else
        Action b
fi
```

Make sure that the `then` is on the second line. Also, notice that the control structure ends in `fi` (which is `if` spelled backwards).

Here's an example of the `if-then-else-fi` structure:

```
#!/usr/bin/sh
LOGFILE= /tmp/dba/error.log
export LOGFILE
grep ORA- $LOGFILE > job.err
    if [ `cat job.err|wc -l` -gt 0 ]
      then
             mailx -s "Backup Job Errors" salapati@netbsa.org < job.err
      else mailx -s " Backup Job Completed Successfully" salapati@netbsa.org
    fi
```

This script checks to see whether there are any errors in an Oracle backup job log. The script uses the mailx program, a UNIX-based mail utility, to send mail to the DBA. The `-s` option of the mailx utility specifies the subject line for the e-mail. The contents of the job.err file will be sent as the output of the e-mail.

Looping

In real-world programming, you may want to execute a command several times based on some condition. UNIX provides several loop constructs to enable this, the main ones being the `while-do-done` loop, which executes a command while a condition is true; the `for-do-done` loop, which executes a command a set number of times; and the `until-do-done` loop, which performs the same command until some condition becomes true.

The next sections examine these three loop structures in more detail.

A while-do-done Loop

The `while-do-done` loop tests a condition each time before executing the commands within the loop. If the test is successful, the commands are executed. If the test is never successful, the commands aren't executed even once. Thus, the loop ensures that the commands inside the loop get executed "while" a certain condition remains true.

Here's the syntax for the `while-do-done` loop:

```
while condition
do
    commands
done
```

In the following example of the `while-do-done` loop, note that the command inside the loop executes 99 times (the `lt` relation ensures that as long as the value of the variable `VAR1` is less than 100, the script will `echo` the value of the variable):

```
#!/usr/bin/ksh
VAR1=1
while [ $VAR1 -lt 100 ]
do
        echo "value of VAR1 is: $VAR1"
        ((VAR1 =VAR1+1))
done
```

A for-do-done Loop

You can use the `for-do-done` loop when you have to process a list of items. For each item in the list, the loop executes the commands within it. Processing will continue until the list elements are exhausted.

The syntax of the `for-do-done` loop is as follows:

```
for var in list
do
    commands
done
```

Here's an example of a `for-do-done` loop (the `for` command uses the letter `F` as a variable to process the list of files in a directory):

```
#!/usr/bin/sh
##  this loop gives you a list of all files (not directories)
## in a specified directory.
for F in /u01/app/oracle
do
     if [ -f  $F]
     then
            ls    $F
     fi
done
```

An until-do-done Loop

An `until-do-done` loop executes the commands inside the loop until a certain condition becomes true. The loop executes as long as the condition remains false.

Here's the general syntax for the `until-do-done` loop:

```
until condition
do
    commands
done
```

The following is a simple example that shows how to use the `until-do-done` loop. The `print` command outputs the sentence within the quotes on the screen. The `-n` option specifies that the output should be placed on a new line. The UNIX command `read` will read a user's input and place it in the `answer` variable. The script then will continue to run until the user inputs the answer "YES":

```
until [[ $answer = "yes" ]];do
   print -n "Please accept by entering \"YES\": "
   read answer
   print ""
done
```

Branching with the case Command

The *case* structure is quite different from all the other conditional statements. This structure lets the program branch to a segment of the program based on the value of a certain variable. The variable's value is checked against several patterns, and when the patterns match, the commands associated with that pattern will be executed.

Here's the general syntax of the `case` command:

```
case var in
pattern1)
            commands
             ;;
pattern2)
            commands
            ;;
...
patternn)
            commands
             ;;
esac
```

Note that the end of the case statement is marked by `esac` (which is `case` spelled backwards). Here's a simple example that illustrates the use of the `case` command:

```
#!/usr/bin/sh
echo " Enter b to see the list of books"
echo " Enter t  to see the library timings"
echo " Enter e to exit the menu"
echo
echo "Please enter a choice": \c"
read VAR
case $VAR in
b/B) book.sh
        ;;
t/T) times.sh
       ;;
```

```
e/E) logout.sh
       ;;
*)    echo " "wrong Key entry: Please choose again"
esac
```

Dealing with UNIX Processes

When you execute your shell program, UNIX creates an active instance of your program, called the *process*. UNIX also assigns your process a unique identification number, called the *process ID* (PID). As a DBA, you need to know how to track the processes that pertain to your programs and the database instance that you are managing.

Gathering Process Information with ps

The `ps` command, with its many options, is what you'll use to gather information about the currently running processes on your system. The `ps -ef` command will let you know the process ID, the user, the program the user is executing, and the length of the program's execution.

In the following example, the `ps -ef` command is issued to display the list of processes, but because the list is going to be very long, the pipe command is used to filter the results. The `grep` command ensures that the list displays only those processes that contain the word "pmon". The pmon process is an essential Oracle background process, and I explain it in Chapter 4. The output indicates that three different Oracle databases are currently running:

```
$  ps -ef | grep pmon
oracle 10703        1     0    09:05:39  ?         0.00    ora_pmon_test
oracle  18655       1     0    09:24:00  ?         0.00    ora_pmon_prod1
oracle 10984        1     0    09:17:50  ?         0.00    ora_pmon_finance
$
```

Running Processes after Logging Out

Sometimes, you may want to run a program from a terminal, but you then need to log out from it after a while. When you log out, a "hangup" signal is sent to all the processes you started in that session. To keep the programs you are executing from terminating abruptly when you disconnect, you can run your shell programs with the `nohup` option, which means "no hang up." You can then disconnect, but your (long) program will continue to run.

Here's how you specify the `nohup` option for a process:

```
$ nohup test.ksh
```

Running Processes in the Background

You can start a job and then run it in the background, returning control to the terminal. The way to do this is to specify the `&` parameter after the program name, as shown in the following example (you can use the `ps` command to see if your process is still running, by issuing either the `ps -ef` or `ps -aux` command):

```
$  test.ksh &
[1]     27149
$
```

You can also put a currently running job in the background, by using the Ctrl+Z sequence. This will suspend the job and run it in the background. You can then use the command `fg%jobnumber` to move your backgrounded job back into the foreground.

Terminating Processes with the kill Command

Sometimes you'll need to terminate a process because it's a runaway or because you ran the wrong program. In UNIX, signals are used to communicate with processes and to handle exceptions. To bring a UNIX process to an abrupt stop, you can use the `kill` command to signal the shell to terminate the session before its conclusion. Needless to say, mistakes in the use of the `kill` command can prove disastrous.

Note Although you can always kill an unwanted Oracle user session or a process directly from UNIX itself, you're better off always using Oracle's methods for terminating database sessions. There are a couple of reasons for this. First, you may accidentally wipe out the wrong session when you exit from the UNIX operating system. Second, when you're using the Oracle shared server method, a process may have spawned several other processes, and killing the operating system session could end up wiping out more sessions than you had intended.

There is more than one `kill` signal that you can issue to terminate any particular process. The general format of the `kill` command is as follows:

```
kill -[signal] PID
```

The *signal* option after the `kill` command specifies the particular signal the `kill` command will send to a process, and *PID* is the process ID of the process to be killed. To kill a process gracefully, you send a SIGTERM signal to the process, using either the signal's name or number. Either of the following commands will kill the process with a PID of 21427:

```
$ Kill -SIGTERM  21427
$ Kill -15  21427
```

If your SIGTERM signal, which is intended to terminate a process gracefully, doesn't succeed in terminating the session, you can send a signal that will *force* the process to die. To do this, use the `kill -9` signal:

```
$ kill -9 21427
```

UNIX System Administration and the Oracle DBA

It isn't necessary for you to be an accomplished system administrator to manage your database, but it doesn't hurt to know as much as possible about what system administration entails. Most organizations hire UNIX system administrators to manage their systems, and as an Oracle DBA, you'll need to interact closely with those UNIX system administrators. Although the networking and other aspects of the system administrator's job may not be your kettle of fish, you do need to know quite a bit about disk management, process control, and backup operations. UNIX system administrators are your best source of information and guidance regarding these issues.

UNIX Backup and Restore Utilities

Several utilities in UNIX make copies or restore files from copies. Of these, the `dd` command pertains mainly to the so-called raw files. Most of the time, you'll be dealing with UNIX file systems, and you'll need to be familiar with two important archiving facilities—tar and cpio—to perform backups and restores. Tar is an abbreviation for "tape file archiver," and was originally designed to write to tapes. Cpio stands for "copy input and output." Other methods such as `fbackup/frecover`, `dump/restore`, and `xdump/vxrestore` exist, but they are mainly of interest to UNIX administrators. You most likely will use the `tar` and `cpio` commands to perform backups. The `tar` command can copy and restore archives of files using a tape system or a disk drive. By default, tar output is placed on /dev/rmt/Om, which refers to a tape drive.

The following `tar` command will copy the data01.dbf file to a tape, which is specified in the format /dev/rmt/0m. The `-cvf` option creates a new archive (the hyphen is optional). The `c` option asks tar to create a new archive file, and the `v` option stands for verbose, which specifies that the files be listed as they are being archived:

```
$ tar -cvf /dev/rmt/0m    /u10/oradata/data/data01.dbf
```

The following `tar` command will extract the backed-up files from the tape to the specified directory:

```
$ tar -xvf/dev/rmt/0m     /u20/oradata/data/data01.dbf
```

The `x` option asks tar to extract the contents of the specified file. The `v` and `f` options have the same meanings as in the previous example.

The `cpio` command with the `-o` (copy out) option copies files to standard output, such as disk or tape. The following command will copy the contents of the entire current directory (all the files) to the /dev/rmt/0m tape:

```
$ ls | cpio -O > /dev/rmt/0m
```

The `cpio` command with the `-i` (copy in) option extracts files from standard input. The following command restores all the contents of the specified tape to the current directory:

```
$ cpio -i < /dev/rmt/0m
```

The crontab and Automating Scripts

Most DBAs will have to schedule their shell programs and other data-loading programs for regular execution by the UNIX system. UNIX provides the cron table, or crontab, to schedule database tasks. In this section, you'll learn how to schedule jobs with this wonderful, easy-to-use utility.

You can invoke the crontab by typing in **crontab -l**. This will give you a listing of the contents of crontab. To add programs to the schedule or change existing schedules, you need to invoke crontab in the edit mode, as shown here:

```
$ crontab -e
```

Each line in the crontab is an entry for a regularly scheduled job or program, and you edit the crontab the same way you edit any normal vi-based file. Each line in the /etc/crontab file represents a job that you want to execute, and it has the following format:

```
Minute     hour      day     month     day of week      command
```

The items in the crontab line can have the following values:

- *minute*: Any integer from 0 to 59
- *hour*: Any integer from 0 to 23
- *day*: Any integer from 1 to 31 (this must be a valid date if a month is specified)
- *month*: Any integer from 1 to 12 (or the short name of the month, such as jan or feb)
- *day of week*: Any integer from 0 to 7, where 0 and 7 represent Sunday, 1 is Saturday, and so on
- *command*: The command you want to execute (this is usually a shell script)

Here's a simple example of a crontab line:

```
#---------------------------------------------------------------------
minute        hour    date   month    day of week         command
30             18       *     *           1-6             analyze.ksh
#----------------------------------------------------------------------
```

The preceding code indicates that the program analyze.ksh will be run Monday through Saturday at 6:30 PM. Once you edit the crontab and input the lines you need to run your commands, you can exit out of cron by pressing Shift+wq, just as you would in a regular vi file. You now have "cronned" your job, and it will run without any manual intervention at the scheduled time.

It's common practice for DBAs to put most of their monitoring and daily data-load jobs in the crontab for automatic execution. If crontab comes back with an error when you first try to edit it, you need to talk to your UNIX system administrator and have appropriate permissions granted.

■**Note** You'll use crontab for all your regularly scheduled database or operating system jobs, but if you want to schedule a task for a single execution, you can use the `at` or `batch` command instead. Look up the man pages for more information on these two scheduling commands.

Using Telnet

Telnet is an Internet protocol for accessing remote computers from your PC or from another UNIX server or workstation. Your machine simply needs to be connected to the target machine through a network, and you must have a valid user account on the computer you are connecting to. To use telnet on your PC, for example, go to the DOS prompt and type **telnet**. At the telnet prompt, type in either the UNIX server's IP address or its symbolic name, and your PC will connect to the server. Unless you are doing a lot of file editing, telnet is usually all you need to connect and work with a UNIX server, in the absence of a terminal emulator.

The following example session shows a connection being made to and disconnection from a server named hp50. Of course, what you can do on the server will depend on the privileges you have on that machine.

```
$ telnet hp5
Trying...
Connected to hp5.netbsa.org.
Escape character is '^]'.
Local flow control on
Telnet TERMINAL-SPEED option ON
login: oracle
Password:
Last   successful login for oracle: Tue Nov  5 09:39:45
CST6CDT 2002 on tty
```

```
Last unsuccessful login for oracle: Thu Oct 24 09:31:17
CST6CDT 2002 on tty
Please wait...checking for disk quotas
...
You have mail.
TERM = (dtterm)
oracle@hp5[/u01/app/oracle]
$
```

Once you log in, you can do everything you are able to do when you log directly into the server without using telnet.

You log out from your telnet session in the following way:

```
$ exit
logout
Connection closed by foreign host.
$
```

Remote Login and Remote Copy

Rlogin is a UNIX service that's very similar to telnet. Using the `rlogin` command, you can log in to a remote system just as you would using the telnet utility. Here is how you can use the `rlogin` command to remotely log in to the server hp5:

```
$ rlogin hp5
```

You'll be prompted for a password after you issue the preceding command, and upon the validation of the password, you'll be logged in to the remote server.

To copy files from a server on the network, you don't necessarily have to log in to that machine or even use the FTP service. You can simply use the `rcp` command to copy the files. For example, to copy a file named /etc/oratab from the server hp5 to your client machine (or to a different server), you would use the `rcp` command as follows:

```
$ rcp hp5:/etc/oratab/  .
```

The dot in the command indicates that the copy should be placed in your current location.

To copy a file called test.txt from your current server to the /tmp directory of the server hp5, you would use the rcp command as follows:

```
$ rcp /test/txt  hp5:/tmp
```

Using SSH, the Secure Shell

The secure shell, SSH, is a protocol like Telnet that enables remote logins to a system. The big difference between the `ssh` command (which uses the SSH protocol) and `rlogin` is that SSH is a secure way to communicate with remote servers—SSH uses encrypted communications to connect two untrusted hosts over an insecure network. The plan is for `ssh` to eventually replace `rlogin` as a way to connect to remote servers.

Here's an example of using the `ssh` command to connect to the hp5 server:

```
$ ssh prod5
Password:
Last   successful login for oracle: Thu Apr  7 09:46:52 CST6CDT 2005 on tty
Last unsuccessful login for oracle: Fri Apr  1 09:02:00 CST6CDT 2005
oracle@prod5   [/u01/app/oracle]
$
```

Using FTP to Send and Receive Files

FTP, the File Transfer Protocol, is a popular way to transmit files between UNIX servers or between a UNIX server and a PC. It's a simple and fast way to send files back and forth.

The following is a sample FTP session between my PC and a UNIX server on my network. I am getting a file from the UNIX server called prod5 using the `ftp get` command.

```
$ ftp prod5
connected to prod5
ready.
User (prod5:-(none)): oracle
331 Password required for oracle.
Password:
User oracle logged in.
ftp> pwd
'/u01/app/oracle" is the current directory.
ftp>  cd admin/dba/test
CWD command successful.
ftp> get analyze.ksh
200  PORT command successful.
150  Opening ASCII mode data connection for analyze.ksh
 (3299 bytes).
226 Transfer complete.
ftp: 3440 bytes received in 0.00Seconds  3440000.00Lbytes/sec.
ftp> bye
221 Goodbye.
$
```

If, instead of getting a file, I wanted to place a file from my PC onto the UNIX server I connected to, I would use the `put` command, as in `put analyze.ksh`. The default mode of data transmission is the ASCII character text mode; if you want binary data transmission, just type in the word **binary** before you use the `get` or `put` command.

Of course, GUI-based FTP clients are an increasingly popular choice. If you have access to one of those, transferring files is usually simply a matter of dragging and dropping files from the server to the client, much like moving files in Windows Explorer.

UNIX System Performance Monitoring Tools

Several tools are available for monitoring the performance of the UNIX system. These tools check on the memory and disk utilization of the host system and let you know of any performance bottlenecks. In this section, you'll explore the main UNIX-based monitoring tools and see how these tools can help you monitor the performance of your system.

The Basics of Monitoring a UNIX System

A slow system could be the result of a bottleneck in processing (CPU), memory, disk, or bandwidth. System monitoring tools help you to clearly identify the bottlenecks causing poor performance. Let's briefly examine what's involved in the monitoring of each of these resources on your system.

Monitoring CPU Usage

As long as you are not utilizing 100 percent of the CPU capacity, you still have juice left in the system to support more activity. Spikes in CPU usage are common, but your goal is to track down what, if any, processes are contributing excessively to CPU usage. These are some of the key factors to remember while examining CPU usage:

- *User versus system usage*: You can identify the percentage of time the CPU power is being used for users' applications as compared with time spent servicing the operating system's overhead. Obviously, if the system overhead accounts for an overwhelming proportion of CPU usage, you may have to examine this in more detail.
- *Runnable processes*: At any given time, a process is either running or waiting for resources to be freed up. A process that is waiting for the allocation of resources is called a *runnable process*. The presence of a large number of runnable processes indicates that your system may be facing a power crunch—it is CPU-bound.
- *Context switches and interrupts*: When the operating system switches between processes, it incurs some overhead due to the so-called context switches. If you have too many context switches, you'll see deterioration in CPU usage. You'll incur similar overhead when you have too many interrupts, caused by the operating system when it finishes certain hardware- or software-related tasks.

Managing Memory

Memory is one of the first places you should look when you have performance problems. If you have inadequate memory (RAM), your system may slow down due to excessive swapping. Here are some of the main factors to focus on when you are checking system memory usage:

- *Page ins and page outs*: If you have a high number of page ins and page outs in your memory statistics, it means that your system is doing an excessive amount of *paging,* the moving of pages from memory to the disk system due to inadequate available memory. Excessive paging could lead to a condition called *thrashing,* which just means you are using critical system resources to move pages back and forth between memory and disk.
- *Swap ins and swap outs*: The swapping statistics also indicate how adequate your current memory allocation is for your system.
- *Active and inactive pages*: If you have too few inactive memory pages, it may mean that your physical memory is inadequate.

Monitoring Disk Storage

When it comes to monitoring disks, you should look for two things. First, check to make sure you aren't running out of room—applications add more data on a continuous basis, and it is inevitable that you will have to constantly add more storage space. Second, watch your disk performance—are there any bottlenecks due to slow disk input/output performance?

Here are the basic things to look for:

- *Check for free space*: Using simple commands, a system administrator or a DBA can check the amount of free space left on the system. It's good, of course, to do this on a regular basis so you can head off a resource crunch before it's too late. Later in this chapter, I'll show you how to use the `df` and the `du` commands to check the free space on your system.
- *Reads and writes*: The read/write figures give you a good picture of how hot your disks are running. You can tell whether your system is handling its workload well, or if it's experiencing an extraordinary I/O load at any given time.

Monitoring Bandwidth

By measuring bandwidth use, you can measure the efficiency of the transfer of data between devices. Bandwidth is harder to measure than simple I/O or memory usage patterns, but it can still be immensely useful to collect bandwidth-related statistics.

Your network is an important component of your system—if the network connections are slow, the whole application may appear to run very slowly. Simple network statistics like the number of bytes received and sent will help you identify network problems.

High network packet collision rates, as well as excessive data transmission errors, will lead to bottlenecks. You need to examine the network using tools like netstat (discussed later) to see if the network has any bottlenecks.

Monitoring Tools for UNIX Systems

In order to find out what processes are running, you'll most commonly use the process command, `ps`. For example, the following example checks for the existence of the essential pmon process, to see if the database is up:

```
$ ps -ef | grep pmon
```

Of course, to monitor system performance, you'll need more sophisticated tools than the elementary `ps` command. The following sections cover some of the important tools available for monitoring your system's performance.

Monitoring Memory Use with vmstat

The vmstat utility helps you monitor memory usage, page faults, processes and CPU activity. The vmstat utility's output is divided into two parts: virtual memory (VM) and CPU. The VM section is divided into three parts: *memory*, *page*, and *faults*. In the memory section, *avm* stands for "active virtual memory" and *free* is short for "free memory." The *page* and *faults* items provide detailed information on page reclaims, pages paged in and out, and device interrupt rates.

The output gives you an idea about whether the memory on the system is a bottleneck during peak times. The *po* (page outs) variable under the page heading should ideally be 0, indicating that there is no swapping—that the system is not transferring memory pages to swap disk devices to free up memory for other processes.

Here is some sample output from vmstat (note that I use the `-n` option to improve the formatting of the output):

