9

RHCSA-Level System Administration Tasks

CERTIFICATION OBJECTIVES

As the final chapter related to the RHCSA exam, this covers those functional system administration tasks not already covered in other chapters. One part of remote access not already covered is based on the GUI-based sharing enabled through Virtual Network Computing (VNC).

The system administration tasks discussed in this chapter include process management and the use of archives. Furthermore, this chapter also helps you work with repetitive system administration tasks. Some of it happens when you want to have a "life," more when you'd rather be asleep. In this chapter, you'll learn how to schedule both one-time and periodic execution of jobs. That's made possible with the cron and at daemons. In this case, "at" is not a preposition, but a service that monitors a system for one-time scheduled jobs. In a similar fashion, cron is a service that monitors a system for regularly scheduled jobs.

When troubleshooting, system logging often provides the clues that you need to solve a lot of problems. The focus in this chapter is local logging; the network logging capabilities of the rsyslog service is an RHCE skill covered in Chapter 17.

INSIDE THE EXAM

Remote Access

Linux administrators are frequently responsible for a variety of systems in remote locations. You reviewed client options associated with SSH in Chapter 2. In this chapter, you'll learn about the VNC part of this objective:

- Access remote systems using SSH and VNC

System Administration

Linux administrators work on Linux systems in a number of ways. In this chapter, you'll learn various methods for meeting the following RHCSA objectives. The first of these objectives involves fundamental command skills:

- Archive, compress, unpack, and uncompress files using **tar**, **star**, **gzip**, and **bzip2**

These other objectives are more closely related to system administration:

- Identify CPU/memory intensive processes; adjust process priority with **renice**, **kill** processes
- Schedule tasks using cron
- Locate and interpret system log files

CERTIFICATION OBJECTIVE 9.01

Configure Access with VNC

If you're skilled enough with Linux to go for the RHCSA, you've probably already used VNC. It's the standard viewer for KVM-based virtual machines. Given the seamless way it's integrated with the virtual machine manager, the use of VNC to view the VMs created so far in this book is essentially painless.

But VNC can do so much more. It's an excellent option for viewing remote systems in another room or another continent. While VNC is not secure, it can be redirected over secure communication systems such as SSH. VNC communication normally proceeds on port 5900, so that port (and those immediately above it) must be open to enable communication. For example, the first connection to a VNC server would also use port 5901; the second connection would use port 5902, and potentially up to port 5909, as port 5910 is normally reserved for a different service. This communication proceeds through the TCP protocol.

As the intent is to provide remote access to a GUI, this section assumes some GUI desktop environment is installed on the local system. In addition, there are three types of VNC Server packages available for RHEL 6.

- The display associated with a KVM-based virtual machine.
- The GNOME-based VNC server known as vino. It's suitable for users who want others such as tech support to see what they're doing.
- The TigerVNC server, based on the TightVNC server. It's suited for administrators who want remote access to a GUI on a different system.

The VNC display associated with a KVM-based virtual machine is integrated into that system. As no additional configuration is required, beyond the steps discussed in Chapter 2, this chapter does not discuss the display associated with KVM. However, that display does not work simultaneously with vino on RHEL 6.

Just be aware, such connections to VNC displays are inherently less secure than a standard login, as all that's required is knowledge of the right port and perhaps a password. Someone who connects remotely via VNC does not even have to have a username. In addition, the TightVNC and vino servers should not be run on the same system simultaneously.

Install and Configure a TigerVNC Server

While alternatives are available, the simplest way to configure remote VNC communication is with TigerVNC. It includes client and server packages: tigervnc and tigervnc-server. But you don't need tigervnc to connect to tigervnc-server. Some users may prefer an alternative GUI client such as the Remote Desktop Viewer.

As always, the simplest method to install these packages is with the **yum install vinagre tigervnc tigervnc-server** command. Once they are installed, you can start the configuration process in the /etc/sysconfig/vncservers file. The last two lines in the default version of the file are sample configuration directives:

```
# VNCSERVERS="2:myusername"
# VNCSERVERARGS[2]="-geometry 800x600 -nolisten tcp -localhost"
```

The first directive would work for a single username; for example, the following line would work for two usernames. The username associated with the number does not matter; however, the numbers should be consecutive. In addition, the numbers shown here correspond to VNC port numbers. For example, the line shown here matches user michael and elizabeth to port numbers 5901 and 5902, respectively.

```
VNCSERVER="1:michael 2:elizabeth"
```

For now, just specify the geometry with the following directive, which supports connections over TCP and does not require tunneling through an SSH connection.

```
# VNCSERVERARGS[2]="-geometry 800x600"
```

Now save the changes and get the VNC service going. The following sequence of actions will seem counterintuitive. To start the VNC service, you first have to make sure it's stopped, with the following command:

```
# /etc/init.d/vncserver stop
```

Now set up one of the users associated with the VNCSERVER directive just configured. You'll have to do so from that user's account with the **vncserver** command. The first available port is assumed, unless one is specified. The following command specifies a connection through port 5902 (not 2):

```
$ vncserver :2
```

Now check the result with the following command. It should confirm that VNC servers are actually running.

```
# /etc/init.d/vncserver status
```

More detailed configuration is possible. The following command adds more information, which in this case overrides the GUI window dimensions set in the /etc/sysconfig/vncservers configuration file:

```
$ vncserver :2 -geometry 640x400
```

The command adds configuration options to the subject user's home directory, in the .vnc/ subdirectory, which will be discussed shortly. The number included after the colon (:) with the **vncserver** command determines the communications port for that connection. In this case, remote users will be able to connect to this system over port 5902. For the command with the :1, a connection to that system can be made over port 5901. In either case, the **vncserver** command normally prompts for a password, which is used for the connection from the remote system.

