# CHAPTER – 6

# SECURITY AND SYSTEM HANDLING

## UNDERSTANDING SHELL SCRIPTS:

Shell scripts are the equivalent of batch files in MS-DOS, and can contain long lists of commands, complex flow control, arithmetic evaluations, user-defined variables, user-defined functions, and sophisticated condition testing. Shell scripts are capable of handling everything from simple one-line commands to something as complex as starting up our Red Hat Linux system.

Red Hat Linux system uses shell scripts (/etc/rc.d/rc.sysint and /etc/rc) to check and mount all our filesystems, set up our consoles, configure our network, launch all our system services, and eventually provide us with our login screen. While there are nearly a dozen different shells available in Red Hat Linux, the default shell is called **bash, the Bourne-Again shell**.

### ☑ EXECUTING AND DEBUGGING SHELL SCRIPTS:

One of the primary advantages of shell scripts is that they can be opened in any text editor to see what they do. A big disadvantage is that shell scripts often execute more slowly than compiled programs. There are two basic ways to execute a shell script:

- The filename is used as an argument to the shell (as in bash myscript). In this method, the file does not need to be executable; it just contains a list of shell commands. The shell specified on the command line is used to interpret the commands in the script file. This is most common for quick, simple tasks.

- The shell script may also have the name of the interpreter placed in the first line of the script preceded by a #! (as in #!/bin/bash), and have its execute bit set (using chmod +x). We can then run our script just like any other program in our path simply by typing the name of the script on the command line.

When scripts are executed in either manner, options to the program may be specified on the command line. Anything following the name of the script is referred to as a *command-line argument*.

As with writing any software, there is no substitute to clear and thoughtful design and lots of comments. The pound sign (#) prefaces comments and can take up an entire line or exist on the same line as script code. It's best to implement more complex shell scripts in stages, making sure the logic is sound at each step before continuing. Here are a few ways to make sure things are working as expected during testing:

- Place an echo statement at the beginning of lines within the body of a loop. That way, rather than executing the code, we can see what will be executed without making any permanent changes.

- To achieve the same goal, we could place dummy echo statements throughout the code. If these lines get printed, we know the correct logic branch is being taken.

- We could use set +x near the beginning of the script to display each command that is executed or launch our scripts using sh –x *myscript*.

### ☑ Understanding Shell Variables:

Often within a shell script, we want to reuse certain items of information. During the course of processing the shell script, the name or number representing this information may change. To store information used by a shell script in a way that it can be easily reused, we can set variables. Variable names within shell scripts are case-sensitive and can be defined in the following manner:

```
NAME=value
```

The first part of a variable is the variable name, and the second part is the value set for that name. Variables can be assigned from constants, like text or numbers. This is useful for initializing values or saving lots of typing for long constants. Here are examples where variables are set to a string of characters (CITY) and a numeric value (PI):

```
CITY="Springfield"
PI=3.14159265
```

Variables can contain the output of a command or command sequence. We can accomplish this by either enclosing the command in backticks (`) or by enclosing the command in parentheses. This is a great way to get information that can change from computer to computer or from day to day. Here we set the output of the uname -n command to the MACHINE variable. Then we use parens to set NUM_FILES to the number of files in the current directory by piping ( | ) the output of the ls command to the word count command (wc -l).

```
MACHINE=`uname –n`
NUM_FILES=(/bin/ls | wc –l)
```

Variables can also contain the value of other variables. This is useful when we have to preserve a value that will change so we can use it later in the script. Here BALANCE is set to the value of the CurBalance variable.

```
BALANCE=$CurBalance
```

***Note:*** *When assigning variables, use only the variable name (for example, BALANCE). When referenced, meaning we want the value of the variable, precede it with a dollar sign (as in $CurBalance).*

### ☑ Special Shell Variables:

There are special variables that the shell assigns for us. The most commonly used are called the *positional parameters* or *command line arguments* and are referenced as $0, $1, $2, $3…$n. $0 is special and is assigned the name used to invoke our script; the remainder are assigned the values of the parameters passed on the command line. For instance, if the shell script named myscript were called as:

```
myscript foo bar
```

The positional parameter $0 would be myscript, $1 would be foo, and $2 would be bar.

Another variable, $#, tells us how many parameters our script was given. In our example, $# would be 2. Another particularly useful special shell variable is $?, which receives the exit status of the last command executed. Typically, a value of zero means everything is okay, and anything other than zero indicates an error of some kind. For a complete list of special shell variables, refer to the bash man page.

***Parameter Expansion in Bash:***

As mentioned earlier, if we want the value of a variable, we precede it with a $ (for example, $CITY). This is really just a shorthand for the notation ${CITY}. Bash has special rules that allow us to expand the value of a variable in different ways. Going into all the rules is probably a little overboard for a quick introduction to shell scripts, but Table below presents some common constructs that we're likely to see in bash scripts we find on our Red Hat Linux box.

| Examples of bash parameter expansion | |
|---|---|
| **Construction** | **Meaning** |
| **${var:-value}** | If variable is unset or empty, expand this to value |
| **${var#pattern}** | Chop the shortest match for pattern from the end of var's value |
| **${var##pattern}** | Chop the longest match for pattern from the end of var's value |
| **${var%pattern}** | Chop the shortest match for pattern from the front of var's value |
| **${var%%pattern}** | Chip the longest match for pattern from the front of var's value |

Try typing the following commands from a shell to test out how parameter expansion works:

```
# THIS="Example"
# THIS=${THIS:-"Not Set"}
# THAT=${THAT:-"Not Set"}
# echo $THIS
Example
# echo $THAT
Not Set
```

In the previous examples, the THIS variable is set to the word Example. In the next two lines, the THIS and THAT variables are set to their current values or to Not Set, if they are not currently set. Notice that because we just set THIS to the string Example, when we echo the value of THIS it appears as Example. However, since THAT was not set, it appears as Not Set.

***Note:*** *For the rest of this section, we show how variables and commands may appear in a shell script. To try out any of those examples, however, we can simply type them into a shell as shown in the previous example.*

In the following example, we set MYFILENAME to /home/digby/myfile.txt. Next, the FILE variable is set to myfile.txt and DIR is set to /home/digby. In the NAME variable, the file name is cut down to simply myfile, then in the EXTENSION variable the file extension is set to txt. (To try these out, we can type them at a shell prompt as we did with the previous example, then echo the value of each variable to see how it is set.)

```
MYFILENAME="/home/digby/myfile.txt"
FILE=${MYFILENAME##*/}        #FILE becomes "myfile.txt"
DIR=${MYFILENAME%/*}          #DIR becomes "/home/digby
NAME=${FILE%.*}               #NAME becomes "myfile"
EXTENSION=${FILE#*.}          #EXTENSION becomes "txt"
```

### ☑ Performing Arithmetic in Shell Scripts:

Bash used *untyped* variables, meaning it normally treats variables as strings or text, but can change them on the fly if we want it to. Unless we tell it otherwise with declare, our variables are just a bunch of letters to bash. But when we start trying to do arithmetic with them, bash will convert them to integers if it can. This makes it possible to do some fairly complex arithmetic in bash.

Integer arithmetic can be performed using the built-in let command or through the external expr or bc commands. After setting the variable BIGNUM value to 1024, the three commands that follow would all store the value 64 in the RESULT variable:

```
BIGNUM=1024
let RESULT=$BIGNUM/16
RESULT=`expr $BIGNUM / 16`
RESULT=`echo "$BUGNUM / 16" | bc -l`
```

***Note:*** *While most elements of shell scripts are relatively freeform (where white space, such as spaces or tabs, is insignificant), both let and expr are particular about spacing. The let command insists on no spaces between each operand and the mathematical operator, whereas the syntax of the expr command requires white space between each operand and its operator. In contrast to those, bc isn't picky about spaces, but can be trickier to use because it does floating-point arithmetic.*

To see a complete list of the kinds of arithmetic we can perform using the let command, type **help let** at the bash prompt.

### ☑ Using Programming Constructs In Shell Scripts:

One of the features that make shell scripts so powerful is that their implementation of looping and conditional execution constructs similar to those found in more complex scripting and programming languages. We can use several different types of loops, depending on our needs.

- ***The "if... then" statements:***

The most commonly used programming construct is conditional execution, or the "if" statement. It is used to perform actions only under certain conditions. There are several variations, depending on whether we're testing one thing, or want to do one thing if a condition is true, but another thing if a condition is false, or if we want to test several things one after the other.

The first if/then example tests if VARIABLE is set to the number 1. If it is, then the echo command is used to say that it is set to 1.

```
if [ $VARIABLE -eq 1 ] ; then
echo "The variable is 1"
fi
```

Instead of using –eq, we can use the equals sign (=), as shown in the following example. Using the else statement, different words can be echoed if the criterion of the if statement isn't met ($STRING = "Friday").

```
if [ $STRING = "Friday" ] ; then
echo "WhooHoo!  Friday!"
else
echo "Will Friday ever get here?"
fi
```

We can also reverse tests with an exclamation mark (!). In the following example, if STRING is not Monday, then "At least it's not Monday" is echoed.

```
if [ $STRING != "Monday" ] ; then
   echo "At least it's not Monday"
fi
```

In the following example, elif is used to test for an additional condition (is filename a file or a directory).

```
if [ -f $filename ] ; then
   echo "$filename is a regular file"
elif [ -d $filename ] ; then
   echo "$filename is a directory"
else
   echo "I have no idea what $filename is"
fi
```

As we can see from the preceding examples, the condition we are testing is placed between square brackets [ ]. When a test expression is evaluated, it will return either a value of 0, meaning

that it is true, or a 1, meaning that it is false. Table below lists the conditions that are testable and is quite a handy reference. (If we're in a hurry, we can type **help test** on the command line to get the same information.)

| Operator | What Is Being Tested? |
|---|---|
| **-a file** | Does the file exist? (same as –e) |
| **-b file** | Is the file a special block device? |
| **-c file** | Is the file character special (e.g., a character device)? Used to identify serial lines and terminal devices. |
| **-d file** | Is the file a directory? |
| **-e file** | Does the file exist? (same as -a) |
| **-f file** | Does the file exist, and is it a regular file (e.g., not a directory, socket, pipe, link, or device file)? |
| **-g file** | Does the file have the set-group-id bit set? |
| **-h file** | Is the file a symbolic link? (same as –L) |
| **?** | ? |
| **-k file** | Does the file have the sticky bit set? |
| **-L file** | Is the file a symbolic link? |
| **-n string** | Is the length of the string greater than 0 bytes? |
| **-O file** | Do you own the file? |
| **-p file** | Is the file a named pipe? |
| **-r file** | Is the file readable by you? |
| **-s file** | Does the file exist, and is it larger than 0 bytes? |
| **-S file** | Does the file exist, and is it a socket? |
| **-t fd** | Is the file descriptor connected to a terminal? |
| **-u file** | Does the file have the set-user-id bit set? |
| **-w file** | Is the file writable by you? |
| **-x file** | Is the file executable by you? |
| **-z string** | Is the length of the string 0 (zero) bytes? |
| **expr1 -a expr2** | Are both the first expression and the second expression true? |
| **expr1 -o expr2** | Is either of the two expressions true? |
| **file1 -nt file2** | Is the first file newer than the second file (using the modification timestamp)? |
| **file1 -ot file2** | Is the first file older than the second file (using the modification timestamp)? |
| **file1 -ef file2** | Are the two files associated by a link (a hard link or a symbolic link)? |
| **var1 = var2** | Is the first variable equal to the second variable? |
| **var1 -eq var2** | Is the first variable equal to the second variable? |
| **var1 -ge var2** | Is the first variable greater than or equal to the second variable? |
| **var1 -gt var2** | Is the first variable greater than the second variable? |
| **var1 -le var2** | Is the first variable less than or equal to the second variable? |
| **var1 -lt var2** | Is the first variable less than the second variable? |
| **var1 != var2** | Is the first variable not equal to the second variable? |
| **var1 -ne var2** | Is the first variable not equal to the second variable? |

There is also a special shorthand method of performing tests that can be useful for simple *one-command* actions. In the following example, the two pipes (||) indicate that if the directory being tested for doesn't exist (-d dirname), then make the directory (mkdir $dirname).

```
# [ test ] || {action}
# Perform simple single command {action} if test is false
[ -d $dirname ] || mkdir $dirname
```

Instead of pipes, two ampersands can be used to test if something is true. In the example below, a command is being tested to see if it includes at least three command line arguments.

```
# [ test ] && {action}
# Perform simple single command {action} if test is true
[ $# -ge 3 ] && echo "There are at least 3 command line arguments."
```

- ***The case command:***

Another frequently used construct is the case command. Similar to a switch statement in programming languages, this can take the place of several nested if statements. A general form of the case statement is as follows:

```
case "VAR" in
   Result1)
      { body };;
   Result2)
      { body };;
   *)
      { body } ;;
esac
```

One use for the case command might be to help with our backups. The following case statement tests for the first three letters of the current day (case `date +%a` in). Then, depending on the day, a particular backup directory (BACKUP) and tape drive (TAPE) is set.

```
# Our VAR doesn't have to be a variable,
# it can be the output of a command as well
# Perform action based on day of week
case `date +%a` in
   "Mon")
         BACKUP=/home/myproject/data0
         TAPE=/dev/rft0
# Note the use of the double semi-colon to end each option
```

```
        ;;
# Note the use of the "|" to mean "or"
   "Tue" | "Thu")
        BACKUP=/home/myproject/data1
        TAPE=/dev/rft1
        ;;
   "Wed" | "Fri")
        BACKUP=/home/myproject/data2
        TAPE=/dev/rft2
        ;;
# Don't do backups on the weekend.
   *)
        BACKUP="none"
        TAPE=/dev/null
        ;;
esac
```

The asterisk (*) is used as a catchall, similar to the default keyword in the C programming language. In this example, if none of the other entries are matched on the way down the loop, the asterisk is matched, and the value of BACKUP becomes none.

- ***The "for . . . do" loop:***

Loops are used to perform actions over and over again until a condition is met or until all data has been processed. One of the most commonly used loops is the for . . . do loop. It iterates through a list of values, executing the body of the loop for each element in the list. The syntax and a few examples are presented here:

```
for VAR in LIST
do
    { body }
done
```

The for loop assigns the values in LIST to VAR one at a time. Then for each value, the body in braces between do and doneis executed. VAR can be any variable name, and LIST can be composed of pretty much any list of values or anything that generates a list.

```
for NUMBER in 0 1 2 3 4 5 6 7 8 9
do
   echo The number is $NUMBER
done
```

```
for FILE in `/bin/ls`
do
   echo $FILE
done

# You can also write it this way, which is somewhat cleaner.
for NAME in John Paul Ringo George ; do
   echo $NAME is my favorite Beetle
done
```

Each element in the LIST is separated from the next by white space. This can cause trouble if we're not careful, since some commands, like ls –l, output multiple fields per line, each separated by whitespace.

If we're a die-hard C programmer, bash allows us to use C syntax to control our loops.

```
LIMIT=10
# Double parentheses, and no $ on LIMIT even though it's a variable!
for ((a=1; a <= LIMIT ; a++)) ; do
  echo  "$a"
done
```

- ***The "while . . . do" and "until . . . do" loops:***

Two other possible looping constructs are the while. . . do loop and the until. . . do loop. The structure of each is presented here:

```
while condition      until condition
do                   do
   { body }            { body }
done                 done
```

Here is an example of a while loop that will output the number 0123456789:

```
N=0
while [ $N –lt 10 ] ; do
   echo –n $N
   let N=$N+1
done
```

Another way to output the number 0123456789 is to use an until loop as follows:

```
N=0
until [ $N –eq 10 ] ; do
   echo –n $N
   let N=$N+1
done
```

### ☑ Some Useful External Programs:

Bash is great and has lots of built-in commands, but it usually needs some help to do anything really useful. Some of the most common useful programs we'll see used are grep, cut, tr, and sed. Like all the best UNIX tools, most of these programs are designed to work with standard input and standard output, so we can easily use them with pipes and shell scripts.

#### ➢ ***The General Regular Expression Parser (grep):***

The name sounds intimidating, but grep is just a way to find patterns in files or text. Think of it as a useful search tool. Getting really good with regular expressions is quite a challenge, but many useful things can be accomplished with just the simplest forms.

For example, we can display a list of all regular user accounts by using grep to search for all lines that contain the text /homein the /etc/passwd file as follows:

```
grep /home /etc/passwd
```

Or we could find all environment variables that begin with HO using the following command:

```
env | grep ^HO
```

To find a list of options to use with the grep command, type **man grep**.