```
$ vmstat -n
VM
    memory              page                   faults
 avm      free    re   at pi po fr de sr   in    sy    cs
1822671  8443043 1052 113 2  0  0  0  0   8554 89158 5272
CPU
    cpu          procs
 us sy id    r     b     w
 23  7 69    8    23     0
 22  8 70
 21  7 72
 22  7 71
$
```

Under the *procs* subheading in the CPU part of the output, the first column, *r*, refers to the run queue. If your system has 24 CPUs and your run queue shows 20, that means 20 processes are waiting in the queue for a turn on the CPUs, and it is definitely not a bad thing. If the same *r* value of 24 occurs on a machine with 2 CPUs, it indicates the system is CPU-bound—a large number of processes are waiting for CPU time.

In the CPU part of vmstat's output, *us* stands for the amount of CPU usage attributable to the users of the system, including your database processes. The *sy* part shows the system usage of the CPU, and *id* stands for the amount of CPU that is idle. In our example, roughly 70 percent of the CPU is idle for each of the four processors, on average.

Viewing I/O Statistics with iostat

The iostat utility gives you input/output statistics for all the disks on your system. The output is displayed in four columns:

- *device*: The disk device whose performance iostat is measuring
- *bps*: The number of kilobytes transferred from the device per second
- *sps*: The number of disk seeks per second
- *msps*: The time in milliseconds per average seek

The iostat command takes two parameters: the number of seconds before the information should be updated on the screen, and the number of times the information should be updated. Here is an example of the iostat output:

```
$ iostat 4 5
      device       bps      sps      msps
      c2t6d0       234      54.9     1.0
      c5t6d0       198      42.6     1.0
      c0t1d1       708      27.7     1.0
      c4t3d1       608      19.0     1.0
      c0t1d2       961      46.6     1.0
      c4t3d2       962      46.1     1.0
      c0t1d3       731      91.3     1.0
      c4t3d3       760      93.5     1.0
      c0t1d4        37       7.0     1.0
$
```

In the preceding output, you can see that the disks c0t1d2 and c4t3d2 are the most heavily used disks on the system.

Analyzing Read/Write Operations with sar

The UNIX sar (system activity reporter) command offers a very powerful way to analyze how the read/write operations are occurring from disk to buffer cache and from buffer cache to disk. By using the various options of the sar command, you can monitor disk and CPU activity, in addition to buffer cache activity.

The output for the sar command has the following columns:

- *bread/s*: The number of read operations per second from disk to the buffer cache
- *lread/s*: The number of read operations per second from the buffer cache
- *%rcache*: The cache hit ratio for read requests
- *bwrit/s*: The number of write operations per second from disk to the buffer cache
- *lwrit/s*: The number of write operations per second to the buffer cache
- *%wcache*: The cache hit ratio for write requests

Here's the output of a typical sar command which monitors your server's CPU activity, using the -u option (the 1 10 tells sar to refresh the output on the screen every second for a total of ten times):

```
$ sar -u 1 10
HP-UX prod5 B.11.11 U 9000/800    04/07/05

16:11:21    %usr      %sys       %wio   %idle
16:11:22      34         6         56       4
16:11:23      31         7         55       7
16:11:24      45         9         43       4
16:11:25      45         9         44       2
16:11:26      45        11         40       3
16:11:27      46        11         40       4
16:11:28      48        10         40       3
16:11:29      56        11         31       2
16:11:30      50        12         36       3
16:11:31      45        12         39       4

Average       44        10         42       4

$
```

In the preceding sar report, %usr shows the percentage of CPU time spent in the user mode, %sys shows the percentage of CPU time spent in the system mode, %wio shows the percentage of time the CPU is idle with some process waiting for I/O, and %idle shows the idle percentage of the CPU. You can see that the percentage of CPU due to processes waiting for I/O is quite high in this example.

Monitoring Performance with top

The top command is another commonly used performance-monitoring tool. Unlike some of the other tools, the top command shows you a little bit of everything, such as the top CPU and memory utilization processes, the percentage of CPU time used by the top processes, and the memory utilization.

The top command displays information in the following columns:

- *CPU*: Specifies the processor
- *PID*: Specifies the process ID
- *USER*: Specifies the owner of the process
- *PRI*: Specifies the priority value of the process
- *NI*: Specifies the nice value (nice invokes a command with an altered scheduling priority)
- *SIZE*: Specifies the total size of the process in memory
- *RES*: Specifies the resident size of the process
- *TIME*: Specifies the CPU time used by the process
- *%CPU*: Specifies the CPU usage as a percentage of total CPU
- *COMMAND*: Specifies the command that started the process

To invoke the top utility, you simply type the word **top** at the command prompt. To end the top display, just use the Ctrl+C key combination.

Here's an example of typical output of the top command on a four-processor UNIX machine. The first part of the output (not shown here) shows the resource usage for each processor in the system. The second part of the output, shown in the following snippet, gives you information about the heaviest users of your system.

```
$ top
CPU   PID  USER   PRI     NI  SIZE    RES     TIME   %CPU  COMMAND
21   2713 nsuser   134     0   118M    104M  173:31  49.90 ns-httpd
23   28611 oracle  241    20  40128K  9300K    2:20  46.60 oraclepasprod
20   6951 oracle   241    20  25172K 19344K    3:45  44.62 rwrun60
13   9334 oracle   154    20  40128K  9300K    1:31  37.62 oraclepasprod
22   24517 oracle  68     20  36032K  5204K    0:55  36.48 oraclepasprod
22   13166 oracle  241    20  40128K  9300K    0:41  35.19 oraclepasprod
12   14097 oracle  241    20  40128K  9300K    0:29  33.75 oraclepasprod
$
```

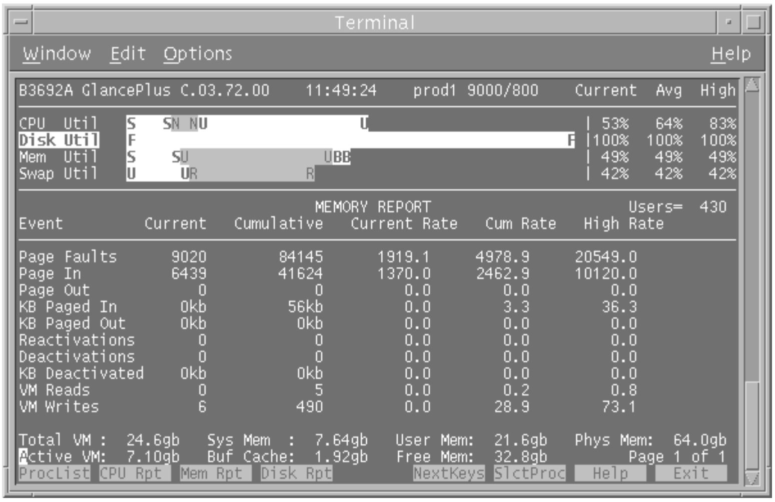
Monitoring the System with GlancePlus

Several UNIX operating systems have their own system-monitoring tools. For example, on the HP-UX operating system, GlancePlus is a package that is commonly used by system administrators and DBAs to monitor memory, disk I/O, and CPU performance.

Figure 3-3 shows a typical GlancePlus session in text mode, invoked with the following command:

```
$ glance -m
```

The CPU, memory, disk, and swap usage is summarized in the top section. The middle of the display gives you a detailed memory report, and at the bottom of the screen you can see a short summary of memory usage again.