#### ➢ ***Remove Sections Of Lines Of Text (cut):***

The cut command can extract specific fields from a line of text or from files. It is very useful for parsing system configuration files into easy-to-digest chunks. We can specify the field separator we want to use and the fields we want, or we can break a line up based on bytes.

The following example lists all home directories of users on our system. Using an earlier example of the grep command, this line pipes a list of regular users from the /etc/passwd file, then displays the sixth field (-f6) as delimited by a colon (-d':').

```
grep /home /etc/passwd | cut –f6 –d':'
```

#### ➢ ***Translate Or Delete Characters (tr):***

The tr command is a character-based translator that can be used to replace one character or set of characters with another or to remove a character from a line of text.

The following example translates all upper-case letters to lowercase letters and displays the words "mixed upper and lower case" as a result:

```
FOO="Mixed UPpEr aNd LoWeR cAsE"
echo $FOO | tr [A-Z] [a-z]
```

In this example, the tr command is used on a list of file names to rename any files in that list so that any spaces contained in a file name are translated into underscores:

```
for file in * ; do
   d=`echo $file | tr [:blank:] [_]`
   [ "$file" -eq "-d" ] || mv "$file" "$d"
done
```

- ***The Stream Editor (sed):***

The sed command is a simple scriptable editor, and as such can perform only simple edits, such as removing lines that have text matching a certain pattern, replacing one pattern of characters with another, and other simple edits. To get a better idea of how sed scripts work, there's no substitute for the online documentation, but here are some examples of common uses.

We can use the sed command to essentially do what we did earlier with the grep example: search the /etc/passwd file for the word home. Here the sed command searches the entire /etc/passwd file, searches for the word home, and prints any line containing the word home.

```
sed –n –e '/home/p' /etc/passwd
```

In this example, sed searches the file somefile.txt and replaces every instance of the string Mac with Linux. The output is then sent to the fixed_file.txt file.

```
sed –e 's/Mac/Linux/' somefile.txt > fixed_file.txt
```

We can get the same result using a pipe:

```
cat somefile.txt | sed –e 's/Mac/Linux/' > fixed_file.txt
```

By searching for a pattern and replacing it with a null pattern, we delete the original pattern. This example searches the contents of the somefile.txt file and replaces extra blank spaces at the end of each line (s/ *$) with nothing (//). Results go to the fixed_file.txt file.

```
cat somefile.txt | sed –e 's/ *$//` > fixed_file.txt
```

- ☑ **TRYING SOME SIMPLE SHELL SCRIPTS**

Sometimes the simplest of scripts can be the most useful. If you type the same sequence of commands repetitively, it makes sense to store those commands (once!) in a file. Here are a couple of simple, but useful, shell scripts.

- ***A Simple Telephone List:***

This idea has been handed down from generation to generation of old UNIX hacks. It's really quite simple, but it employs several of the concepts just introduced.

```
#!/bin/bash
# (@)/ph
# A very simple telephone list
# Type "ph new name number" to add to the list, or
# just type "ph name" to get a phone number

PHONELIST=~/.phonelist.txt

# If no command line parameters ($#), there
# is a problem, so ask what they're talking about.
if [ $# -lt 1 ] ; then
   echo "Whose phone number did you want?"
   exit 1
fi
# Did you want to add a new phone number?
if [ "$1" = "new" ] ; then
   shift
   echo $* >> $PHONELIST
   echo $* added to database
   exit 0
fi
# Nope. But does the file have anyting in it yet?
# This might be our first time using it, after all.
if [ ! -s $PHONELIST ] ; then
   echo "No names in the phone list yet!"
   exit 1
else
   grep –q "$*" $PHONELIST     # Quietly search the file
   if [ $? –ne 0 ] ; then          # Did we find anything?
      echo "Sorry, that name was not found in the phone list"
      exit 1
   else
      grep "$*" $PHONELIST
```

```
   fi
fi
exit 0
```

- ***A Simple Backup Script:***

Since nothing works forever and mistakes happen, backups are just a fact of life when dealing with computer data. This simple script will back up all the data in the home directories of all the users on your Red Hat Linux system.

```
#!/bin/bash
# (@)/my_backup
# A very simple backup script
#
TAPE=/dev/rft0
# Rewind the tape device $TAPE
mt $TAPE rew
# Get a list of home directories
HOMES=`grep /home /etc/passwd | cut –f6 –d';'`
# Backup the data in those directories
tar cvf $TAPE $HOMES
# Rewind and eject the tape.
mt $TAPE rewoffl
```

## System Start Up and Shut Down:

During system start-up, a series of scripts are run to start the services that we need. These include scripts to start network interfaces, mount directories, and monitor our system. Most of these scripts are run from subdirectories of /etc/rc.d. The program that starts most of these services up when we boot and stops them when we shut down is the /etc/rc.d/rc script. The following sections describe run-level scripts and what we can do with them.

### ☑ Starting Run-Level Scripts:

As previously mentioned, the /etc/rc.d/rc script is a script that is integral to the concept of run levels. Any change of run level causes the script to be executed, with the new run level as an argument. Here's a quick run-down of what the /etc/rc.d/rcscript does:

- **Checks that run level scripts are correct.???** The rc script checks to find each run level script that exists and excludes those that represent backup scripts left by rpm updates.
- **Determines current and previous run levels.???** Determines the current and previous run levels to know which run-level scripts to stop (previous level) and start (current level).

- **Decides whether to enter interactive startup.???** If the confirm option is passed to the boot loader at boot time, all server processes must be confirmed at the system console before starting.
- **Kills and starts run-level scripts.???** Stops run-level scripts from the previous level, then starts run-level scripts from the current level.

In Red Hat Linux, most of the services that are provided to users and computers on the network are started from run-level scripts.

### ☑ Understanding Run-Level Scripts:

A software package that has a service to start at boot time (or when the system changes run levels) can add a script to the /etc/init.d directory. That script can then be linked to an appropriate run-level directory and either be started or stopped (to start or stop the service).

Table below lists many of the typical run-level scripts that are found in /etc/init.d and explains their function. Depending on the Red Hat Linux software packages we installed on our system, we may have dozens more run-level scripts than we see here.

| Run-Level Scripts | What Does It Do? |
|---|---|
| **apmd** | Controls the Advanced Power Management daemon, which monitors battery status, and which can safely suspend or shut down all or part of a machine that supports it. |
| **atd** | Starts or stops the at daemon to receive, queue, and run jobs submitted via the at or batchcommands. (The anacron run-level script runs at and batch jobs that were not run because the computer was down.) |
| **autofs** | Starts and stops the automount daemon, for automatically mounting file systems (so, for example, a CD can be automatically mounted when it is inserted). |
| **crond** | Starts or stops the cron daemon to periodically run routine commands. |
| **cups-lpd** | Controls the printer daemon that handles spooling printing requests. |
| **dhcpd** | Starts or stops the dhcpd daemon, which automatically assigns IP addresses to computers on a LAN. |
| **gpm** | Controls the gpm daemon, which allows the mouse to interact with console- and text-based applications. |
| **halt** | Terminates all processes, writes out accounting records, removes swap space, unmounts all file systems, and either shuts down or reboots the machine (depending upon how the command was called). |
| **httpd** | Starts the httpd daemon, which allows our computer to act as an HTTP server (i.e., to serve Web pages). |
| **iptables** | Starts the iptables firewall daemon, which manages any iptables-style firewall rules set up for our computer. |
| **keytable** | Loads the predefined keyboard map. |
| **killall** | Shuts down any subsystems that may still be running prior to a shutdown or reboot. |
| **kudzu** | Detects and configures new hardware at boot time. |
| **netfs** | Mounts or unmounts network (NFS, SMB, and NCP) file systems. |

| | |
|---|---|
| **network** | Starts or stops all configured network interfaces and initializes the TCP/IP and IPX protocols. |
| **nfs** | Starts or stops the NFS-related daemons (rpc.nfsd, rpc.mountd, rpc.statd, and rcp.rquotad) and exports shared file systems. |
| **pcmcia** | Loads or unloads modules, drivers, and programs (including the cardmgr daemon) to support PCMCIA cards (Ethernet adapters, modems, memory cards, etc.) in laptop computers. |
| **portmap** | Starts or stops the portmap daemon, which manages programs and protocols that utilize the Remote Procedure Call (RPC) mechanism. |
| **random** | Loads or saves the current state of the machine's random number generator's random seed to ensure more random randomness. |
| **routed** | Starts or stops the routed daemon, which controls dynamic-routing table updates via the Router Information Protocol (RIP). |
| **rwhod** | Starts or stops the rwhod daemon, which enables others on the network to obtain a list of all currently logged-in users. |
| **sendmail** | Controls the sendmail daemon, which handles incoming and outgoing SMTP (Simple Mail Transport Protocol) mail messages. |
| **single** | Terminates all running processes and enters run level 1 (single-user mode). |
| **smb** | Starts or stops the smbd and nmbd daemons for allowing access to Samba file and print services. |
| **snmpd** | Starts or stops the snmpd (Simple Network Management Protocol) daemon, which enables others to view machine-configuration information. |
| **squid** | Starts or stops the squid services, which enables proxy service to clients on your network. |
| **syslog** | Starts or stops the klogd and syslogd daemons that handle logging events from the kernel and other processes, respectively. |
| **xfs** | Starts or stops xfs, the X Window font server daemon. |
| **xinetd** | Sets the machine's host name, establishes network routes, and controls xinetd, the network services daemon which listens for incoming TCP/IP connections to the machine. |
| **ypbind** | Binds to an NIS (Network Information Service) master server (if NIS is configured), and starts or stops the ypbind process, which communicates with the master server. |

Each script representing a service that we want to start or stop is linked to a file in each of the run-level directories. For each run level, a script beginning with K stops the service, whereas a script beginning with S starts the service.

The two digits following the K or S in the filename provide a mechanism to select the priority in which the programs are run. For example, S12syslog is run before S90crond. However, while humans can readily see that 85 is less than 110, the file S110my_daemon is run before S85gpm. This is because the "ASCII collating sequence orders the files," which simply means that one positional character is compared to another. Therefore, a script beginning with the characters S110 is executed between S10network and S15netfs in run level 3.

All of the programs within the /etc/rc*X*.d directories (where *X* is replaced by a run-level number) are symbolic links, usually to a file in /etc/init.d. The /etc/rc*X*.d directories include the following:

- /etc/rc0.d: Run level 0 directory
- /etc/rc1.d: Run level 1 directory
- /etc/rc2.d: Run level 2 directory
- /etc/rc3.d: Run level 3 directory
- /etc/rc4.d: Run level 4 directory
- /etc/rc5.d: Run level 5 directory
- /etc/rc6.d: Run level 6 directory

Inthismanner, /etc/rc0.d/K05atd, /etc/rc1.d/K05atd, /etc/rc2.d/K05atd, /etc/rc3.d/S95atd, /etc/rc4.d/S95atd, /etc/rc5.d/S95atd, and /etc/rc6.d/K05atd are all symbolic links to /etc/init.d/atd. Using this simple, consistent mechanism, we can customize which programs are started at boot time.

### ☑ Understanding What Startup Scripts Do:

Despite all the complicated rc*X*s, Ss, and Ks, the form of each start-up script is really quite simple. Because they are in plain text, we can just open one with a text editor to take a look at what it does. For the most part, a run-level script can be run with a start option, a stop option, and possibly a restart option. For example, the following lines are part of the contents of the smbscript that defines what happens when the script runs with different options to start or stop the Samba file and print service:

```
#!/bin/sh
#
# chkconfig: - 91 35
# description: Starts and stops the Samba smbd and nmbd daemons \
#              used to provide SMB network services.
        .
        .
        .
start() {
        KIND="SMB"
        echo -n $"Starting $KIND services: "
        daemon smbd $SMBDOPTIONS
        RETVAL=$?
        echo
        KIND="NMB"
        echo -n $"Starting $KIND services: "
        daemon nmbd $NMBDOPTIONS
        RETVAL2=$?
```

```
        echo
        [ $RETVAL -eq 0 -a $RETVAL2 -eq 0 ] && touch /var/lock/subsys/smb || \
           RETVAL=1
        return $RETVAL
}

stop() {
        KIND="SMB"
        echo -n $"Shutting down $KIND services: "
        killproc smbd
        RETVAL=$?
        echo
        KIND="NMB"
        echo -n $"Shutting down $KIND services: "
        killproc nmbd
        RETVAL2=$?
        [ $RETVAL -eq 0 -a $RETVAL2 -eq 0 ] && rm -f /var/lock/subsys/smb
        echo ""
        return $RETVAL
}

restart() {
        stop
        start
}
      .
      .
      .
```

To illustrate what this script essentially does, some of the beginning and end of the script (where it checked if the network was up and running and set some values) are skipped. Here are the actions smb takes when it is run with start or stop:

- **Start:** This part of the script starts the smbd and nmbd servers when the script is run with the start option.
- **Stop**: When run with the stop option, the /etc/init.d/smb script stops the smbd and nmbd servers.

The restart option runs the script with a stop option followed by a start option. If we want to start the smb service our self, we can type the following command (as root user):

```
# /etc/init.d/smb start
Starting SMB services:                 [ OK ]
Starting NMB services:                 [ OK ]
```

To stop the service, we can type the following command:

```
# /etc/init.d/smb stop
Shutting down SMB services:            [ OK ]
Shutting down NMB services:            [ OK ]
```

The smb run-level script is different from other run-level scripts in that it supports several other options than start and stop. For example, this script has options (not shown in the example) that allow us to reload the smb.conf configuration file (reload) and check the status of the service (rhstatus).

### ☑ Changing Run-Level Script Behavior:

Modifying the start-up behavior of any such script merely involves opening the file in a text editor. For example, the atddaemon queues jobs submitted from the at and batch commands. Jobs submitted via batch are executed only if the system load is below a particular value, which can be set with a command-line option to the atd command. The default value of 0.8 is based on the assumption that a single-processor machine with less than 80 percent CPU utilization could handle the additional load of the batch job. However, if we were to add another CPU to our machine, the default threshold value would be too low and the batch jobs would not be sufficiently restricted.

We can change the system load threshold value from 0.8 to 1.6 to accommodate the increased processing capacity. To do this, simply modify the following line (in the start section) of the /etc/init.d/atd script:

```
daemon /usr/sbin/atd
```

Replace it with this line, using the -l argument to specify the new minimum system load value:

```
daemon /usr/sbin/atd -l 1.6
```

After saving the file and exiting the editor, you can reboot the machine or just run any of the following three commands to begin using the new batch threshold value:

```
/etc/init.d/atd reload
/etc/init.d/atd restart
/etc/init.d/atd stop ; /etc/rc.d/init.d/atd start
```

***Note:*** *Always make a copy of a run-level script before we change it. Also, keep track of changes we make to run-level scripts before we upgrade the packages they come from. We need to make those changes again after the upgrade.*

If we are uncomfortable editing start-up scripts and we simply want to add options to the daemon process run by the script, there maybe a way of entering these changes without editing the start-up script directly. Check the /etc/sysconfig directory and see if there is a file by the same name as the script we want to modify. If there is, that file probably provides values that we can set to pass options to the start-up script. Sysconfig files exist for amd, arpwatch, dhcpd, kudzu, ntpd, samba, squid, and others.

### ☑ Reorganizing Or Removing Run-Level Scripts:

There are several ways to deal with removing programs from the system start-up directories, adding them to particular run levels, or changing when they are executed. From a Terminal window, we can use the chkconfig command. From a GUI, use the Service Configuration window.

***Caution:*** *We should never remove the run-level file from the /etc/init.d directory. Because no scripts are run from the /etc/init.d directory automatically, it is okay to keep them there. Scripts in /etc/init.d are only accessed as links from the /etc/rcX.d directories. Keep scripts in the init.d directory so we can add them later by re-linking them to the appropriate run-level directory.*

To reorganize or remove run-level scripts from the GUI, use the Service Configuration window. Either click System Settings? Services or log in as root user and type the following from a Terminal window:

```
# serviceconf &
```

Figure below shows an example of the Service Configuration window.


Figure above: Reorganize, add, and remove run-level scripts from the Service Configuration window.

The Service Configuration window lets us reconfigure services for run levels 3, 4, and 5. The run levels that we are currently running and currently editing are displayed near the top of the

screen. Services that are available for the run level appear in the middle of the frame, with check marks next to the ones configured to start at that level. Here is what we can do from this window:

- **Add**: Click the box next to each service we want to start automatically at that run level so that a check mark appears in the box.
- **Remove**: Click the run-level script that we want to remove for a particular run level to remove the check mark.
- **Save:** Click File, then Save Changes on the window to save any changes we have made to the run-level scripts.
- **Refresh**: Click View, then Refresh Service List to refresh the list of services.
- **Start, Stop, or Restart**: Click a service on the list. Select Actions, then either Start, Stop, or Restart Service. The selected service immediately starts, stops, or restarts.

Some administrators prefer text-based commands for managing run-level scripts and for managing other system services that start automatically. The chkconfig command can be used to list whether services that run-level scripts start, as well as services the xinetd daemon starts, are on or off. To see a list of all system services, with indications that they are on or off, type the following:

```
# chkconfig --list | less
```

We can then page through the list to see those services. If we want to view the status of an individual service, we can add the service at the end of the list option. For example, to see whether the nfs service starts in each run level, type the following:

```
# chkconfig --list nfs
nfs      0:off   1:off   2:off   3:on   4:on   5:on   6:off
```

This example shows that the nfs service is set to be on for run levels 3, 4, and 5, but that it is set to off for run levels 0, 1, 2, and 6.

Another tool that can be run from the shell to change which services start and do not start at various levels is the ntsysvcommand. Type the following as root user from the shell:

```
# ntsysv
```

A screen appears with a list of available services. Use the up and down arrow keys to locate the service we want. With the cursor on a service, press the Spacebar to toggle the service on or off. Press the Tab key to highlight the OK button, and press the Spacebar to save the change and exit. The ntsysv tool is somewhat clever; it checks our default run level (the initdefault set in the /etc/inittab file) and turns on the service so it starts at that run level.

### ☑ Adding Run-Level Scripts:

Suppose we want to create and configure our own run-level script. For example, after installing the binaries for the fictitious my_daemon program, it needs to be configured to start up in run

levels 3, 4, and 5, and terminated in any other run level. We can add the script to the /etc/init.d directory, then use the chkconfig command to configure it.

To use chkconfig, ensure that the following lines are included in the /etc/init.d/my_daemon script:

```
# chkconfig: 345 82 28
# description: Does something pretty cool - you really \
#    have to see it to believe it!
# processname: my_daemon
```

With those lines in place, simply run the following command:

```
# chkconfig --add my_daemon
```

Appropriate links are created automatically. This can be verified with the following command:

```
# chkconfig --list my_daemon
```

The resulting output should look like:

```
my_daemon 0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

### ☑ Managing Xinetd Services:

There are a bunch of services, particularly Internet services, that are not handled by separate run-level scripts. Instead, a single run-level script called xinetd (formerly inetd) is run to handle incoming requests for these services. For that reason, xinetd is sometimes referred to as the *super-server*. The xinetd run-level script (along with the xinetd daemon that it runs) offers the following advantages:

- **Fewer daemon processes.???** Instead of one (or more) daemon processes running on our computer to monitor incoming requests for each service, the xinetd daemon can listen for requests for many different services. As a result, when we type ps -ax to see what processes are running, dozens of fewer daemon processes will be running than there would be if each service had its own daemon.
- **Access control and logging.???** By using xinetd to oversee the management of services, consistent methods of access control (such as PAM) and consistent logging methods (such as the /var/log/messages file) can be used across all of the services.

When a request comes into our computer for a service that xinetd is monitoring, xinetd uses the /etc/xinetd.conf file to read configuration files contained in the /etc/xinetd.d directory. Then, based on the contents of the xinetd.d file for the requested service, a server program is launched to handle the service request (provided that the service is not disabled).

Each server process is one of two types: single-thread or multithread. A single-thread server will handle only the current request, whereas a multithread server will handle all incoming requests

for the service as long as there is still a client holding the process open. Then the multithread server will close and xinetd will begin monitoring that service again.

The following are a few examples of services that are monitored by xinetd. The daemon process that is started up to handle each service is also listed.

- **comsat** (/usr/sbin/in.comsat): Alerts the biff client that new mail has arrived.
- **eklogin** (/usr/kerberos/sbin/klogind): Kerberos-related login daemon.
- **finger** (/usr/sbin/in.fingerd): Handles incoming finger requests for information from remote users about local users.
- **gssftp** (/usr/kerberos/sbin/ftpd): Kerberos-related daemon for handling file transfer requests (FTP).
- **imap** (/usr/sbin/imapd): Daemon for handling requests from remote mail clients to get their mail from this IMAP server.
- **ntalk** (/usr/sbin/in.ntalkd): Daemon for handling requests to set up chats between a remote user and a local one (using the talk command).
- **rlogin** (/usr/sbin/in.rlogind): Daemon for responding to remote login requests (from a remote rlogin command).
- **rsh** (/usr/sbin/in.rshd): Handles requests from a remote client to run a command on the local computer.

Other services that can be launched by requests that come to xinetd include services for POP mail, remote telnet requests, Samba configuration requests (swat), and amanda network backups. A short description of each service is included in its /etc/xinetd.d file.

### ☑ Manipulating Run Levels:

Aside from the run level chosen at boot time (usually 3 or 5) and the shutdown or reboot levels (0 and 6, respectively), you can change the run level at any time while you're logged in (as root user). The telinit command (really just a symbolic link to init) enables you to specify a desired run level, causing the termination of all system processes that shouldn't exist in that run level, and starting all processes that should be running.

***Note:*** *The telinit command is also used to instruct init to reload its configuration file, /etc/inittab. This is accomplished with either the telinit q or the telinit Q commands.*

For example, if we encountered a problem with our hard disk on startup, we may be placed in single-user mode (run level 1) to perform system maintenance. After the machine is stable, we can just execute the command as follows:

```
# telinit 5
```

The init command handles terminating and starting all processes necessary to present us with a graphical login window.

- ***Determining The Current Run Level:***

We can determine the machine's current run level with the aptly named runlevel command. Using the previous example of booting into single-user mode and then manually changing the run level, the output of the runlevel command would be:

```
# runlevel
S 5
```

This means that the previous run level was S (for single-user mode) and the current run level is 5. If the machine had booted properly, the previous run level would be listed as N to indicate that there really wasn't a previous run level.

- ***Changing To A Shutdown Run Level:***

Shutting down the machine is simply a change in run level. With that in mind, other ways to change the run level include the reboot, halt, poweroff, and shutdown commands. The reboot command, which is a symbolic link to the halt command, executes a shutdown -r now, terminating all processes and rebooting the machine. The halt command executes shutdown -hnow, terminating all processes and leaving the machine in an idle state (but still powered on). Similarly, the poweroffcommand, which is also a link to the halt command, executes a change to run level 0, but if the machine's BIOS supports Advanced Power Management (APM), it will switch off the power to the machine.

***Note:*** *A time must be given to the shutdown command, either specified as +m (representing the number of minutes to delay before beginning shutdown) or as hh:mm (an absolute time value, where hh is the hour and mm is the minute that you would like the shutdown to begin). Alternatively, now is commonly used to initiate the shutdown immediately.*

## Scheduling System Tasks:

Frequently, we need to run a process unattended or at off-hours. The at facility is designed to run such jobs at specific times. Jobs we submit are spooled in the directory /var/spool/at, awaiting execution by the at daemon atd. The jobs are executed using the current directory and environment that was active when the job was submitted. Any output or error messages that haven't been redirected elsewhere are e-mailed to the user that submitted the job.

The following sections describe how to use the at, batch, and cron facilities to schedule tasks to run at specific times. These descriptions also include ways of viewing which tasks are scheduled and deleting scheduled tasks that we don't want to run anymore.

- ☑ **Using at.allow and at.deny:**

There are two access control files designed to limit which users can use the at facility. The file /etc/at.allow contains a list of users that are granted access, and the file /etc/at.deny contains a similar list of those who may not submit at jobs. If neither file exists, only the superuser is granted access to at. If a blank /etc/at.deny file exists (as in the default configuration), all users are allowed to utilize the at facility to run their own at jobs.

### ☑ Specifying When Jobs Are Run:

There are many different ways to specify the time at which an at job should run (most of which look like spoken commands). Table below has a few examples. These are not complete commands, they only provide an example of how to specify the time that a job should run.

| Command Line | When the Command Is Run |
|---|---|
| **at now** | The job is run immediately. |
| **at now + 2 minutes** | The job will start 2 minutes from the current time. |
| **at now + 1 hour** | The job will start one hour from the current time. |
| **at now + 5 days** | The job will start five days from the current time. |
| **at now + 4 weeks** | The job will start four weeks from the current time. |
| **at now next minute** | The job will start in exactly 60 seconds. |
| **at now next hour** | The job will start in exactly 60 minutes. |
| **at now next day** | The job will start at the same time tomorrow. |
| **at now next month** | The job will start on the same day and at the same time next month. |
| **at now next year** | The job will start on the same date and at the same time next year. |
| **at now next fri** | The job will start at the same time next Friday. |
| **at teatime** | The job will run at 4 p.m. They keywords noon and midnight can also be used. |
| **at 16:00 today** | The job will run at 4 p.m. today. |
| **at 16:00 tomorrow** | The job will run at 4 p.m. tomorrow. |
| **at 2:45pm** | The job will run at 2:45 p.m. on the current day. |
| **at 14:45** | The job will run at 2:45 p.m. on the current day. |
| **at 5:00 Sep 14 2003** | The job will begin at 5 a.m. on September 14, 2003. |
| **at 5:00 9/14/03** | The job will begin at 5 a.m. on September 14, 2003. |

### ☑ Submitting Scheduled Jobs:

The at facility offers a lot of flexibility in how we can submit scheduled jobs. There are three ways to submit a job to the at facility:

- ☑ **Piped in from standard input**.??? For example, the following command will attempt to build the Perl distribution from source in the early morning hours while the machine is likely to be less busy:

```
echo "cd /tmp/perl; make ; ls -al" | at 2am tomorrow
```

An ancillary benefit to this procedure is that a full log of the compilation process will be e-mailed to the user that submitted the job.

- ☑ **Read as standard input.???** If no command is specified, at will prompt us to enter commands at the special at>prompt, as shown in the following example. We must indicate the end of the commands by pressing Ctrl+D, which signals an End of Transmission (<EOT>) to at.

```
$ at 23:40
```

```
at> cd /tmp/perl
at> make
at> ls -al
at> <Ctrl-d>
```

☑ **Read from a file.???** When the -f command-line option is followed by a valid filename, the contents of that file are used as the commands to be executed, as in the following example:

```
$ at -f /root/bin/runme now + 5 hours
```

This runs the commands stored in /root/bin/runme in five hours. The file can either be a simple list of commands or a shell script to be run in its own subshell (that is, the file begins with #!/bin/bash or the name of another shell).

### ☑ Viewing Scheduled Jobs:

We can use the atq command (effectively the same as at -l) to view a list of our pending jobs in the at queue, showing each job's sequence number, the date and time the job is scheduled to run, and the queue in which the job is being run.

The two most common queue names are "a," which represents the at queue, and "b," which represents the batch queue. All other letters (upper- and lowercase) can be used to specify queues with lower priority levels. If the atq command lists a queue name as =, it indicates that the job is currently running. Here is an example of output from the atq command:

```
# atq
2    2003-09-02 00:51 a
3    2003-09-02 00:52 a
4    2003-09-05 23:52 a
```

Here you can see that there are three at jobs pending (job numbers 2, 3, and 4, all indicated as "a"). After the job number, the output shows the date and hour each job is scheduled to run.

### ☑ Deleting Scheduled Jobs:

If we decide that we'd like to cancel a particular job, we can use the atrm command (equivalent to at -d) with the job number (or more than one) as reported by the atq command. For example, using the following output from atq:

```
# atq
18      2003-09-01 03:00 a
19      2003-09-29 05:27 a
20      2003-09-30 05:27 a
21      2003-09-14 00:01 a
```

```
22      2003-09-01 03:00 a
```

We can remove the jobs scheduled to run at 5:27 a.m. on September 29 and September 30 from the queue with the command atrm 19 20.

### ☑ Using The Batch Command:

If system resources are at a premium on our machine, or if the job we submit can run at a priority lower than normal, the batch command (equivalent to at -q b) may be useful. It is controlled by the same atd daemon, and it allows job submissions in the same format as at submissions (although the time specification is optional).

However, to prevent our job from usurping already scarce processing time, the job will run only if the system load average is below a particular value. The default value is 0.8, but specifying a command-line option to atd can modify this. This was used as an example in the earlier section describing start-up and shutdown. Here is an example of the batch command:

```
$ batch
at> du -h /home > /tmp/duhome
at> <Ctrl-d>
```

In this example, after the batch command, the at facility is invoked to enable, to enter the command(s) that run. Typing the du -h /home > /tmp/duhome command line has the disk usages for everything in the /home directory structure output to the /tmp/duhome file. On the next line, pressing Ctrl+D ends the batch job. As soon as the load average is low enough, the command is run. (Run the top command to view the current load average.)

### ☑ Using the cron facility:

Another way to run commands unattended is via the cron facility. Part of the vixie-cron RPM package, cron addresses the need to run commands periodically or routinely (at least, more often than we'd care to manually enter them) and allows lots of flexibility in automating the execution of the command. As with the at facility, any output or error messages that haven't been redirected elsewhere are e-mailed to the user that submitted the job.

Also like the at facility, cron includes two access control files designed to limit which users can use it. The file /etc/cron.allowcontains a list of users that are granted access, and the file /etc/cron.deny contains a similar list of those who may not submit cron jobs. If neither file exists, all users are granted access to cron.

There are four places where a job can be submitted for execution by the cron daemon crond:

- **The /var/spool/cron/*username* file.???** This method, where each individual user (indicated by *username*) controls his or her own separate file, is the method used on UNIX System V systems.
- **The /etc/crontab file.???** This is referred to as the *system crontab file*, and was the original crontab file from BSD UNIX and its derivatives. Only root has permission to modify this file.

- **The /etc/cron.d directory.???** Files placed in this directory have the same format as the /etc/crontab file. Only root is permitted to create or modify files in this directory.
- **The /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly directories.???** Each file in these directories is a shell script that runs at the times specified in the /etc/crontab file (by default at one minute after the hour, at 4:02 a.m. every day, Sunday at 4:22 a.m., and 4:42 a.m. on the first day of the month, respectively). Only root is allowed to create or modify files in these directories.

The standard format of an entry in the /var/spool/cron/*username* file consists of five fields specifying when the command should run: minute, hour, day of the month, month, and day of the week. The sixth field is the actual command to be run.

The files in the /etc/cron.d directory and the /etc/crontab file use the same first five fields to determine when the command should run. However, the sixth field represents the name of the user submitting the job (because it cannot be inferred by the name of the file as in a /var/spool/cron/*username* directory), and the seventh field is the command to be run. Table 12-7 lists the valid values for each field common to both types of files.

| Field Number | Field | Acceptable Values |
|---|---|---|
| **1** | minute | Any integer between 0 and 59 |
| **2** | hour | Any integer between 0 and 23 |
| **3** | day of the month | Any integer between 0 and 31 |
| **4** | month | Any integer between 0 and 12, or an abbreviation for the name of the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec) |
| **5** | day of the week | Any integer between 0 and 7 (where both 0 and 7 can represent Sunday), or abbreviation for the day (Sun, Mon, Tue, Wed, Thu, Fri, Sat) |

An asterisk (*) in any field indicates all possible values for that field. For example, an asterisk in the second column is equivalent to 0,1,2 . . . 22,23, and an asterisk in the fourth column means Jan, Feb, Mar . . . Nov, Dec. In addition, lists of values, ranges of values, and increments can be used. For example, to specify the days Monday, Wednesday, and Friday, the fifth field could be represented as the list Mon, Wed, Fri. To represent the normal working hours in a day, the range 9–5 could be specified in the second field. Another option is to use an increment, as in specifying 0–31/3 in the third field to represent every third day of the month, or */5 in the first field to denote every five minutes.

Lines beginning with a # character in any of the crontab-format files are comments, which can be very helpful in explaining what task each command is designed to perform. It is also possible to specify environment variables (in Bourne shell syntax, e.g., NAME="value") within the crontab file. Any variable can be specified to fine-tune the environment in which the job will run, but one that may be particularly useful is MAILTO. The following line will send the results of the cron job to a user other than the one that submitted the job:

```
MAILTO=otheruser
```

If the following line appears in a crontab file, all output and error messages that haven't already been redirected will be discarded:

```
MAILTO=
```

### ☑ Modifying scheduled tasks with crontab:

The files in /var/spool/cron should not be edited directly. They should only be accessed via the crontab command. To list the current contents of your own personal crontab file, type the following command:

```
$ crontab -l
```

All crontab entries can be removed with the following command:

```
$ crontab -r
```

Even if our personal crontab file doesn't exist, we can use the following command to begin editing it:

```
$ crontab -e
```

The file automatically opens in the text editor that is defined in our EDITOR or VISUAL environment variables, with vi as the default. When we're done, simply exit the editor. Provided there were no syntax errors, our crontab file will be installed. For example, if our user name is jsmith, we have just created the file /var/spool/cron/jsmith. If we add a line (with a descriptive comment, of course) to remove any old core files from our source code directories, that file may look similar to this:

```
# Find and remove core files from /home/jsmith/src
5 1 * * Sun,Wed find /home/jsmith/src -name core -exec rm {} \; > /dev/null 2>&1
```

The root user can access any user's individual crontab file by using the -u *username* option to the crontab command.

### ☑ Understanding cron files:

There are separate cron directories set up to contain cron jobs that run hourly, daily, weekly, and monthly. These cron jobs are all set up to run from the /etc/crontab file. The default /etc/crontab file looks like this:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/


# run-parts
```

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

The first four lines initialize the run-time environment for all subsequent jobs (the subshell in which jobs will run, the executable program search path, the recipient of output and error messages, and that user's home directory). The next five lines execute (as the user root) the run-parts program that controls programs that we may wish to run periodically.

run-parts is a shell script that takes a directory as a command-line argument. It then sequentially runs every program within that directory (shell scripts are most common, but binary executable and links are also evaluated). The default configuration executes programs in /etc/cron.hourly at one minute after every hour of every day; /etc/cron.daily at 4:02 a.m. every day; /etc/cron.weekly at 4:22 a.m. on Sundays; and /etc/cron.monthly at 4:42 a.m. on the first day of each month.