```
B3692A GlancePlus C.03.72.00     11:49:24    prod1 9000/800     Current  Avg  High

CPU  Util    S   SN NU                    U                      | 53%    64%   83%
Disk Util    F                                                  F |100%  100%  100%
Mem  Util    S    SU                 UBB                         | 49%   49%   49%
Swap Util    U     UR              R                             | 42%   42%   42%

                                MEMORY REPORT                         Users=  430
Event           Current   Cumulative   Current Rate   Cum Rate   High Rate

Page Faults       9020        84145        1919.1      4978.9     20549.0
Page In           6439        41624        1370.0      2462.9     10120.0
Page Out             0            0           0.0         0.0         0.0
KB Paged In        0kb         56kb           0.0         3.3        36.3
KB Paged Out       0kb          0kb           0.0         0.0         0.0
Reactivations        0            0           0.0         0.0         0.0
Deactivations        0            0           0.0         0.0         0.0
KB Deactivated     0kb          0kb           0.0         0.0         0.0
VM Reads             0            5           0.0         0.2         0.8
VM Writes            6          490           0.0        28.9        73.1

Total VM :  24.6gb   Sys Mem   :  7.64gb   User Mem:   21.6gb   Phys Mem:  64.0gb
Active VM:  7.10gb   Buf Cache:  1.92gb   Free Mem:   32.8gb         Page 1 of 1
ProcList CPU Rpt  Mem Rpt  Disk Rpt          NextKeys SlctProc   Help    Exit
```

Figure 3-3. *A typical GlancePlus session in text mode*

Note that this session shows memory usage in detail because GlancePlus was invoked with the -m option (glance -c would give you a report on CPU usage, and glance -d would give you a disk usage report).

GlancePlus also has an attractive and highly useful GUI interface, which you can invoke by using the command `gpm`.

Monitoring the Network with Netstat

Besides monitoring the CPU and memory on the system, you need to monitor the network to make sure there are no serious traffic bottlenecks. The netstat utility comes in handy for this purpose, and it works the same way on UNIX as it does on the Windows servers.

Disks and Storage in UNIX

The topic of physical storage and using the disk system in UNIX is extremely important for the DBA—the choice of disk configuration has a profound impact on the availability and the performance of the database. Some Oracle databases benefit by using "raw" disk storage instead of disks controlled by the UNIX operating system. The Oracle Real Application Clusters (RACs) can only use the raw devices; they can't use the regular UNIX-formatted disks.

All the UNIX files on a system make up its *file system*, and this file system is created on a *disk partition*, which is a "slice" of a disk, the basic storage device.

Disk Storage Configuration Choices

The choices you make about how you configure your disk storage will have a major impact on the performance and the uptime of your database. It's not a good idea to make storage device decisions in a vacuum; rather, you should consider your database applications and the type of database that is going to be located on the storage systems when making these decisions.

For example, if you have a data warehouse, you may want your system administrator to use larger striping sizes for the disks. If you are going to have large numbers of writes to or reads from the database, you need to choose the appropriate disk configuration. Compared to the technologies of only a few years ago, today's ultra-sophisticated storage technologies make it possible to have both a high level of performance and high availability of data simultaneously.

Still, you have plenty of choices to make that will have an impact on performance and availability. The nature of the I/Os, database caches, read/write ratios, and other issues are fundamentally different in OLTP and DSS systems. Also, response-time expectations are significantly different between OLTP and DSS systems. Thus, a storage design that is excellent for one type of database may be a terrible choice for another type, so you need to learn more about the operational needs of your application at the physical design stage to make smart choices in this extremely critical area.

Monitoring Disk Usage

When setting up an Oracle system, you will typically make a formal request to the system administrator for physical disk space based on your sizing estimates and growth expectations for the database. Once the general space request is approved by the system administrator, he or she will give you the location of the mount points where your space is located. *Mount points* are directories on the system to which the file systems are mounted. You can then create all the necessary directories prior to the installation of the Oracle software and the creation of the database itself.

Once space is assigned for your software and databases, it's your responsibility to keep track of its usage. If you seem to be running out of space, you will need to request more space from the system administrator. Ideally, you should always have some extra free disk space on the mount points assigned to you so you can allocate space to your database files if the need arises. There are a couple of very useful commands for checking your disk space and seeing what has been used and what is still free for future use.

The df (disk free) command indicates the total allocation in bytes for any mount point and how much of it is currently being used. The df -k option gives you the same information in kilobytes, which is generally more useful. The following example shows the use of the df command with the -k option:

```
$df -k /finance09
/finance09 ( /dev/vgxp1_0f038/lvol1) :
7093226 total allocated Kb
1740427 free allocated Kb
5352799 used allocated Kb
75% allocation used
$
```

The preceding output shows that out of a total of 7.09GB allocated to the /finance09 mount point; about 5.35GB is currently allocated to various files and about 1.74GB of space is still free.

Another command that displays how the disks are being used is the du command, which indicates, in bytes, the amount of space being used by the mount point.

```
$ du -k /finance09
           /finance09/lost+found
           /finance09/ffacts/home
. . .
5348701    /finance09
$
```

As you can see in the preceding example, the du command indicates the actual space used by the various files and directories of the mount point (/finance09 in this case) and the total space used up by it.

I prefer the df -k command over the du -k command, because I can see at a glance the percentages of free space and used space.

Disk Storage, Performance, and Availability

Availability and performance lie at the heart of all disk configuration strategies. The one thing you can be sure of when you use disk-based storage systems is that a disk will fail at some point. All disks come with a mean time between failures (MTBF) rating, which could run into hundreds of thousands of hours, and you can expect an average disk with a high rating to last for many years. As we all know, though, averages can be dangerous in situations like this because an individual disk can fail at any time, regardless of its MTBF rating. Disk controllers manage the disks, and a controller failure can doom your system operations. It is common now to build redundancy into your disk systems (and other key components of the entire system) to provide continuous availability.

Performance is also an issue when you are considering the configuration of your storage devices. In systems with highly intensive simultaneous reads and writes, you can quickly end up with disk bottlenecks unless you plan the disk configuration intelligently from the beginning.

To improve performance, the common strategy employed is *disk striping,* which enables you to create a single logical unit out of several physical disks. The single logical unit is composed of alternating stripes from each disk in the set, and data is divided into equally sized blocks and written in stripes to each disk at the same time. Reads are done in the same way, with the simultaneous use of all the disks. Thus, you can enhance I/O operations dramatically, because you are now using the I/O capacity of a set of disks rather than just one.

Disk Partitioning

Raw disks aren't amenable to easy data access—you need to impose a structure on these disks. The first thing you need to do before using a hard disk is to partition, or slice, the disk. Partitioning enables you to store system and application data in separate sections of the disk, as well as manage space issues easily. Sometimes these partitions themselves are called disks, but they are all really parts of a single physical disk. Once you partition a disk, you can create operating system file systems on it.

Creating File Systems

Even after partitioning the whole disk, you still don't have a convenient way to access data or to store it. You can further refine your access methods by using file systems. File systems provide you with the following benefits:

- Individual ownership of files and directories
- Tracking of creation and modification times
- Data access control
- Accounting of space allocation and usage

Disk Striping

It's important to realize that you can place the file system on a single physical disk or you can put it across several "striped" physical disks. In the latter case, although the file system is on several disks, the user will see the files as being on one so-called logical volume. UNIX systems offer several ways of combining multiple disks into single logical volumes.

One way to create a logical device on many UNIX systems is to use a utility known as the Logical Volume Manager (LVM). Using an LVM, you can take, for example, ten physical disks of 4GB each and create one 40GB logical disk. Thus, disk striping can also enable you to create a much larger logical disk that can handle a larger file system. File systems can't traverse disks, so logical disks offer an easy way to create large volumes.

Logical Volumes and the Logical Volume Manager

Let's briefly look at the two basic methods of configuring physical disks. Although you may never have to do this yourself, it's a good idea to have a basic understanding of how disks are managed by system administrators. You can configure disks as whole disks or as logical volumes.

Whole disks are exactly what their name implies: each physical disk is taken as a whole and a single file system is created on each disk. You can neither extend nor shrink the file system at a later stage.

Logical volumes, on the other hand, are created by combining several hard disks or disk partitions. System administrators usually employ the sophisticated LVM to combine physical disks. A set of physical disks is combined into a volume group, which is then sliced up by the LVM into smaller logical volumes. Most modern systems use the LVM approach because it is an extremely flexible and easy way to manage disk space. For example, it's no problem at all to add space and modify partitions on a running system by using the LVM tool.

Once you create logical volumes, you can designate disk volumes as mount points, and individual files can then be created on these mount points.

RAID Systems

A redundant array of independent disk (RAID) device is a popular way to configure large logical (or virtual) disks from a set of smaller disks. The idea is simply to combine several small inexpensive disks into an array in order to gain higher performance and data security. This allows you to replace one very expensive large disk with several much cheaper small disks. Data is broken up into equal-sized chunks (called the stripe size), usually 32KB or 64KB, and a chunk is written on each disk, the exact distribution of data being determined by the RAID level adopted. When the data is read back, the process is reversed, giving you the appearance that one large disk, instead of several small disks, is being used.

RAID devices provide you with redundancy—if a disk in a RAID system fails, you can immediately and automatically reconstruct the data on the failed disk from the data on the rest of the devices. RAID systems are ubiquitous, and most Oracle databases employ them for the several performance and redundancy benefits they provide.

When it comes to the performance of disk systems, two factors are of interest: the transfer rate and the number of I/O operations per second. The *transfer rate* refers to the efficiency with which data can move through the disk system's controller. As for I/O operations, the more a disk system can handle in a specified period, the better.

Compared to traditional disks, which have an MTBF of tens of thousands of hours, disk arrays have an MTBF of millions of hours. Even when a disk in a RAID system fails, the array itself continues to operate successfully. Most modern arrays automatically start using one of the spare disks, called *hot spares,* to which the data from the failed drive is transferred. Most disk arrays also permit the replacement of failed disks without bringing the system itself down (this is known as *hot swapping*).

RAID Levels

The inherent trade-off in RAID systems is between performance and reliability. You can employ two fundamental techniques, striping and mirroring the disk arrays, to improve disk performance and enhance reliability.

Mirroring schemes involve complete duplication of the data, and while most of the nonmirrored RAID systems also involve redundancy, it is not as high as in the mirrored systems. The redundancy in nonmirrored RAID systems is due to the fact that they store the necessary parity information needed for reconstructing disks in case there is a malfunction in the array.

The following sections describe the most commonly used RAID classifications. Except for RAID 0, all the levels offer redundancy in your disk storage system.

RAID 0: Striping

Strictly speaking, this isn't really a RAID level, since the striping doesn't provide you with any data protection whatsoever. The data is broken into *chunks* and placed across several disks that make up the disk array. The *stripe* here refers to the set of all the chunks.

Let's say the chunk or stripe size is 8KB. If we have three disks in our RAID and 24KB of data to write to the RAID system, the first 8KB would be written to the first disk, the second 8KB would be written to the second disk, and the final 8KB would be written to the last disk.

Because input and output are spread across multiple disks and disk controllers, the throughput of RAID 0 systems is quite high. For example, you could write an 800KB file over a RAID set of eight disks with a stripe size of 100KB in roughly an eighth of the time it would take to do the same operation on a single disk. However, because there is no built-in redundancy, the loss of a single drive could result in the loss of all the data, as data is stored sequentially on the chunks. RAID 0 is all about performance, with little attention paid to protection. Remember that RAID 0 provides you with zero redundancy.

RAID 1: Mirroring

In RAID 1, all the data is duplicated, or mirrored, on one or more disks. The performance of a RAID 1 system is slower than a RAID 0 system because input transactions are completed only when all the mirrored disks are successfully written to. The reliability of mirrored arrays is high, though, because the failure of one disk in the set doesn't lead to any data loss. The system continues operation under such circumstances, and you have time to regenerate the contents of the lost disks by copying data from the surviving disks. RAID 1 is geared toward protecting the data, with performance taking a back seat. Nevertheless, of all the redundant RAID arrays, RAID 1 still offers the best performance.

It is important to note that RAID 1 means that you will pay for *n* number of disks, but you get to allocate only *n*/2 disks for your system, because all the disks are duplicated.

Read performance improves under a RAID 1 system, because the data is scanned in parallel. However, there is slower write performance, amounting to anywhere from 10 to 20 percent, since both disks have to be written to each time.

RAID 2: Striping with Error Detection and Correction