Here are examples of files that are installed in cron directories for different software packages:

- **/etc/cron.daily/logrotate.cron**: Automates rotating, compressing, and manipulating system logfiles.
- **/etc/cron.daily/makewhatis.cron**: Updates the whatis database (contains descriptions of man pages), which is used by the man -k, apropos, and whatis commands to find man pages related to a particular word.
- **/etc/cron.daily/slocate.cron**: Updates the /var/lib/slocate/slocate.db database (using the updatedb command), which contains a searchable list of files on the machine.
- **/etc/cron.daily/tmpwatch**: Removes files from /tmp, /var/tmp, and /var/catman that haven't been accessed in ten days.
- **/etc/cron.hourly/diskcheck**: Checks available disk space on your hard drive each hour. Responds to low disk space based on settings in /etc/diskcheck.conf.

The makewhatis.cron script installed in /etc/cron.weekly is similar to the one in /etc/cron.daily, but it completely rebuilds the whatis database, rather than just updating the existing database.

Finally, in the /etc/cron.d directory are files that have the same format as /etc/crontab files.

## Backing Up and Restoring:

If we've ever suffered a hard drive crash, we know just how aggravating it can be. Irreplaceable data can be lost. Countless hours may be spent reinstalling our operating system and applications. It is not a fun experience. It need happen only once for us to learn the importance of making regular backups of our critical data.

Today, larger and faster backup media can simplify the process of backing up our data. Red Hat Linux supports many different types of media, such as writable CD-ROM, DVD, and magnetic tape

for creating backups. Using tools such as cron, we can configure backups to run unattended at scheduled times.

### ☑ Selecting a Backup Strategy:

There are several approaches to backing up our data. We need to ask ourself a few questions to decide which approach is best for us. Some things that we should consider are:

- In the event of a crash, how much downtime can I tolerate?
- Will I need to recover older versions of my files or is the most recent revision sufficient?
- Do I need to back up files for just one computer or for many computers on a network?

Our answers to these questions will help us decide how often to do full backups and how often to do incremental backups. If the data is particularly critical, we may even decide that we need to have our data duplicated constantly, using a technique called *disk mirroring*. The following sections describe different backup methods.

#### 1. Full Backup:

A full backup is one that stores every file on a particular disk or partition. If that disk should ever crash, we can rebuild our system by restoring the entire backup to a new disk. Whatever backup strategy we decide on, some sort of full backup should be part of it. We may perform full backups every night or perhaps only once every week; it depends on how often we add or modify files on our system, as well as the capacity of our backup equipment.

#### 2. Incremental Backup:

An incremental backup is one that contains only those files that have been added or modified since the last time a more complete backup was made. We may choose to do incremental backups to conserve our backup media. Incremental backups also take less time to complete. This can be important when systems are in high use during the work week and running a full backup would degrade system performance. Full backups can be reserved for the weekend when the system is not in use.

#### 3. Disk Mirroring:

Full and incremental backups can take time to restore, and sometimes we just can't afford that downtime. By duplicating our operating system and data on an additional hard drive, we can greatly increase the speed with which we can recover from a server crash.

With disk mirroring, it is usually common for the system to continuously update the duplicate drive with the most current information. In fact, with a type of mirroring called RAID 1, the duplicate drive is written to at the same time as the original, and if the main drive fails, the duplicate can immediately take over. This is called *fault-tolerant* behavior, which is a must if we are running a mission-critical server of some kind.

#### 4. Network Backup:

All of the preceding backup strategies can be performed over a network. This is good because we can share a single backup device with many computers on a network. This is much cheaper and more convenient than installing a tape drive or other backup device in every system on our

network. If we have many computers, however, our backup device will require a lot of capacity. In such a case, we may consider a mechanical tape loader, writable DVD drive or CD jukebox.

It is even possible to do a form of disk mirroring over the network. For example, a Web server may store a duplicate copy of its data on another server. If the first server crashes, a simple TCP/IP host name change can redirect the Web traffic to the second server. When the original server is rebuilt, it can recover all of its data from the backup server and be back in business.

### ☑ Selecting a Backup Medium:

Armed with a backup strategy in mind, it is time to select a backup medium. Several types of backup hardware and media are available for use with Red Hat Linux. Each type has its advantages and disadvantages.

The type of medium to choose depends largely on the amount of data we need to archive, how long we will store backups, how often we expect to recover data from our backups, and how much we can afford to spend. Table below compares the most common backup media.

| Backup Medium | Advantage | Disadvantage |
|---|---|---|
| **Magnetic tape** | High capacity, low cost for archiving massive amounts of data. | Sequential access medium, so recovery of individual files can be slow. |
| **Writable CDs** | Random access medium, so recovery of individual files is easier. Backups can be restored from any CD-ROM. | Limited storage space (approximately 650MB per CD). |
| **Writable DVD** | Random access medium (like CDs). Large capacity (4.7GB, although the actual capacity we can achieve might be less). | DVD-RW drives and DVD-R disks are relatively expensive (though coming down in price). Slower and less common than CD-ROM drives. |
| **Additional hard drive** | Allows faster and more frequent backups. Fast recovery from crashes. No media to load. Data can be located and recovered more quickly. We can configure the second disk to be a virtual clone of the first disk, so that we can boot off of the second disk if the first disk crashes. | Data cannot be stored offsite, thus there is risk of data loss if the entire server is destroyed. This method is not well suited to keep historical archives of the many revisions of our files. The hard drive will eventually fill up. |

The following sections describe how to use magnetic tape, writable DVDs, and writable CDs as backup media. How to use additional hard drives as backup media is described later in this chapter.

#### 1. Magnetic Tape:

Magnetic tape is the most common medium used for backing up large amounts of computer data. Tapes provide a low-cost, convenient way to archive our files. Today's high-capacity tape drives

can back up many gigabytes of data on an amazingly small tape, allowing vast amounts of information to be safely stored.

The primary disadvantage of magnetic tape is that it is a sequential access medium. This means that tapes are read or written from beginning to end, and searching for a particular file can be time-consuming. For this reason, tape is a good choice for backing up and restoring entire file systems, but not the ideal choice to recover individual files on a regular basis.

Red Hat Linux can use a wide variety of tape drives. Most SCSI tape drives will work with the generic Linux kernel. Even many IDE tape drives are supported via a "SCSI emulation" mode. Some drives, however, require installation of additional software.

- ***Using ftape Tools For Magnetic Tape:***

If our tape drive is attached to an IDE floppy controller cable, we will need to use the ftape driver to access it. Fortunately, version 3.04d of the ftape loadable module is bundled with the Linux 2.4 kernel. When our Linux system boots, it should auto-detect the tape drive and load the ftape driver. To verify that our system loaded the tape driver, type the following command shortly after we boot our computer:

```
dmesg | grep ftape
```

This searches the most recent kernel messages for lines containing the word ftape. If the ftape module was loaded, we should see something like this:

```
ftape v3.04d 25/11/97
[000] ftape-init.c (ftape_init) - installing QIC-117 floppy tape
hardware drive... .
[001] ftape-init.c (ftape_init) - ftape_init @ 0xd08b0060.
[002] ftape-buffer.c (add_one_buffer) - buffer nr #1 @ c1503914, dma
area @ c02c0000.
[003] ftape-buffer.c (add_one_buffer) - buffer nr #2 @ c1503c44, dma
area @ c0298000.
[004] ftape-buffer.c (add_one_buffer) - buffer nr #3 @ c50abaac, dma
area @ c0328000.
[005] ftape-calibr.c (time_inb) - inb() duration: 1109 nsec.
[006] ftape-calibr.c (ftape_calibrate) - TC for `ftape_udelay()' = 310
nsec (at 20479 counts).
[007] ftape-calibr.c (ftape_calibrate) - TC for `fdc_wait()' = 2208 nsec
(at 2559 counts).
```

If the module was not loaded, then we should check if our kernel is compiled with support for the ftape module and our particular tape drive. Refer to the section on reconfiguring the kernel located in Appendix A. When running the make config or make xconfig command, choose Character Devices? ftape, the floppy tape device driver. We can then verify that the ftape module

is enabled and that our specific hardware is supported. Continue with compiling and installing the kernel in the normal way.

In most cases, an ftape device can be accessed just like a SCSI device. The primary difference is that an ftape device file contains the letters qft (for QIK Floppy Tape) where a SCSI tape contains st. For example, the device file for the first SCSI tape on our system will probably be /dev/st0; the device file for the first floppy tape will likely be /dev/qft0.

All of the standard tape and archiving related programs should work fine with both types of hardware. Nevertheless, there are a few extra programs that you may find useful when working with a floppy tape drive. These programs can be found in the ftape-tools package located at ftp://metalab.unc.edu/pub/Linux/kernel/tapes/. Download the file named ftape-tools-1.09.tar.gz. If a version higher than 1.09 is available, download that instead. Extract the ftape package using the tar command:

```
$ tar -xzvf ftape-tools-1.09.tar.gz
```

This extracts the package into an ftape-tools-1.09 directory. Use the cd command to go to that directory and to run the ./configure script to prepare the package's makefiles. Next, compile the package by typing the make command:

```
$ ./configure
$ make
```

Finally, assume root privilege using the su command and type make install to install the ftape-tools programs and online man pages to the appropriate directories:

```
# make install
```

- ***Testing the Magnetic Tape Drive:***

We should now be ready to test our tape drive. Insert a blank tape into the tape drive and type the following command:

```
$ mt -f /dev/qft0 rewind
```

We should hear the tape spin as the system rewinds the tape. This will be a very short process if the tape is already rewound. The mt command provided with ftape-tools is used to scan, rewind, and eject magnetic tapes in a floppy controller tape drive. It is very similar in operation to the st command, which is used to perform the same functions on SCSI tapes.

- ***Formatting Magnetic Tapes:***

The ftape-tools package also includes a tool for formatting tapes as well. Most tapes now come preformatted. In the event that we have an older floppy controller tape drive that uses unformatted tapes, use the ftformat command to format them:

```
$ /usr/local/bin/ftformat -f /dev/qft0
```

Usually, the -f parameter with the device name is the only parameter that we need to supply. Nevertheless, we are encouraged to read the online man page for ftformat to learn more about its options and capabilities.

### 2. Writable CD-ROM drives:

Another backup medium that is gaining popularity is the writable CD-ROM drive. Writable CD-ROM drives have several advantages over tape, the primary one being that CDs are a random access medium. This means that the CD drive can quickly locate a particular file on the CD without sequentially scanning through the entire disc. This is useful when we need to keep a revision history of frequently changing data files (such as source code for a software project or drafts of legal documents).

Another advantage is the extremely long life span of CDs. If we wish to archive our backups for a very long time, a writable CD drive is a good choice. If our backups are intended for short-term storage, we should probably consider a rewritable CD drive. A rewritable CD (unlike plain writable CDs) can be reformatted and used to store new backups.

The biggest drawback is that a CD can store at most about 650MB of data. In contrast, DVDs can store 4.7GB of data and many tape drives can store multiple gigabytes of data. For example, DAT DDS-3 tapes can hold up to 24GB of compressed data, while 8mm AIT-2 tapes can hold up to 100GB of compressed data.

- ***Getting cdrecord for writable CDs:***

To write CDs with Red Hat Linux we need to install the cdrecord package. This package contains components such as the cdrecord, devdump, isodump, isoinfo, isovfy, and readcd commands. The cdrecord package is included with the Red Hat Linux distribution.

***Note:*** *The cdrecord package requires that you use a SCSI CD drive. If you have an IDE/ATAPI CD drive, which is a very popular device, you may need to configure that drive to do SCSI emulation. For the latest versions of Red Hat Linux, however, SCSI emulation is usually taken care of automatically.*

- ***Writing to CDs:***

Because the data written to a CD becomes permanent once it is written, we need to format the CD and copy files to it all in one step. If we formatted it first, we would end up with an empty file system on a CD that can no longer be written to.

The first step is to create an image of the CD file system as a file on our computer. We do this with the mkisofs command. As an example, imagine that we want to back up the home directory for user mary. We would invoke the mkisofs command and pass it the name of the file system image file to create, followed by the directory to base it on:

```
$ mkisofs -R -o /var/tmp/mary.cd /home/mary
```

This creates an ISO9660 file system image in a file named mary.cd located in the /var/tmp directory. The –R option causes Linux-specific file ownership and long file names to be used. If our /var partition does not have enough room for the image, choose a different location.

***Note:*** *By default, mkisofs preserves the ownership and access rights of files and directories when it creates the file system image. This is appropriate when we are making a backup, but not when we are creating a software distribution CD. In such a case, add the -r option instead of -R as the first parameter to mkisofs. It will then store all files as publicly readable and, where appropriate, executable.*

Before we can write this image file to a CD, we must first discover the SCSI bus number, device ID number, and Logical Unit Number (LUN) of the CD drive. Unless we have an actual SCSI bus in our computer, the emulated SCSI bus is probably numbered zero. We can find out which SCSI device ID the CD drive is using. Invoke the cdrecord command with the single parameter scanbus:

```
# cdrecord -scanbus
```

We should see a response similar to the following:

```
Cdrecord 2.0 (i686-pc-linux-gnu) Copyright (C) 1995-2002 J?rg Schilling
Linux sg driver version: 3.1.25
Using libscg version 'schily-0.7'
scsibus0:
        0,0,0     0) 'IDE-CD ' 'R/RW 4x4x24  ' '1.04' Removable CD-ROM
        0,0,1     1) *
        0,0,2     2) *
        0,0,3     3) *
        0,0,4     4) *
        0,0,5     5) *
        0,0,6     6) *
        0,0,7     7) *
```

This tells us that the CD drive is using SCSI ID zero. The Logical Unit Number in this case should always be zero, so we now have all three numbers. We supply them to cdrecord as part of the dev parameter. The SCSI bus number is listed first; it is followed by the ID number, and then by the LUN. The entire command should look similar to this:

```
# cdrecord -v speed=2 dev=0,0,0 -data /var/tmp/mary.cd
```

Several additional parameters are included in the command. The -v parameter tells cdrecord to supply verbose output to the screen. The speed parameter tells cdrecord what speed to record at (in this case X2). (We might choose to leave off speed=2 and let cdrecord auto-detect the record speed of our CD burner.) The -data parameter tells cdrecord that the next parameter is the name of the file system image to write to the CD. (We can add the -eject to eject the CD when it is done.) As it works, cdrecord should display status messages that look similar to the following:

```
Cdrecord 2.0 (i686-pc-linux-gnu) Copyright (C) 1995-2002 J?rg Schilling
TOC Type: 1 = CD-ROM
scsidev: '0,0,0'
scsibus: 0 target: 0 lun: 0
Linux sg driver version: 3.1.24
Using libscg version 'schily-0.7'
atapi: 1
Device type    : Removable CD-ROM
Version        : 0
Response Format: 1
Vendor_info    : 'IDE-CD'
Identifikation : 'R/RW 4x4x24        '
Revision       : '1.04'
Device seems to be: Generic mmc CD-RW.
Using generic SCSI-3/mmc CD-R driver (mmc_cdr).
Driver flags   : MMC SWABAUDIO
Supported modes: TAO PACKET RAW/R16
Drive buf size : 1572864 = 1536 KB
FIFO size      : 4194304 = 4096 KB
Track 01: data   322 MB
Total size:     370 MB (36:43.12) = 165234 sectors
Lout start:     370 MB (36:45/09) = 165234 sectors
Current Secsize: 2048
ATIP info from disk:
  Indicated writing power: 4
  Is not unrestricted
  Is not erasable
  Disk sub type: Medium Type A, high Beta category (A+) (3)
  ATIP start of lead in:  -11849 (97:24/01)
  ATIP start of lead out: 359848 (79:59/73)
Disk type:    Long strategy type (Cyanine, AZ0 or similar)
Manuf. index: 25
Manufacturer: Taiyo Yuden Company Limited
Blocks total: 359848 Blocks current: 359848 Blocks remaining: 194614
Starting to write CD/DVD at speed 2 in real TAO mode for single session.
Last chance to quit, starting real write in 0 seconds. Operation starts.
```

```
Waiting for reader process to fill input buffer ... input buffer ready.
Performing OPC...
Starting new track at sector: 0
Track 01:  322 of  322 MB written (fifo 100%)  [buf  99%]   2.0x.
Track 01: Total bytes read/written: 338395136/338395136 (165232 sectors).
Writing  time:  1110.710s
Average write speed   2.0x.
Fixating...
Fixating time:  126.108s
cdrecord: fifo had 5331 puts and 5331 gets.
cdrecord: fifo was 0 times empty and 5262 times full, min fill was 96%.
```

After cdrecord finishes writing the CD and our shell prompt returns, delete the file system image file /var/tmp/mary.cd. Label the CD appropriately and store it in a safe place.