RAID 2 uses striping with additional error detection and correction capabilities built in. The striping guarantees high performance, and error-correction methods are supposed to ensure reliability. However, the mechanism used to correct errors is bulky and takes up a lot of the disk space itself. This is a costly and inefficient storage system.

RAID 3: Striping with Dedicated Parity

RAID 3 systems are also striped systems, with an additional parity disk that holds the necessary information for correcting errors for the stripe. Parity involves the use of algorithms to derive values that allow the lost data on a disk to be reconstructed on other disks.

Input and output are slower on RAID 3 systems than on pure striped systems, such as RAID 0, because information also has to be written to the parity disk. RAID 3 systems can also only process one I/O request at a time.

Nevertheless, RAID 3 is a more sophisticated system than RAID 2, and it involves less overhead than RAID 2. You'll only need one extra disk drive in addition to the drives that hold the data. If a single disk fails, the array continues to operate successfully, with the failed drive being reconstructed with the help of the stored error-correcting parity information on the extra parity drive.

RAID 5 arrays with small stripes can provide better performance than RAID 3 disk arrays.

RAID 4: Modified Striping with Dedicated Parity

The stripes on RAID 4 systems are done in much larger chunks than in RAID 3 systems, which allows the system to process multiple I/O requests simultaneously. In RAID 4 systems, the individual disks can be independently accessed, unlike RAID 3 systems, which leads to much higher performance when reading data from the disks.

Writes are a different story, however, under this setup. Every time you need to perform a write operation, the parity data for the relevant disk must be updated before the new data can be written. Thus, writes are very slow, and the parity disk could become a bottleneck.

RAID 5: Modified Striping with Interleaved Parity

Under this disk array setup, both the data and the parity information are interleaved across the disk array. Writes under RAID 5 tend to be slower, but not as slow as under RAID 4 systems, because it can handle multiple concurrent write requests. Several vendors have improved the write performance by using special techniques, such as using nonvolatile memory for logging the writes.

RAID 5 gives you virtually all the benefits of striping (high read rates), while providing the redundancy needed for reliability, which RAID 0 striping does not offer.

RAID 0+1: Striping and Mirroring

These RAID systems provide the benefits of striped and mirrored disks. They tend to achieve a high degree of performance because of the striping, while offering high reliability due to the fact that all disks are mirrored (duplicated). You just have to be prepared to request double the number of disks you actually need for your data, because you are mirroring all the disks.

Choosing the Ideal Disk Configuration

Table 3-6 outlines the basic conclusions you can draw about the various RAID systems described in the preceding sections.

Table 3-6. *Benefits and Disadvantages of Different RAID Systems*

System	Benefits	Disadvantages
RAID 0	Offers high read and write performance and is cheap	Not very reliable (no redundancy)
RAID 1	Provides 100 percent redundancy	Expensive, and all writes must be duplicated
RAID 2		Expensive and wastes a lot of space for overhead; it is not commercially viable because of special disk requirements
RAID 3	Provides the ability to reconstruct data when only one disk fails (if two disks fail at the same time, there will be data loss)	Expensive and has poor random access performance
RAID 4	Provides the ability to reconstruct data when only one disk fails (if two disks fail at the same time, there will be data loss)	Expensive and leads to degraded write performance as well as a potential parity bottleneck
RAID 5	Offers high reliability and provides the ability to reconstruct data when only one disk fails (if two disks fail at the same time, there will be data loss)	Involves a write penalty, though it is smaller than in RAID 4 systems
RAID 0+1	Offers great random access performance as well as high transfer rates	Expensive (due to the mirroring of the disks)

What's the best strategy in terms of disk configuration? You, the DBA, and your system administrator should discuss your data needs, management's business objectives, the impact and cost of downtime, and available resources. The more complex the configuration, the more you need to spend on hardware, software, and training.

The choice essentially depends upon the needs of your organization. If your database needs the very highest possible performance and reliability at the same time, you may want to go first class and adopt the RAID 0+1 system. This is an expensive way to go, but several companies in critical data-processing areas, such as airline reservations systems, have adopted this as a company standard for data storage.

If data protection is your primary concern, however, and you can live with a moderate throughput performance, you can go with the RAID 5 configuration and save a lot of money in the process. This is especially true if read operations constitute the bulk of the work done by your database.

If you want complete redundancy and the resulting data protection, you can choose to use the RAID 1 configuration, and if you are concerned purely with performance and your data can be reproduced easily, you'll be better off just using a plain vanilla RAID 0 configuration.

To make the right choice, find out the exact response-time expectations for your databases, your finances, the nature of your applications, availability requirements, performance expectations, and growth patterns.

■ **Caution** Once you configure a certain RAID level on your disk, you can't easily switch to a different configuration. You have to completely reload all your applications and the databases if you decide to change configurations.

In general, the following guidelines will serve you well when you are considering the RAID configuration for your disks:

- RAID 5 offers many advantages over the other levels of RAID. The traditional complaint about the "write penalty" should be discounted because of sophisticated advances in write caches and other strategies that make RAID 5 much more efficient than in the past. The RAID 5 implementations using specialized controllers are far more efficient than software-based RAID or RAID 5 implementations based on the server itself. Using write caches in RAID 5 systems improves the overall write performance significantly.
- Allow for a lot more raw disk space than you figure you'll need. This includes your expansion estimates for storage space. Fault tolerance requires more disks under RAID systems than other systems. If you need 400GB of disk space, and you are using a RAID 5 configuration, you will need seven disks, each with 72GB storage capacity. One of the seven drives is needed for writing parity information. If you want to have a hot spare on the system, you would need a total of eight disks.
- Stripe widths depend on your database applications. If you are using OLTP applications, you need smaller stripe sizes, such as 128KB per stripe. Data warehouses benefit from much larger stripe sizes.
- Know your application. Having a good idea about what you are trying to achieve with the databases you are managing will help you decide among competing RAID alternatives.
- Always have at least one or two hot spares ready on the storage systems.

Redundant Disk Controllers

If you have a RAID 5 configuration, you are still vulnerable to a malfunction of the disk controllers. To avoid this, you can configure your systems in a couple of different ways. First, you can mirror the disks on different controllers. Alternatively, you can use redundant pairs of disk controllers, where the second controller takes over automatically by using an alternative path if the first controller fails for some reason.

IMPLEMENTING RAID

You can implement RAID in a number of ways. You could make a fundamental distinction between software-based and hardware-based RAID arrays.

Software RAID implementation uses the host server's CPU and memory to send RAID instructions and I/O commands to the arrays. Software RAID implementations impose an extra burden on the host CPU, and when disks fail, the disks with the operating system may not be able to boot if you are using a software-based RAID system.

Hardware RAID uses a special RAID controller, which is usually external to the server—host-based controllers can also be used to provide RAID functionality to a group of disks, but they are not as efficient as external RAID controllers.

RAID and Backups

Suppose you have a RAID 0+1 or a RAID 5 data storage array, which more or less ensures that you are protected adequately against disk failure. Do you still need database backups? Of course you do!

RAID systems mainly protect against one kind of failure involving disks or their controllers. But what about human error? If you or your developers wipe out data accidentally, no amount of disk mirroring is going to help you—you need those backups with the good data on them. Similarly, when a disaster such as a fire destroys your entire computer room, you need to fall back upon reliable and up-to-date offsite backups. Do not neglect the correct and timely backing up of data, even though you may be using the latest disk storage array solution.

RAID systems, it must be understood, do not guarantee nonstop access to your mission-critical data. The way to ensure that is to go beyond the basic RAID architecture and build a system that is disaster-tolerant.

RAID and Oracle