If we need any files that were copied to the CD, just return the CD to the CD drive. If it doesn't automatically open a window displaying the contents of the CD, type: mount /mnt/cdrom as root user. Open /mnt/cdrom in a folder window and copy the files we want.

### 3. Writable DVD drives:

Using a writable DVD drive and the dvdrecord command, we can back up our data to DVD-R disks. The procedure is almost identical to back up data onto a CD-ROM, with the following exceptions:

- We use the dvdrecord command instead of cdrecord (although they both have nearly identical interfaces).
- Each backup disk can hold a lot more data (4.7GB compared to 700MB).
- Both the DVD writer and medium are more expensive than the CD counterparts.

***Note:*** *When manufacturers say 4.7GB, they are talking about 1000MB per GB, not 1024MB. Therefore, we can really only store up to about 4.4GB of data on a DVD.*

The dvdrecord command should allow us to burn DVDs on any DVD burner that is compliant with the Multimedia Command (MMC) standard. Development of the dvdrecord command was done on a Pioneer DVR-A03 DVD writer.

Follow the procedure in the "Writable CD-ROM drives" section to create a file system image file (using mkisofs) and determine the location of our DVD-R driver. Then use the dvdrecord command to actually burn the DVD. Here is an example of a dvdrecord command line that burns a file system image called bigimage.cd:

```
# dvdrecord -v speed=2 dev=0,0,0 -data bigimage.cd
```

### 4. Backing Up to a Hard Drive:

Removable media such as tapes, DVDs, and CDs are not the only choice for backing up our data. We may find it useful to install a second hard drive in our system and use that drive for backups. This has several advantages over other backup media:

- Data can be backed up quickly and throughout the day; thus, backed-up data will be more current in the event of a crash.
- There's no medium to load. Data can be located and recovered more quickly.
- We can configure the second disk to be a virtual clone of the first one. If the first disk crashes, we can simply boot off of the second disk rather than installing new hardware. With disk mirroring software, this process can even be automated.

There are, however, a few disadvantages to backing up to a hard drive:

- This method is not well suited to keeping historical archives of the many revisions of our files because the hard drive will eventually fill up.
- The backups cannot be stored offsite, so if the server is destroyed, our data is lost.

Because of these disadvantages, we should probably not use a hard drive as our only backup medium. Instead, combine it with some sort of removable media backup such as a tape drive. Hard-drive mirroring can provide a *snapshot* backup that will get us up and running again quickly, while removable backups can provide longer-term storage of data and protection from more catastrophic failures. Hard-drive mirroring is discussed in the next section.

The simplest form of second-hard-drive backup is to simply copy important files to the other drive using the cp or tarcommand. The most sophisticated method is to provide fault-tolerant disk mirroring using RAID software. If we don't need the high level of protection from data loss that RAID disk mirroring provides, but we still like the idea of having a complete duplicate of our data on-hand and ready for use, there is an alternative.

- ***Getting and installing mirrordir to clone directories***

The mirrordir package is a way of doing hard-drive mirroring. Mirrordir is a powerful tool that enables us to make and maintain an exact copy of a hierarchy of directories. We can find its official Web site at http://mirrordir.sourceforge.net. It can be downloaded by selecting the Download RPM button from that site.

After downloading the file, install it in the same way we install any rpm. For example, if we have downloaded the rpm file to /tmp, we can type:

```
# rpm -i /tmp/mirrordir*rpm
```

- ***Cloning a directory with mirrordir***

Now that we have mirrordir installed, we can use it to clone a directory. Suppose we have a second hard drive with a partition large enough to hold a copy of our /home partition. Our first step is to create a directory to mount the partition on, and then mount it. Log in as root and type the following:

```
# mkdir -p /mirror/home
```

```
# mount /dev/hdb5 /mirror/home
```

In this example, we are mounting the fifth partition (5) of the second hard drive (hdb), hence the device name /dev/hdb5. The b refers to the second disk, and the 5 refers to the partition number. We may use a different drive or partition. If so, adjust the device name accordingly. Assuming the mount command successfully mounted the drive, we are ready to copy our /homepartition. Enter the following command:

```
# mirrordir /home /mirror/home
```

This command causes the contents of the /home directory to be exactly duplicated in the /mirror/home directory.

***Caution:*** *It is very important that we get the order of these parameters correct. If we reverse them, the entire contents of our /home directory will be deleted! we will, in essence, be copying an empty directory back to our /homedirectory.*

We now have a backup of our entire /home partition. We can run mirrordir again in the future, and it will again make the /mirror/home directory an exact duplicate of /home. It will copy to /mirror/home only those files that have been added or modified since the previous run of mirrordir. Also, it will delete any files from /mirror/home that are no longer on /home. Thus, the mirror is kept current without copying the entire /home partition each time. If the disk with the /home partition should ever crash, we can replace the disk and then copy the file system back by issuing the mirrordir command with the parameters reversed:

```
# mirrordir /mirror/home /home
```

Furthermore, we can be up and running even faster by simply turning the mirrored partition into the actual home partition. Just edit the /etc/fstab file and change the device name for /home so that it matches the mirrored directory. Now unmount and remount the /home partition (or reboot the computer), and the mirrored directory will be mounted to /home. Our users will never be the wiser. We may even consider mirroring every partition on our main disk. Then, even a complete disk crash can be recovered from quickly. Simply turn the secondary disk into the primary disk, and we are up and running again.

- ***Automating mirroring:***

To automate the mirroring process, we are suggested to creat a small script that mounts the mirror partition, runs the mirrordircommand, and then unmounts the partition. Leaving the partition unmounted when not in use reduces the chance of data being accidentally deleted from it. Create a script named mirror.sh and place it in the /usr/local/sbin directory. Something similar to the following should do the trick:

```
#!/bin/sh
#
#  mirror.sh:  Mirror the /home partition to a second hard drive
#
```

```
/bin/mount /dev/hdb5 /mirror/home
mirrordir /home /mirror/home
/bin/umount /mirror/home
```

Now tell cron to periodically run this script. Invoke the cron command with the -e option. This brings up an editor containing root's cron jobs. Add a line similar to the following at the end of the list.

```
0 * * * * /usr/local/sbin/mirror.sh
```

Save and exit the editor. This cron entry causes the mirror.sh script to run once an hour. If we decide to mirror other partitions, we can do it quite easily by creating appropriate mount points in /mirror and then adding matching sections to the mirror.sh script.

### ☑ Backing Up Files with dump:

The dump command is the most commonly used tool for performing backups on UNIX systems. This command traces its history back to the early days of UNIX and thus is a standard part of nearly every version of UNIX. Likewise, the dump package is included in Red Hat Linux. If it was not installed by default when we first set up our Linux system, we can install it from the dump RPM file located on Red Hat Linux installation CD #1.

The dump package actually consists of several commands. We can read online man pages for more information about them, but Table below has a short description of the programs.

| Command | Description |
|---|---|
| **Dump** | Creates backup archives of whole disk partitions or selected directories. |
| **Restore** | Can be used to restore an entire archive or individual files from an archive to the hard drive. |
| **Rmt** | A program used by the dump and restore commands to copy files across the network. We should never need to use this command directly. |

#### ❖ Creating A Backup With Dump:

When making a file system backup using the dump command, we must supply parameters specifying the dump level, the backup media, and the file system to back up. We can also supply optional parameters to specify the size of the backup media, the method for requesting the next tape, and the recording of file system dump times and status.

The first parameter to dump is always a list of single-letter option codes. This is followed by a space-separated list of any arguments needed by those options. The arguments appear in the same order as the options that require them. The final parameter is always the file system or directory being backed up.

```
# dump options arguments filesystem
```

Table below lists the various one-letter option codes for the dump command.

| Dump Options | Description |
|---|---|
| **0-9** | The dump level. Selecting a dump level of 0 backs up all files (a full dump). A higher number backs up only those files modified since the last dump of an equal or lower number (in essence, an incremental dump). The default dump level is 9. |
| **-B records** | The number of dump records per volume. Basically, the amount of data we can fit on a tape. This option takes a numeric argument. |
| **-b kbperdump** | The number of kilobytes per dump record. Useful in combination with the -B option. This option takes a numeric argument. |
| **-h level** | Files can be marked with a nodump attribute. This option specifies the dump level at or above which the nodump attribute is honored. This option takes a numeric argument of 1-9. |
| **-f file** | The name of the file or device to write the dump to. This can even be a file or device on a remote machine. |
| **-d density** | Sets the tape density. The default is 1600 bits per inch. This option takes a numeric argument. |
| **-n** | When a dump needs attention (such as to change a tape), dump will send a message to all of the users in the operator group. This option takes no arguments. |
| **-s feet** | Specifies the length in feet of the dump tape. This calculation is dependent on tape density (option d) and the dump record (options B and b). This takes a numeric argument. |
| **-u** | Record this backup in the /etc/dumpdates file. It is a good idea to use this option, especially if we create incremental backups. |
| **-t date** | Specify a date and time on which to base incremental backups. Any files modified or added after that time will be backed up. This option causes dump to ignore the /etc/dumpdates file. It takes a single argument, a date in the format specified by the ctime man page. |
| **-W** | This option causes dump to list the file systems that need to be backed up. It does this by looking at the /etc/dumpdates file and the /etc/fstab file. |
| **-w** | This works like the W option but lists the individual files that should be backed up. |

Thus, a typical dump command may look similar to the following:

```
# dump 0uBf 500000 /dev/qft0 /dev/hda6
```

This command results in dump performing a level zero (full) backup of the /dev/hda6 file system, storing the backup on the tape drive /dev/qft0, and recording the results in /etc/dumpdates. The B option is used to increase the expected tape block count to 500000; otherwise, dump would prompt for a new tape far earlier than required. The dump command prints status messages to the screen, letting us know how far along the backup has progressed and estimating how much time it will take to complete. The output looks similar to this:

```
DUMP: Date of this level 0 dump: Sat Aug 23 23:33:37 2003
 DUMP: Dumping /dev/hda6 (/home) to /dev/qft0
```

```
DUMP: Exclude ext3 journal inode 8
DUMP: Label: /home
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 93303 tape blocks on 0.19 tape(s).
DUMP: Volume 1 started with block 1 at: Sat Aug 23 23:33:47
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: Closing /dev/qft0
DUMP: Volume 1 completed at: Sat Aug 23 23:35:35 2003
DUMP: Volume 1 94360 tape blocks (92.15MB)
DUMP: Volume 1 took 0:01:48
DUMP: Volume 1 transfer rate: 873 kB/s
DUMP: 94360 tape blocks (92.15MB) on 1 volume(s)
DUMP: finished in 108 seconds, throughput 873 kBytes/sec
DUMP: Date of this level 0 dump: Sat Aug 23  23:33:37 2003
DUMP: Date this dump completed: Sat Aug 23  23:35:35 2003
DUMP: Average transfer rate: 873 kB/s
DUMP: DUMP IS DONE
```

❖ **Understanding Dump Levels:**

The dump command can back up all files on a file system, or it can selectively back up only those files that have changed recently. The dump level parameter is used to specify this behavior. A dump level of 0 results in a full backup of all files on the file system. Specifying a higher number (1–9) backs up only those files that have been changed or added since the most recent dump of the same or lower dump level. It is recommend to use dump levels to implement a full and incremental backup schedule similar to that shown in Table below.

| Day of Week | Dump Level |
|---|---|
| **Sunday** | Level 0 (full dump). Eject the tape when done. |
| **Monday** | Level 9 (incremental dump). |
| **Tuesday** | Level 8 (incremental dump). |
| **Wednesday** | Level 7 (incremental dump). |
| **Thursday** | Level 6 (incremental dump). |
| **Friday** | Level 5 (incremental dump). |
| **Saturday** | Level 4 (incremental dump). |

Note that after the full backup on Sunday, a level 9 incremental dump is done the next day, and a successively lower dump level is done each day after that. This results in all the files that have changed since Sunday being backed up on every single incremental backup. Each incremental backup is thus larger than the previous; the backup contains all of the files from the previous

incremental backup plus any files that have changed since then. This may seem wasteful of storage space on the backup tape, but it will save a lot of time and effort should there be a need to restore the file system.

For example, imagine that our hard drive crashed on Friday. After replacing the hard drive, we can restore the entire file system in two steps: restore the full backup from the prior Sunday and then the most recent incremental backup from Thursday. We can do this because Thursday's backup contains all of the files from Monday, Tuesday, and Wednesday's tape as well as the files that changed after that. If the dump levels had progressed in positive order (level 1 for Monday, level 2 for Tuesday, and so on), *all* of the incremental backups would have to be restored in order to restore the file system to its most current state.

### ☑ Automating Backups with cron:

We can automate most of our backups with shell scripts and the cron daemon. Use the su command to become root, and then cd to the /usr/local/bin directory. Use any text editor to create a shell script called backups.sh that looks similar to the following:

```
#!/bin/sh
#
# backups.sh - A simple backup script, by Thad Phetteplace
#
# This script takes one parameter, the dump level.
# If the dump level is not provided, it is
# automatically set to zero. For level zero (full)
# dumps, rewind and eject the tape when done.
#

if [ $1 ]; then
        level=$1
else
        #
        # No dump level was provided, so set it
        # to zero
        #
        level="0"
fi

/sbin/dump $level'uf' /dev/nrft0 /
/sbin/dump $level'uf' /dev/nrft0 /home
/sbin/dump $level'uf' /dev/nrft0 /var
/sbin/dump $level'uf' /dev/nrft0 /usr
```

```
#
# If we are doing a full dump, rewind and eject
# the tape when done.
#
if [ $level = "0" ]; then
        #
        # Note: We should replace this with the /bin/st command
        # instead if we ever switch to a SCSI tape drive.
        #
        /bin/mt -f /dev/nrft0 rewind
        /bin/mt -f /dev/nrft0 offline
fi
```

We may choose to change the partitions being backed up to match our situation, but this script should otherwise work quite well for us. After saving and exiting the editor, change the permissions on the file so that it is executable only by root:

```
# chmod 700 backups.sh
```

We can now back up our entire system by running the backups.sh script when logged in as root. The script accepts the dump level as its only parameter. If we leave the parameter off, it will automatically assume a level zero dump. Thus, the following two commands are equivalent:

```
# backups.sh
# backups.sh 0
```

We may need to customize this script for our situation. For example, we are using the tape device /dev/nrft0. We may be using a different tape device. Whatever device we use, we should probably use the version of its device name that begins with a the letter *n*. That tells the system that after it finishes copying data to the tape, it should *not* rewind the tape. As an example, we used /dev/nrft0 instead of /dev/rst0 in the preceding script. If we had used /dev/rst0, each successive incremental backup would have overwritten the previous one.