Oracle uses several different kinds of files as part of its database. You may need a combination of several of the RAID configurations to optimize the performance of your database while keeping the total cost of the disk arrays reasonable. An important thing to remember here is that when you use a RAID 3 or RAID 5 system, there is no one-to-one correspondence between the physical disks in the array and the logical disks, or logical unit numbers (LUNs), that are used by your system administrator to create logical volumes, which are in turn mounted for your file systems. Advise your system administrator to try and create as many logical volumes on each LUN as there are physical drives in the LUN. This way, the Oracle optimizer will have a more realistic idea about the physical disk setup that the database is using. Logical volumes are deceptive and could mislead the optimizer.

New Storage Technologies

Today's storage technologies are vastly superior to the technologies of even five years ago. Disk drives themselves have gotten faster—it is not difficult to find disks with 10,000 RPM and 15,000 RPM spindle speeds today. These disks have seek speeds of about 3.5 milliseconds.

In addition, advanced SCSI interfaces and the increasing use of fiber channel interfaces between servers and storage devices have increased data transfer rates to 100MB per second and faster. The capacity of individual disks has also risen considerably, with 180GB disks being fairly common today. The average MTBF for these new-generation disks is also very high—sometimes more than a million hours.

New technological architectures for data storage take advantage of all the previous factors to provide excellent storage support to today's Oracle databases. Two such storage architectures are Storage Area Networks (SANs) and Network Attached Storage Systems (NASs). Let's take a closer look at these storage architectures.

Storage Area Networks

Today, large databases are ubiquitous, with terabyte (1,000GB) databases not being a rarity any longer. Organizations tend to not only have several large databases for their OLTP work, but also use huge data warehouses and data marts for supporting management decision making. Storage Area Networks (SANs) use high-performance connections and RAID storage techniques to achieve the high performance and reliability that today's information organizations demand.

Modern data centers use SANs to optimize performance and reliability. SANs can be very small or extremely large, and they lend themselves to the latest technologies in disk storage and network communications. Traditionally, storage devices were hooked up to the host computer through a SCSI device. SANs can be connected to servers via high-speed fiber channel technology with the help of switches and hubs. You can adapt legacy SCSI-based devices for use with a SAN, or you can use entirely new devices specially designed for the SAN. A SAN is enabled by the use of fiber channel switches called *brocade switches*. By using hubs, you can use SANs that are several miles away from your host servers.

The chances are that if you are not using one already, you'll be using a SAN in the very near future. SANs offer many benefits to an organization. They allow data to be stored independently of the servers that run the databases and other applications. They enable backups that do not affect the performance of the network. They facilitate data sharing among applications.

SANs are usually preconfigured, and depending on your company's policy, they could come mirrored or as a RAID 5 configuration. The individual disks in the SANs are not directly controllable by the UNIX system administrator, who will see the LUN as a single disk—the storage array's controllers map the LUNs to the underlying physical disks. The administrator can use LVMs to create file systems on these LUNs after incorporating them into volume groups first.

When you use RAID-based storage arrays, the RAID controllers on the SAN will send the server I/O requests to the various physical drives, depending on the mirroring and parity level chosen.

Networked Attached Storage

Put simply, Networked Attached Storage (NAS) is a black box connected to your network, and it provides additional storage. The size of a NAS box can range from as small as 2GB up to terabytes of storage capacity.

The main difference between a NAS and a SAN is that it is usually easier to scale up a SAN's base storage system using the software provided by your supplier. For example, you can easily combine several disks into a single volume in a SAN. A NAS is set up with its own address, thus moving the storage devices away from the servers onto the NAS box. The NAS communicates with and transfers data to client servers using protocols such as the Network File System (NFS).

The NAS architecture is really not very suitable for large OLTP databases. One of the approaches now being recommended by many large storage vendors for general storage as well as for some databases is to combine the SAN and NAS technologies to have the best of both worlds.

■ **Note** A good paper comparing the RAID and SAN technologies is "Comparison of Performance of Competing Database Storage Technologies: NetApp Storage Networking vs. Veritas RAID," by Dan Morgan and Jeff Browning (http://www.netapp.com/tech_library/3105.html). This article is slightly dated, as the article's authors used Oracle8 for the tests, but it still provides a useful comparison of the technologies.

InfiniBand

One of the latest network technologies is InfiniBand, a standards-based alternative to Ethernet that seeks to overcome the limitations of TCP/IP-based networks. One of the driving forces behind network storage is to reduce the I/O bottlenecks between the CPU and the disks. InfiniBand takes another approach and works between a host channel controller on the server and a special adapter on the storage machines or device, thereby not requiring an I/O bus. A single link can operate at 2.5GB per second. InfiniBand provides higher throughput, and lower latency and CPU usage than normal TCP/IP and Ethernet solutions. You can find a full discussion of this new technology at http://www.infinibandta.org/ibta/.

Given the high-profile companies involved in developing this concept (Microsoft, IBM, Sun, HP, and some of the main storage vendors), you can expect to see considerable push in the storage area. InfiniBand supports its own protocol, called Sockets Direct Protocol (SDP).

Oracle Database 10*g* and the New Automatic Storage Management

Remember that whatever RAID configuration you use, or however you use the Logical Volume Manger tools to stripe or mirror your disks, it's the operating system that's ultimately in charge of managing your data files. Whenever you need to add or move your data files, you have to rely on operating system file-manipulation commands. Oracle overcomes the raw device limits and partition limits by using its Clustered File System, while avoiding the performance hits associated with SANs.

Oracle Database 10*g* introduces the innovative Automatic Storage Management (ASM) feature, which for the first time provides the Oracle DBA with the option (option, because you don't have to use the ASM) of managing the database data files directly, bypassing the underlying operating system. When you use ASM, you don't have to manage disks and data files directly. You deal with disk groups instead, which consist of several disk drives. Disk groups make it possible for you to avoid having to deal with filenames when you manage the database.

Using ASM is like having Oracle's own built-in logical manager manage your disks and file systems. ASM lets you dynamically reorganize your disk storage and perform rebalancing operations to avoid I/O contention. If you're spending a significant proportion of your time managing disks and file systems, it's time to switch to the far more efficient ASM system.

Chapter 17 shows you how to use the powerful ASM feature.

Oracle and Storage System Compatibility

Oracle Corporation actively works with vendors to ensure that the storage arrays and other technologies are compatible with its own architectural requirements. Oracle manages a vendor-oriented certification program called the Oracle Storage Compatibility Program (OSCP). As part of the OSCP, Oracle provides test suites for vendors to ensure their products are compatible with Oracle Database 10*g*. In this certification program, vendors normally test their storage systems on several platforms, including several variants of the UNIX operating system, Linux, and Windows.

Oracle has also been responsible for the Hardware Assisted Resilient Data (HARD) initiative. HARD's primary goal is to prevent data corruption and thus ensure data integrity. The program includes measures to prevent the loss of data by validating the data in the storage devices. RAID devices do help protect the physical data, but the HARD initiative seeks to protect the data further by ensuring that it is valid and is not saved in a corrupted format. Availability and protection of data are enhanced because data integrity is ensured through the entire pipeline, from the database to the hardware. Oracle Database 10*g* does have its own corruption-detecting features, but the HARD initiative is designed to prevent data corruption that could occur as you move data between various operating system and storage layers. For example, EMC Corporation's solution to comply with the HARD Initiative involves checking the checksums of data when they reach their storage devices, and comparing them with the Oracle checksums. Data will be written to disk only if the two checksums are identical.

■ **Note** New technologies have come to the fore in recent years that enable businesses to operate on a 24/7 basis as well as to provide data protection. Backup windows are considerably reduced by the use of these new technologies, which enable nondisruptive backup operations. These backup technologies include the *clone* or *snapshot* techniques, which enable a quick copy to be made of the production data onto a different server. Compaq's SANworks Enterprise Volume Manager, Hewlett-Packard's Business Copy, Fujitsu's Remote Equivalent Copy, and Sun's StorEdge Instant Image all allow data copying between Oracle databases at a primary site to databases at remote locations. The key thing to remember is that these techniques take snapshots of live data in very short time periods, so these techniques can be used for backup purposes as well as for disaster recovery.