Other things that we may change in this script include the partitions being backed up and the dump level at which the tape is ejected. It is not uncommon to eject the tape after the last incremental backup just before performing a full backup.

The most useful thing about this script is that we can easily configure our system to run it automatically. Simply add a few lines to the root crontab file, and the cron daemon will invoke the script on the days and times specified. While logged in as root, enter the crontab command with the -e option:

```
# crontab -e
```

This opens the root crontab file in an editor. Add the following lines at the end of the file:

```
0 22 * * 0 /usr/local/bin/backup.sh 0
0 22 * * 1 /usr/local/bin/backup.sh 9
0 22 * * 2 /usr/local/bin/backup.sh 8
0 22 * * 3 /usr/local/bin/backup.sh 7
0 22 * * 4 /usr/local/bin/backup.sh 6
0 22 * * 4 /usr/local/bin/backup.sh 5
0 22 * * 5 /usr/local/bin/backup.sh 4
```

Save and exit the file. The cron daemon will now run the backup script at 10:00 p.m. (22:00 in military time) every day of the week. This example implements the dump schedule outlined earlier. A full dump is performed on Sunday, and the tape is ejected when it is done. A new tape should be loaded on Monday, and then incremental backups will be written to that same tape for the rest of the week. The next full dump will be written to the end of that tape, unless someone is around on Sunday to eject and replace the tape before 10:00 p.m.

### ☑ Restoring Backed Up Files:

The restore command is used to retrieve files from a backup tape or other medium that was created by dump. We can use restore to recover an entire file system or to interactively select individual files. It recovers files from the specified media and copies them into the current directory (the one we ran the recover command in), re-creating subdirectories as needed. Much as with the dump command, the first parameter passed to the restore command is a list of single character option codes, as shown in Table below.

| Restore Options | Description |
|---|---|
| **-r** | Restore the entire dump archive. |
| **-C** | Compare the contents of the dump file with the files on the disk. This is used to check the success of a restore. |
| **-R** | Start the restore from a particular tape of a multitape backup. This is useful for restarting an interrupted restore. |
| **-X filelist** | Extract only specific files or directories from the archive. This option takes one argument, a list of files or directories to extract. |
| **-T file** | List the contents of the dump archive. If a file or directory is given as an argument, list only the occurrence of that file, directory, or anything within the directory. |
| **-i** | Restore files in interactive mode. |
| **-b blocksize** | Specify the block size of the dump in kilobytes. This option takes a numeric argument. |
| **-D filesystem** | Specify the name of the file system to be compared when using the -C option. The file system name is passed as an argument. |

| | |
|---|---|
| **-F script** | Specify the name of the dump archive to restore from. This option takes an alphanumeric argument. |
| **-h** | If this option is specified, restore re-creates directories marked for extraction but will not extract their contents. |
| **-m** | Files are extracted by inode number instead of name. This is generally not very useful. |
| **-N** | Instead of extracting files, print their names. |
| **-s file#** | Specify the dump file to start with on a multiple file tape. This takes a numeric argument. |
| **-T directory** | Tells restore where to write any temporary files. This is useful if we booted from a floppy (which has no space for temporary files). |
| **-v** | Run in verbose mode. This causes restore to print information about each file as it restores it. |
| **-y** | The restore command will always continue when it encounters a bad block, rather than asking you if we want to continue. |

- **Restoring An Entire File System:**

Let's return to our earlier example of the Friday disk crash. We installed a shiny new hard drive and our backup tapes are in hand. It is time to restore the files. For the purpose of this example, we assume that the crashed drive contained only the /homepartition and that the Red Hat Linux operating system is still intact. If the crashed drive had contained the Red Hat Linux operating system, we would first have to reinstall Red Hat Linux before restoring the backup.

Before any files can be recovered to our new hard drive, an empty file system must be created on it. We will use the mkfscommand to do this. The mkfs command can accept a variety of parameters, but usually you only need to supply the name of the device to create the file system on. Thus, to prepare the new hard drive type:

```
# mkfs /dev/hda6
```

Alternatively, because our /home drive is listed in the /etc/fstab file, we can simply specify the /home mountpoint and mkfs will figure out the correct device. Thus, the preceding command could be replaced with this:

```
# mkfs /home
```

***Caution:*** *We should, of course, exercise extreme caution when using the mkfs command. If we specify the wrong device name, we could unintentionally wipe out all data on an existing file system.*

After creating a file system on our new disk, mount the partition to a temporary mount point.

```
# mkdir /mnt/test
# mount /dev/hda6 /mnt/test
```

This connects the new file system to the /mnt/test directory. Now change into the directory (cd /mnt/test) and use the restorecommand to recover the entire file system off of our backup tape. Of course, it is assumed that we have loaded the tape into the tape drive.

```
# cd /mnt/test
# restore rf /dev/nrft0
```

When the restore is finished, we can unmount the partition and remount it to the appropriate mount point. If we have restored the file system to a different physical partition than it was originally on, be sure to modify the /etc/fstab file appropriately so that the correct partition is mounted next time the system is rebooted.

- **Recovering Individual Files:**

The restore command can also be used to recover individual files and directories. By using restore in interactive mode, we can type a series of restore commands to selectively restore files. To run restore in interactive mode, use the i parameter instead of r:

```
# restore if /dev/nrst0
```

The restore command will then read the file index from the backup tape and present us with a restore prompt. At this prompt, we can type the commands that enable us to select which directories and files to recover. We can navigate the directory structure of the backup index much the same way that we navigate an actual file system using a shell prompt. The interactive restore command even has its own version of the familiar cd and ls commands, as shown in Table below.

| Command | Description |
|---|---|
| **add** | Add a file or directory to the list of files to be extracted. If a directory is marked for extraction, all of the directories and files within it will also be extracted. |
| **cd** | Change the current directory being viewed within the dump archive. Works similar to the cdcommand used at a shell prompt. |
| **delete** | Delete a file or directory from the list of files to be extracted. Deleting a directory from the list results in all of the files and directories within it also being deleted. |
| **extract** | Extract all of the marked files and directories from the archive and write them back to the file system. |
| **help** | List the available commands. |
| **ls** | List the contents of the current directory. If a directory name is provided as an argument, list the contents of that directory. Files or directories marked for extraction have a * character in front of them. |
| **pwd** | Print the full path name of the current directory of the dump archive. |
| **quit** | Exit the interactive restore program. |
| **setmodes** | Do not restore the files; instead, set the modes of already existing files on the target disk to match the modes recorded in the dump file. This is useful for recovering from a restore that was prematurely aborted. |
| **verbose** | Toggles verbose output versus quiet output during the restore process. Verbose output mode will echo information to the screen for every file that is restored. |

As an example, pretend that the user joe has accidentally deleted his Mail subdirectory from his home directory. Joe happens to be our boss, so it is urgent that we recover his files. Here is how we may go about it.

Load the appropriate tape into the tape drive and log in as root. Use the cd command to go to the top of the /home partition, and then run the restore program in interactive mode:

```
# cd /home
# restore if /dev/nrft0
```

Verify that we have the backup tape for the /home partition by entering the ls command. We should see something like the following list of directories, representing users that have home directories in /home:

```
restore > ls
.:
bob/        jane/       joe/        lost+found/ mary/       thad/
```

Yes, this is the home partition. Now change the current directory to Joe's home directory using the cd command. Type **ls**again to view the contents of his home directory.

```
restore > cd joe
restore > ls
./joe:
.mozilla/           Desktop/            report.html
.tcshrc             Mail/               letter.txt
.xinitrc            News/               www/
```

Now mark the Mail directory for extraction using the add command:

```
restore > add Mail
```

If we use the ls command again, we see that the Mail directory is preceded with an asterisk (*) character, which means it has been marked for extraction.

```
restore > ls
./joe:
.mozilla/            Desktop/            report.html
.tcshrc             *Mail/               letter.txt
.xinitrc             News/               www/
```

Now use the extract command to begin the extraction process. Restore will prompt for the number of the tape to start with. This is a single tape backup, so just enter the number 1. When it prompts to "set owner/mode for '.'?," answer yes by typing y and pressing Enter. Restore will

then set restore the file permissions (if necessary) of the directory it is restoring to. This isn't critical when extracting individual files like this, but we should always answer yes to this prompt when doing a full restore. Anyway, your screen should now contain the following:

```
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you
should start with the last volume and work toward the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] y
restore >
```

At this point, the files have been recovered and we can exit the restore program by issuing the quit command. That's all there is to it. We now know the basics of using the dump and restore commands.

### ❖ Configuring Amanda for Network Backups

Using Amanda (the Advanced Maryland Automatic Network Disk Archiver), we can use a single large-capacity tape drive on a server to back up files from multiple computers over a network. The Amanda package includes a variety of commands. The online man page for Amanda describes the commands as shown in Table 13-7.

| Command | Description |
|---|---|
| **amdump** | Do automatic Amanda backups. It is normally run by cron on a computer called the tape server host and requests backups of file systems located on backup clients. amdump backs up all disks in the disklist file to tape or, if there is a problem, to a special holding disk. After all backups are done, amdump sends mail reporting failures and successes. |
| **amflush** | Flush backups from the holding disk to tape. amflush is used after amdump has reported it could not write backups to tape for some reason. When this happens, backups stay in the holding disk. After the tape problem is corrected, run amflush to write backups from the holding disk to the tape. |
| **amcleanup** | Clean up after an interrupted amdump. This command is only needed if amdump was unable to complete for some reason, usually because the tape server host crashed while amdumpwas running. |
| **amrecover** | Provide an interactive interface to browse the Amanda index files and select which tapes to recover files from. amrecover can also run amrestore and the system restore program (such as tar) sometimes. |
| **amrestore** | Read an Amanda tape, searching for requested backups. amrestore is suitable for everything from interactive restores of single files to a full restore of all partitions on a failed disk. |
| **amlabel** | Write an Amanda format label onto a tape. All Amanda tapes must be labeled with amlabel. amdump and amflush will not write to an unlabeled tape. |
| **amcheck** | Verify the correct tape is in the tape drive and that all file systems on all backup client systems are ready to be backed up. Can optionally be run by cron before amdump, so someone will get mail warning that backups will fail unless corrective action is taken. |

| | |
|---|---|
| **amadmin** | Take care of administrative tasks, such as finding out which tapes are needed to restore a file system, forcing hosts to do full backups of selected disks, and looking at schedule balance information. |
| **amtape** | Take care of tape changer control operations, such as loading particular tapes, ejecting tapes, and scanning the tape rack. |
| **amverify** | Check Amanda backup tapes for errors (GNU tar format backups only). |
| **amrmtape** | Delete a tape from the tape list and from the Amanda database. |
| **amstatus** | Give the status of a running amdump. |

The amdump command is the one that we will use the most, but before we can get started, we need to configure a few things on both the backup server (the system with the tape drive) and the backup clients (the systems being backed up).

- **Creating Amanda Directories:**

We need to create some directories to hold the Amanda configuration files and to provide a location to write Amanda log files. The configuration files go in an /etc/amanda directory, and the log files go in /var/lib/amanda. In both cases, we should log in as the amanda user and create subdirectories within those directories, one subdirectory for each backup schedule that we intend to run and an index file, as shown here:

***Note:*** *For security reasons, we should do all amanda administration as the amanda user. To do this, the root user should create a password for amanda by typing passwd amanda and entering the new password. The rest of this procedure assumes that you are logged in as the amanda user.*

```
# su - amanda
$ mkdir -p /var/lib/amanda/normal
$ mkdir –p /var/lib/amanda/normal/index
$ mkdir -p /etc/amanda/normal
```

For the purpose of this example, we have created only a normal backup configuration that backs up the data drives on several machines. We may also decide to create an upgrade backup configuration that backs up the operating system partitions. We could then run that backup before we perform any operating system upgrades.

We also need to specify a holding disk that Amanda can use to temporarily spool backups before it writes them to disk. This directory should have a lot of free space. We have a large /home partition on my server, so we created an Amanda directory there to use as a holding disk:

```
# mkdir /home/amanda
# chmod 700 /home/amanda
# chown amanda /home/amanda
# chgrp disk /home/amanda
```

- **Creating the amanda.conf file:**

Next, as the amanda user, we must create two configuration files for Amanda and store them in the /etc/amanda/normaldirectory: amanda.conf and disklist. We can start by copying samples of these files from the /etc/amanda/DailySet1 directory as follows:

```
$ cd /etc/amanda/DailySet1
$ cp amanda.conf disklist /etc/amanda/normal
```

The amanda.conf file sets a variety of general configuration values, and the disklist file defines which machines and partitions to back up. The amanda.conf file can be rather complicated, but, fortunately, most of its values can be left at their default values. Here is a simplified amanda.conf file with some comments embedded in it to help explain things:

```
#
# amanda.conf - sample Amanda configuration file. This started life as
#               the actual config file in use at CS.UMD.EDU.

org "GLACI"        # your organization name for reports
mailto "amanda"    # space separated list of operators at your site
dumpuser "amanda"  # the user to run dumps under

# Specify tape device and/or tape changer. If you don't have a tape
# changer, and you don't want to use more than one tape per run of
# amdump, just comment out the definition of tpchanger.

runtapes 1              # number of tapes to be used in a single
                        # run of amdump
tapedev "/dev/nrft0"    # the no-rewind tape device to be used
rawtapedev "/dev/rft0"  # the raw device to be used (ftape only)

tapetype HP-DAT   # what kind of tape it is
         # (see tapetypes below)
labelstr "^normal[0-9][0-9]*$"   # label constraint all
                                 # tapes must match

# Specify holding disks. These are used as a temporary
# staging area for dumps before they are written to tape and
# are recommended for most sites.

holdingdisk hd1 {
```

```
    comment "main holding disk"
    directory "/home/amanda"   # where the holding disk is
    use 290 Mb                 # how much space can we use on it
    chunksize -1               # size of chunk
    }

# Note that, although the keyword below is infofile, it is
# only so for historic reasons, since now it is supposed to
# be a directory (unless you have selected some database
# format other than the `text' default)

infofile "/usr/adm/amanda/normal/curinfo"   # database DIRECTORY
logdir   "/usr/sdm/amanda/normal"      # log directory
indexdir "/usr/adm/amanda/normal/index"   # index directory

# tapetypes

# Define the type of tape you use here, and use it in "tapetype"
# above. Some typical types of tapes are included here. The
# tapetype tells amanda how many MB will fit on the tape, how
# big the filemarks are, and how fast the tape device is.

define tapetype HP-DAT {
    comment "DAT tape drives"
    # data provided by Rob Browning <rlb@cs.utexas.edu>
    length 1930 mbytes
    filemark 111 kbytes
    speed 468 kbytes
}

# dumptypes
#
# These are referred to by the disklist file.

define dumptype global {
    comment "Global definitions"
```

```
    # This is quite useful for setting global parameters, so you
    # don't have to type them everywhere.
}

define dumptype always-full {
    global
    comment "Full dump of this filesystem always"
    compress none
    priority high
    dumpcycle 0
}
```

This example amanda.conf was trimmed down from a larger example we copied from the /etc/amanda/DailySet1 directory. The example amanda.conf file provides additional information on the available configuration options. Also, the online man page for Amanda should be helpful (type man amanda to read it). We can find more instructions in the /usr/share/doc/amanda-server*directory. Generally, you have to do the following:

- Modify the org name for reports.
- Change the device names set for tapedev and rawtapedev to match our tape device.
- Select a tape type entry that is appropriate for our tape drive.
- Change the name of the directory specified in the holding disk section to match the directory we created earlier.

❖ **Creating a disklist file:**

We also must create a disklist file in the /etc/amanda/normal directory. This simply contains a list of the systems and disk partitions to back up. The qualifier always-full is included on each entry to tell Amanda what type of backup to perform. It means to use full, rather than incremental, backups.

```
# sample Amanda2 disklist file
#
# File format is:
#
#       hostname diskdev dumptype [spindle [interface]]
#
# where the dumptypes are defined by you in amanda.conf.

dexter hda5 always-full
dexter hda6 always-full
dexter hda7 always-full
```

```
dexter hda8 always-full


daffy hda5 always-full
daffy hda6 always-full
daffy hda7 always-full
daffy hdb1 always-full
daffy hdb2 always-full
```

This example file backs up two systems, dexter and daffy. The order of the systems and the partitions is selected so that the most important data is backed up first. This way, if a tape drive becomes full, we have still managed to back up the most important data.

- **Adding Amanda Network Services:**

Amanda is designed to perform backups over a network. The following amanda services are defined in the /etc/services file:

```
amanda          10080/udp
amanda          10080/tcp
amandaidx       10082/tcp
amidxtape       10083/tcp
```

- ***On The Amanda Server:***

To offer these services to the network in Red Hat Linux, we need to configure the xinetd daemon to listen for those services. We do this by enabling the amandaidx and amidxtape services by typing the following (as root user):

```
# chkconfig amidxtape on
# chkconfig amandaidx on
```

This enables Amanda to accept requests from the client system and to start the backup process without any user intervention. We need to tell the xinetd daemon to reload the /etc/xinetd.d files before this change takes place. We can do this by typing the following as root user:

```
# /etc/init.d/xinetd restart
```

- ***On Each Amanda Client***

We next need to configure the .amandahosts file in the /var/lib/amanda directory on each computer (client) that the amanda server will backup from. This file should contain the fully qualified host and domain name of any backup servers that will connect to this client. When we begin, only our localhost is defined in this file as our backup server. To add another computer as

a backup server, we could type the following (replacing *amandahost* with the name of the backup server, while we are logged in as the amanda user):

```
$ echo amandahost >> /var/lib/amanda/.amandahosts
```

We also need to make sure that the amanda client daemon is configured to run on the client. We do this by enabling the amanda service by typing the following (as root user) :

```
# chkconfig amanda on
```

This enables the amanda client to communicate with the amanda server. We need to tell the xinetd daemon to reload the /etc/xinetd.d files before this change takes place. We can do this by typing the following as root user:

```
# /etc/init.d/xinetd restart
```

- ***Performing an Amanda backup:***

Now that everything is configured, we are ready to perform an Amanda backup. While logged in as root, type the following command:

```
# /usr/sbin/amdump normal
```

This runs the amdump command and tells it to read the configuration files it finds in the /etc/amanda/normal directory created earlier. It then works its way down the list of systems and partitions in the disklist file, backing up each partition in the order it occurs. The results of the amdump are written to the /var/lib/amanda/normal directory. Read the files we find there to check on the results of the backup.

We can, of course, automate this process with cron. To create an amdump schedule similar to the regular dump schedule discussed in an earlier section, do the following. While logged in as root, enter the crontab command with the -e option:

```
# crontab -e
```

This opens the root crontab file in an editor. Add the following lines to the end of the file:

```
0 22 * * 0 /usr/sbin/amdump normal
0 22 * * 1 /usr/sbin/amdump incremental
0 22 * * 2 /usr/sbin/amdump incremental
0 22 * * 3 /usr/sbin/amdump incremental
0 22 * * 4 /usr/sbin/amdump incremental
0 22 * * 5 /usr/sbin/amdump incremental
0 22 * * 6 /usr/sbin/amdump incremental
```

Save and exit the file. The cron daemon will now run amdump at 10:00 p.m. (22:00 in military time) every day of the week. This example assumes that a second incremental configuration has

been created. We can do this by creating a subdirectory named incremental under /etc/amanda and populating it with appropriately modified amanda.conf and disklist files. You must also create a subdirectory named incremental under /usr/adm/amanda so that amanda has somewhere to write the logfiles for this configuration.

It may be a bit of work to get it all in place, but once we do, Amanda can make our network backups much easier to manage. It may be overkill for a small office, but in a large enterprise network situation, it enables Red Hat Linux to act as a powerful backup server.

### ☑ Using The Pax Archiving Tool

Over the years, a variety of UNIX operating systems have arisen, resulting in a variety of similar but incompatible file archiving formats. Even tools that go by the same name may use slightly different storage formats on different systems. This can lead to big problems when trying to archive and retrieve data in a multiplatform environment. Fortunately, there is a solution.

The pax program is a POSIX standard utility that can read and write a wide variety of archive formats. An RPM package for pax is included with Red Hat Linux. If it is not already installed, copy the pax-* RPM file from your distribution media (CD #1), or download it from a Red Hat Linux FTP site, and then use the rpm command to install it.

```
# rpm -i pax-*
```

Remember we need to be logged in as root when installing software with the rpm command.

Pax takes a variety of command-line options. The last parameter is usually the file or directory to archive. We may use wildcard characters such as "*" or "?" to specify multiple files or directories. The options we will use most often include the -rand -w parameters for specifying when we are reading or writing an archive. These are usually used in conjunction with the -fparameter, which is used to specify the name of the archive file.

By using pax parameters in different combinations, it is possible to extract an archive, create an archive, list the contents of an archive, or even copy an entire directory hierarchy from one location to another. Table below shows a few examples of the pax command in action.

| Pax Command | Description |
|---|---|
| **pax -f *myfiles*** | List the contents of the archive named myfiles. |
| **pax -r -f *myfiles*** | Extract the contents of the archive named myfiles. |
| **pax -w -f *myfiles* /etc** | Create an archive named myfiles containing everything within the /etc directory. |
| **pax -w -f *myfiles* *.txt** | Archive all of the files in the current directory that have a .txt file extension. |
| **pax -r -w /*olddir* /*newdir*** | Copy the entire contents of the directory /oldir into a new directory called /newdir. |
| **pax -w -B 1440000 -f /dev/fd0 *** | Archive the contents of the current directory onto multiple floppy disks. |
| **pax -w -x cpio -f *myfiles* *** | Archive the contents of the current directory into an archive file named myfiles using the cpio format. |

| | |
|---|---|
| **pax -r -U mary -f *backups*** | Extract all of the files owned by user mary from the archive named backups. |

Note that by leaving off both the -r and -w options, we cause pax to simply list the contents of the archive. If we specify both the -r and -w options, then we should leave off the -f option and supply source and destination directories instead. This will cause the source directory to be completely cloned in the specified destination directory.

We can use additional parameters to further modify pax's behavior. For example, we may use the -x option in conjunction with the -w option to specify the specific archive type to create, or we may use the -B option to specify the number of bytes to write to each volume of a multi-volume archive.

Table below briefly describes the many optional parameters to the pax command.

| Pax Options | Description |
|---|---|
| **-r** | Read files from an archive. |
| **-w** | Write files to an archive. |
| **-a** | Append files to a previously created archive. |
| **-b *blocksize*** | Specify the archive's data block size. It must be a multiple of 512. |
| **-c** | Match all files except those that match the specified pattern. |
| **-d** | Match filename wildcards against file or directory names only, not the complete path. |
| **-f *archive*** | Specify the name of the archive. |
| **-i** | Interactively rename files when archiving. |
| **-k** | Do not overwrite existing files. |
| **-l** | Link files with hard links when in copy mode (-r -w). |
| **-n** | Match only the first file that matches the supplied pattern. |
| **-o *options*** | Extra options specific to the archiving format used. |
| **-p *string*** | Specify the file characteristics to retain when archiving or copying. Read the pax man page for more information on this option. |
| **-s *replstr*** | Modify the archived filenames using the supplied regular expression. |
| **-t** | Preserve the access times of archived files. |
| **-u** | Do not overwrite files with older versions. |
| **-v** | Provide verbose output when running. |
| **-x *format*** | Specify format of the archive. Valid formats include cpio, bcpio, sv4cpio, sv4crc, tar, and ustar. The default is to use ustar when creating an archive. Pax will automatically determine the correct file type when reading an archive. |
| **-z** | Indicates that gzip should be used to compress/decompress the archive. |
| **-B *bytes*** | Specify the number of bytes per archive volume. Use this option to create multivolume archives on removable media. |
| **-D** | Do not overwrite existing files with files that have an older inode modification time. |
| **-E *limit*** | Limit the number of times pax will retry on encountering a read or write error. |
| **-G *group*** | Select files based on a group name or GID. To select by GID, place a # sign in front of the group number. |

| | |
|---|---|
| **-H** | Follow only command-line symbolic links while performing a physical file system traversal. |
| **-L** | Follow all symbolic links when traversing a directory hierarchy. |
| **-P** | Do not follow symbolic links. This is the default. |
| **-T *time*** | Select files based on their modification time. Read the pax man page for complete discussion of this parameter's syntax. |
| **-U *user*** | Select files based on the owner's user name, or by UID with a # sign in front of it. |
| **-X** | Do not traverse into directories that reside on a different device. |
| **-Y** | This option is the same as the -D option, except that the inode change time is checked using the pathname created after all the filename modifications have completed. |
| **-Z** | This option is the same as the -u option, except that the modification time is checked using the pathname created after all the filename modifications have completed. |

As we can see, pax is a very flexible and powerful archiving tool. It can be particularly helpful in migrating data from older legacy systems to your new Linux system. When we are faced with the task of recovering archived data from an antiquated or even nonfunctioning UNIX system, the multiple file format support of pax can be a literal lifesaver.

## Using Password Protection:

Passwords are the most fundamental security tool of any modern operating system and consequently, the most commonly attacked security feature. It is natural to want to choose a password that is easy to remember, but very often this means choosing a password that is also easy to guess. Crackers know that on any system with more than a few users, at least one person is likely to have an easily guessed password.

By using the "brute force" method of attempting to log in to every account on the system and trying the most common passwords on each of these accounts, a persistent cracker has a good shot of finding a way in. Remember that a cracker will automate this attack, so thousands of login attempts are not out of the question. Obviously, choosing good passwords is the first and most important step to having a secure system.

Here are some things to avoid when choosing a password:

- Do not use any variation of our login name or our full name. Even if we use varied case, append or prepend numbers or punctuation, or type it backwards, this will still be an easily guessed password.
- Do not use a dictionary word, even if we add numbers or punctuation to it.
- Do not use proper names of any kind.
- Do not use any contiguous line of letters or numbers on the keyboard (such as "qwerty" or "asdfg").

❖ **Choosing Good Passwords:**

A good way to choose a strong password is to take the first letter from each word of an easily remembered sentence. The password can be made even better by adding numbers, punctuation, and varied case. The sentence we choose should have meaning only to us, and should not be publicly available (choosing a sentence on our personal Web page is a bad idea). Table below lists examples of strong passwords and the tricks used to remember them.

| Password | How to Remember it |
|---|---|
| **Mrci7yo!** | My rusty car is 7 years old! |
| **2emBp1ib** | 2 elephants make BAD pets, 1 is better |
| **ItMc?Gib** | Is that MY coat? Give it back |

The passwords look like gibberish, but are actually rather easy to remember. As we can see, we can place emphasis on words that stand for capital letters in the password. We set our password using the passwd command. Type the passwd command within a command shell, and it will enable us to change our password. First, it will prompt us to enter our old password. To protect against someone "shoulder surfing" and learning our password, the password will not be displayed as we type.

Assuming we type our old password correctly, the passwd command will prompt us for the new password. Once we type in our new password, the passwd command checks the password against cracklib to determine if it is a *good* or *bad*password. Non-root users will be required to try a different password if the one they have chosen is not a good password. The root user is the only user who is permitted to assign *bad* passwords. Once the password has been accepted by cracklib, the passwd command will ask us to enter the new password a second time to make sure there are no typos (which are hard to detect when we can't see what we are typing). When running as root, it is possible to change a user's password by supplying that user's login name as a parameter to the passwd command. Typing this:

```
# passwd joe
```

Results in the passwd command prompting us for joe's new password. It does not prompt us for his old password in this case. This allows root to reset a user's password when that user has forgotten it (an event that happens all too often).

- **Using A Shadow Password File:**

In early versions of UNIX, all user account and password information was stored in a file that all users could read (although only root could write to it). This was generally not a problem because the password information was encrypted. The password was encrypted using a *trapdoor algorithm*, meaning the non-encoded password could be encoded into a scrambled string of characters, but the string could not be translated back to the non-encoded password.

How does the system check our password in this case? When we log in, the system encodes the password we entered, compares the resulting scrambled string with the scrambled string that is stored in the password file, and grants us access only if the two match. Have we ever asked a system administrator what the password on our account is only to hear, "I don't know" in response? If so, this is why: The administrator really doesn't have the password, only the encrypted version. The non-encoded password exists only at the moment we type it.

- ***Breaking Encrypted Passwords:***

There is a problem with people being able to see encrypted passwords, however. Although it may be difficult (or even impossible) to reverse the encryption of a trapdoor algorithm, it is very easy to encode a large number of password guesses and compare them to the encoded passwords in the password file. This is, in orders of magnitude, more efficient than trying actual login attempts for each user name and password. If a cracker can get a copy of our password file, the cracker has a much better chance of breaking into our system.

Fortunately, Linux and all modern UNIX systems support a shadow password file by default. The shadow file is a special version of the passwd file that only root can read. It contains the encrypted password information, so passwords can be left out of the passwd file, which any user on the system can read. Linux supports both the older, single password file method as well as the newer shadow password file. We should always use the shadow password file (it is used by default).

- ***Checking For The Shadow Password File:***

The password file is named passwd and can be found in the /etc directory. The shadow password file is named shadow and is also located in /etc. If our /etc/shadow file is missing, then it is likely that our Linux system is storing the password information in the /etc/passwd file instead. Verify this by displaying the file with the less command.

```
# less /etc/passwd
```

Something similar to the following should be displayed:

```
root:DkkS6Uke799fQ:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
  .
  .
  .
mary:KpRUp2ozmY5TA:500:100:Mary Smith:/home/mary:/bin/sh
joe:0sXrzvKnQaksI:501:100:Joe Johnson:/home/joe:/bin/sh
jane:ptNoiueYEjwX.:502:100:Jane Anderson:/home/jane:/bin/sh
bob:Ju2vY7A0X6Kzw:503:100:Bob Renolds:/home/bob:/bin/sh
```

Each line in this listing corresponds to a single user account on the Linux system. Each line is made up of seven fields separated by colon (:) characters. From left to right the fields are the login name, the encrypted password, the user ID, the group ID, the description, the home directory, and the default shell. Looking at the first line, we see that it is for the root account and has an encrypted password of DkkS6Uke799fQ. We can also see that root has a user ID of zero, a group ID of zero, and a home directory of /root, and root's default shell is /bin/sh.

All of these values are quite normal for a root account, but seeing that encrypted password should set off alarm bells in our head. It confirms that our system is not using the shadow

password file. At this point, we should immediately convert our password file so that it uses /etc/shadow to store the password information. We do this by using the pwconv command. Simply log in as root (or use the su command to become root) and enter the pwconv command at a prompt. It will print no messages, but when our shell prompt returns, we should have a /etc/shadow file and our /etc/passwd file should now look like this:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
  .
  .
  .
mary:x:500:100:Mary Smith:/home/mary:/bin/sh
joe:x:501:100:Joe Johnson:/home/joe:/bin/sh
jane:x:502:100:Jane Anderson:/home/jane:/bin/sh
bob:x:503:100:Bob Renolds:/home/bob:/bin/sh
```

Encrypted password data is replaced with an x. Password data moved to /etc/shadow.

There is also a screen-oriented command called authconfig that we can use to manage shadow passwords and other system authentication information. This tool also has features that let us work with MD5 passwords, LDAP authentication, or Kerberos 5 authentication as well. Type authconfig and step through the screens to use it.

To work with passwords for groups, we can use the grpconv command to convert passwords in /etc/groups to shadowed group passwords in /etc/gshadow. If we change passwd or group passwords and something breaks (we are unable to log in to the accounts), we can use the pwunconv and grpunconv commands, respectively, to reverse password conversion.

So, now we are using the shadow password file and picking good passwords. We have made a great start toward securing our system. We may also have noticed by now that security is not just a one-time job. It is an ongoing process, as much about policies as programs. Keep reading to learn more.

## File Security:

### ☑ Ownership of Linux files:

Every file and directory on our Unix/Linux system is assigned 3 types of owner, given below:

- **User:**

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

- **Group:**

A user-group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose we have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, we could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

- **Other:**

Any other user who has access to a file. This person has neither created the file, nor he/she belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when we set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like we do not want our colleague, who works on our Linux computer, to view our images. This is where **Permissions** set in, and they define **user behavior**.

☑ **PERMISSIONS:**

Every file and directory in our UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read (r):**

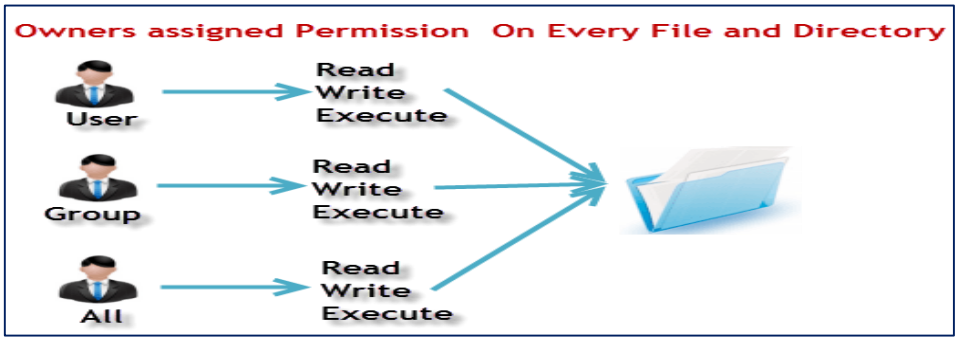
This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

- **Write (w):**

The write permission gives us the authority to modify the contents of a file. The write permission on a directory gives us the authority to add, remove and rename files stored in the directory. Consider a scenario where we have to write permission on file but do not have write permission on the directory where the file is stored. We will be able to modify the file contents. But we will not be able to rename, move or remove the file from the directory.

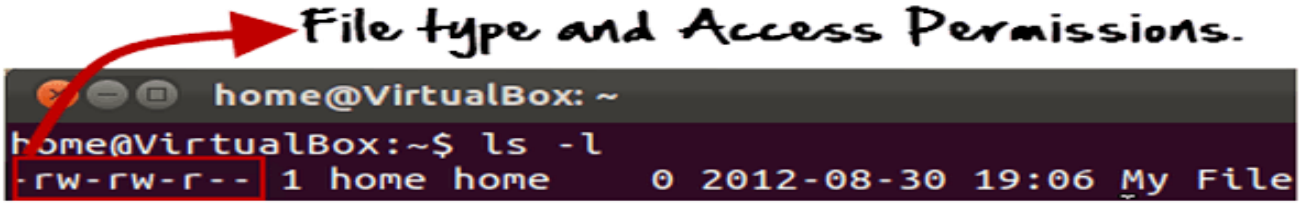
- **Execute (x):**

In Windows, an executable program usually has an extension ".exe" and which we can easily run. In Unix/Linux, we cannot run a program unless the execute permission is set. If the execute permission is not set, we might still be able to see/modify the program code (provided read & write permissions are set), but not run it.



### ☑ LS - L COMMAND:

```
ls - l
```




Here, we have highlighted **'-rw-rw-r--'**and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world.

Here, the first '**-**' implies that we have selected a file.


Else, if it were a directory, **d** would have been shown.


The characters are pretty easy to remember.

**r = read permission**
**w = write permission**
**x = execute permission**
**- = no permission**

The first part of the code is **'rw-'**. This suggests that the owner 'Home' can:


**Read the file**
**Write or edit the file**
**He cannot execute the file since the execute bit is set to '-'.**

By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.

The second part is **'rw-'.** It for the user group 'Home' and group-members can:

**Read the file**
**Write or edit the file**
**Cannot execute the file since the execute bit is set to '-'**

The third part is for the world which means any user. It says **'r--'.** This means the user can only:

**Read the file**


### ☑ CHANGING FILE/DIRECTORY PERMISSIONS WITH 'CHMOD' COMMAND:

Say we do not want our colleague to see our personal images. This can be achieved by changing file permissions.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

**Syntax:** chmod permissions filename

There are 2 ways to use the command:

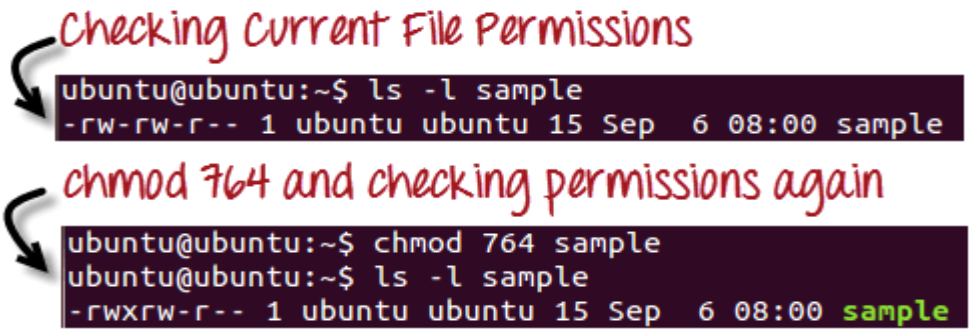
1. ***Absolute(Numeric) Mode:***

In this mode, file **permissions are not represented as characters but a three-digit octal number**.

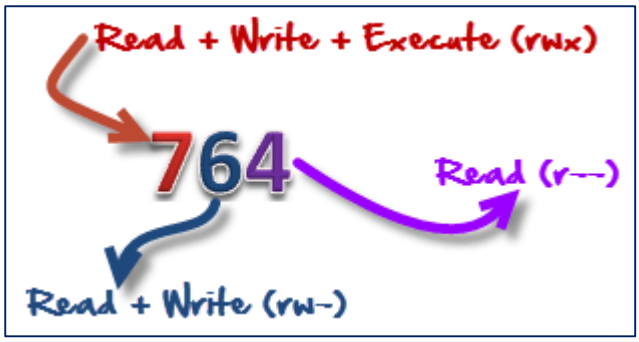
The table below gives numbers for all for permissions types.

| Number | Permission Type | Symbol |
|---|---|---|
| **0** | No Permission | --- |
| **1** | Execute | --x |
| **2** | Write | -w- |
| **3** | Execute + Write | -wx |
| **4** | Read | r-- |
| **5** | Read + Execute | r-x |
| **6** | Read +Write | rw- |
| **7** | Read + Write +Execute | Rwx |

Let's see the chmod command in action.




In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.




'764' absolute code says the following:

**Owner can read, write and execute**
**Usergroup can read and write**
**World can only read**

**This is shown as '-rwxrw-r-**

This is how we can change the permissions on file by assigning an absolute number.

2. ***Symbolic Mode:***

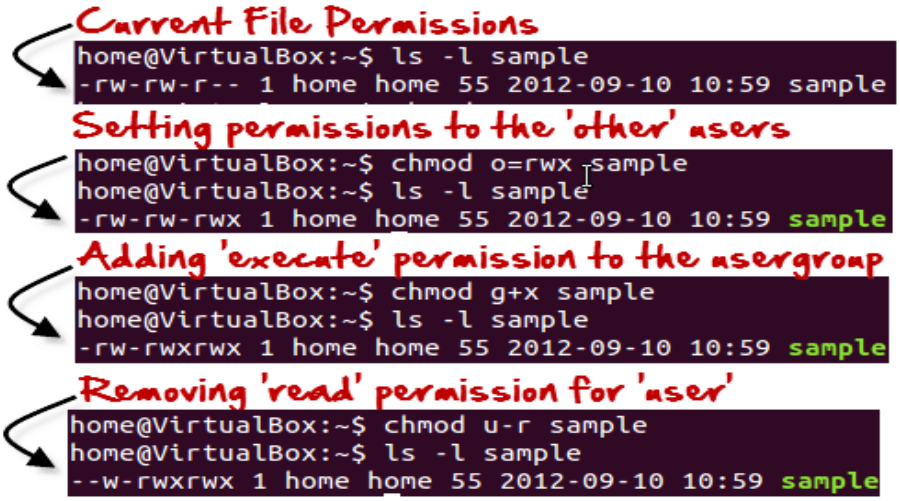
In the Absolute mode, we change permissions for all 3 owners. In the symbolic mode, we can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

| Operator | Description |
|---|---|
| **+** | Adds a permission to a file or directory |
| **-** | Removes the permission |
| **=** | Sets the permission and overrides the permissions set earlier. |

The various owners are represented as:

| User Denotations | Description |
|---|---|
| **u** | user/owner |
| **g** | group |
| **o** | other |
| **a** | all |

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example




### ☑ Changing Ownership and Group:

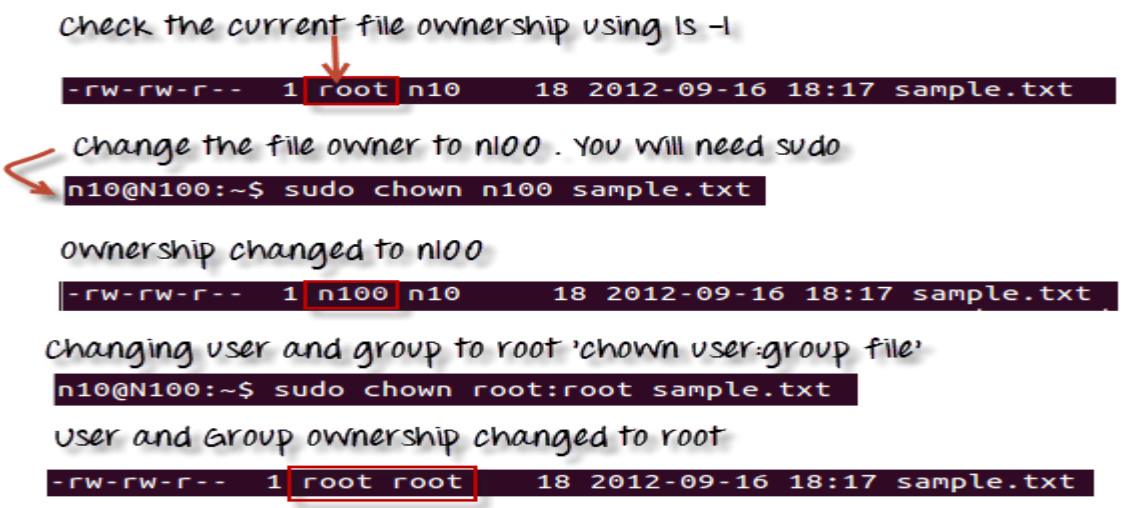
For changing the ownership of a file/directory, we can use the following command:

```
chown user
```

In case we want to change the user as well as group for a file or directory use the command

```
chown user:group filename
```

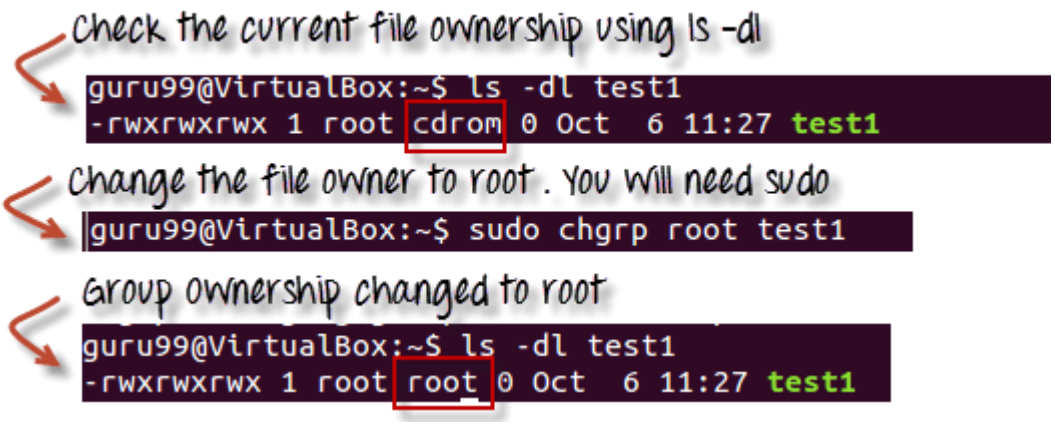
Let's see this in action




In case we want to change group-owner only, use the command

```
chgrp group_name filename
```

'**chgrp**' stands for change group.




**Tip**

- The file /etc/group contains all the groups defined in the system
- We can use the command "groups" to find all the groups we are a member of

```
guru99@VirtualBox:~$ groups
cdrom guru99 adm sudo dip plugdev lpadmin sambashare
guru99@VirtualBox:~$
```

- We can use the command newgrp to work as a member a group other than our default group

```
guru99@VirtualBox:~$ newgrp cdrom
guru99@VirtualBox:~$ cat > test
this is a test to change group
^C
guru99@VirtualBox:~$ ls -dl test
-rw-rw-r-- 1 guru99 cdrom 31 Oct 11 16:39 test
guru99@VirtualBox:~$
```

- We cannot have 2 groups owning the same file.
- We do not have nested groups in Linux. One group cannot be sub-group of other
- x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-dirs.

### ☑ Finding Files with Unwanted Permissions:

This section provides information on the use of the find command to find and display files and directories with permissions that may be undesirable. All examples in this section will output results in the format typically provided by ls -l.

***All examples will use the following format:***

```
find /dir/name -perm /005 -type f -print0 | xargs -0 ls -l
```

***Explanation of these options:***

/dir/name — Directory being searched
-perm — Instructs find to match files based on given permissions
-type f — Instructs find to match only regular files
-print0 — Sends output to stdout using the null character
xargs — Command that builds another command to execute based on the content of stdin

-0 — Use the null character as the item delimiter

***Find All Files Accessible by 'Other' in Any Way:***

```
$ find ~ -perm /007 -type f -print0 | xargs -0 ls –l
```

This will display all files within our home directory that have read, write, or execute permissions for 'other.' This is useful for getting a general idea of what others can access in our home directory.

***Find All Files Readable by 'Other':***

```
$ find . -perm /004 -type f -print0 | xargs -0 ls –l
```

This will display all files within the current working directory that are world-readable. Many of these files will be 'common' files (such as shared libraries, icons, etc.) located in hidden directories, which are generally harmless. Any private files that appear on this list (e.g., private SSH keys or files on non-hidden directories) should be investigated.

***Find All Files Writable by 'Other':***

```
$ find /extra/research0 -perm /002 -type f -print0 | xargs -0 ls –l
```

This will display all files in the /extra/research0 directory that are writable by 'other.' Any files that appear in this list should be investigated, and in most cases, permissions should be changed to something more restrictive.

***Find All Files Executable by User or Group, and Writable by 'Other':***

```
$ find ~ -perm -102 -type f -print0 | xargs -0 ls -l  # User
$ find ~ -perm -012 -type f -print0 | xargs -0 ls -l  # Group
```

This will display all files in our home directory that are both world-writable and executable by either the user or the group that owns the file. Files that are user-executable and world-writable are a major security vulnerability and should be fixed immediately unless we know exactly what we are doing.

***Find All Files Owned by a Specific Group:***

```
$ find ~ -group NameOfGroup -type f -print0 | xargs -0 ls –l
```

This will display all files in our home directory that are owned by the specified group.

***Find All Files Not Owned by a Specific Group:***

```
$ find . -not -group NameOfGroup -type f -print0 | xargs -0 ls -l
```

### ☑ The File Mask:

When a new file is saved somewhere, it is first subjected to the standard security procedure. Files without permissions don't exist on Linux. The standard file permission is determined by the *mask* for new file creation. The value of this mask can be displayed using the **umask** command:

```
bert:~> umask
0002
```

Instead of adding the symbolic values to each other, as with **chmod**, for calculating the permission on a new file they need to be subtracted from the total possible access rights. In the example above, however, we see 4 values displayed, yet there are only 3 permission categories: *user*, *group* and *other*. The first zero is part of the special file attributes settings. It might just as well be that this first zero is not displayed on our system when entering the **umask** command, and that we only see 3 numbers representing the default file creation mask.

Each UNIX-like system has a system function for creating new files, which is called each time a user uses a program that creates new files, for instance, when downloading a file from the Internet, when saving a new text document and so on. This function creates both new files and new directories. Full read, write and execute permission is granted to everybody when creating a new directory. When creating a new file, this function will grant read and write permissions for everybody, but set execute permissions to none for all user categories. This, before the mask is applied, a directory has permissions *777* or *rwxrwxrwx*, a plain file *666* or *rw-rw-rw-*.

The *umask* value is subtracted from these default permissions after the function has created the new file or directory. Thus, a directory will have permissions of *775* by default, a file *664*, if the mask value is *(0)002*. This is demonstrated in the example below:

```
bert:~> mkdir newdir
bert:~> ls -ld newdir
drwxrwxr-x    2 bert    bert             4096 Feb 28 13:45 newdir/
bert:~> touch newfile
bert:~> ls -l newfile
-rw-rw-r--    1 bert    bert                0 Feb 28 13:52 newfile
```

**Files versus directories**

A directory gets more permissions by default: it always has the *execute* permission. If it wouldn't have that, it would not be accessible. Try this out by chmodding a directory 644!

If we log in to another group using the **newgrp** command, the mask remains unchanged. Thus, if it is set to *002*, files and directories that we create while being in the new group will also be accessible to the other members of that group; we don't have to use **chmod**.

The *root* user usually has stricter default file creation permissions:

```
[root@estoban root]# umask
022
```

These defaults are set system-wide in the shell resource configuration files, for instance /etc/bashrc or /etc/profile.