The GNOME-Based vino Server

The GNOME-based vino server allows administrators to view the current state of remote GUI desktop environments. The necessary tools are included in the package of the same name. Once it is installed, run the **vino-preferences** command from the desired account, to open the Remote Desktop Preferences window shown in Figure 9-1.

FIGURE 9-1

Remote Desktop Preferences with vino

The options associated with the Remote Desktop Preferences window are straightforward. In addition, in most cases, if you hover a cursor over an option, the tool provides additional explanation. As the objective is to share the local desktop environment over a network, all other options are grayed out unless you activate the first option: Allow Other Users To View Your Desktop.

When you activate that first option, the Remote Desktop Preferences window changes; it evaluates the current configuration to identify whether and how the desktop may be accessible over the local network. That changes, depending on the other now-active options in the window. Each option is described as follows:

- **Allow Other Users To Control Your Desktop** If this option is active, remote users will be able to enter keystrokes and mouse clicks remotely onto your desktop environment.
- **You Must Confirm Each Access To This Machine** If this option is active, you'll get a chance to confirm remote requests to access the local GUI desktop environment. It's an excellent way to allow local users to retain control over their local systems, perhaps until prompted by a technical support person who is trying to help them.
- **Require The User To Enter This Password** If this option is active, it provides one additional security measure for the local user.
- **Configure Network To Automatically Accept Connections** If a local network router works with universal plug and play (UPnP), this option enables access from remote networks.

Only one of the final three options can be active. As they relate to a notification icon, they have no bearing on the configuration of vino as a server. They just determine if and when an icon associated with the vino server is shown in the upper-right corner area of the GNOME desktop environment.

The vino server does not work with the standard configuration of a physical host and virtual machines created in the first two chapters of this book. If KVM-based virtual machines are currently running, they should be deactivated. You'll then be able to connect to the vino-based VNC server from a remote system, using either the **vncviewer** command or an option like the Remote Desktop Preferences tool.

Incidentally, the Remote Desktop Preferences tool saves its settings in an XML file in the subject user's home directory, in the .gconf/desktop/gnome/remote_access/ subdirectory.

Although the vino server requires the same open ports as the aforementioned TigerVNC server, the two VNC servers should not be run simultaneously. If you're running into a problem with vino, I did warn you about this issue earlier in the chapter.

Install and Configure a VNC Client

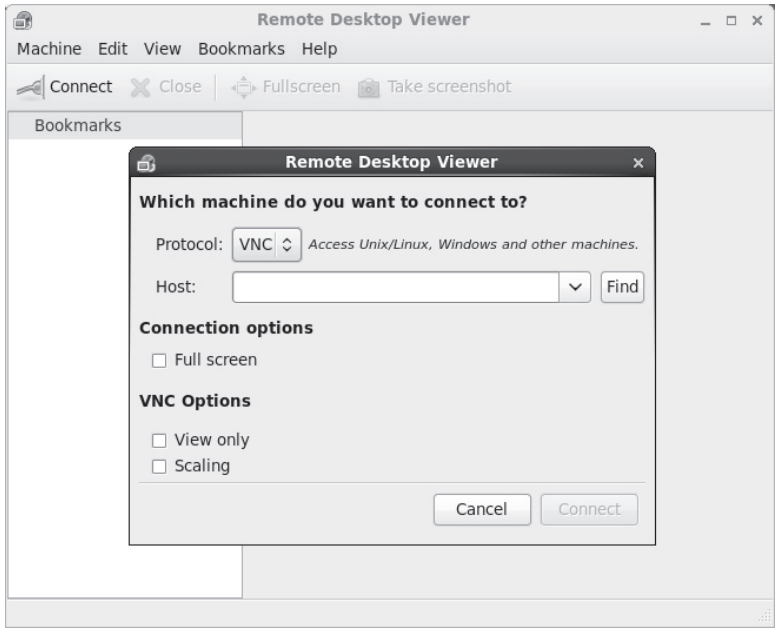
You should then be able to test the result locally, with the **vncviewer** command. But strangely enough, that action fails with a "couldn't find suitable pixmap" error message. If you're already on a remote system, that command will work with the IP address of the VNC Server, coupled with the associated port. For example, if the previous **vncserver :2** command has been run on a local 192.168.122.1 system, you should be able to connect remotely through port 5902 with the following command:

```
# vncviewer 192.168.122.1:2
```

To test a VNC server on a local system, you can use the Remote Desktop Viewer, which can be started from a GUI desktop with the **vinagre** command or from the GNOME desktop by clicking Applications | Internet | Remote Desktop Viewer. It opens the Remote Desktop Viewer window. Click Connect to open the Remote Desktop Viewer window shown in Figure 9-2.

FIGURE 9-2

Remote Desktop Preferences



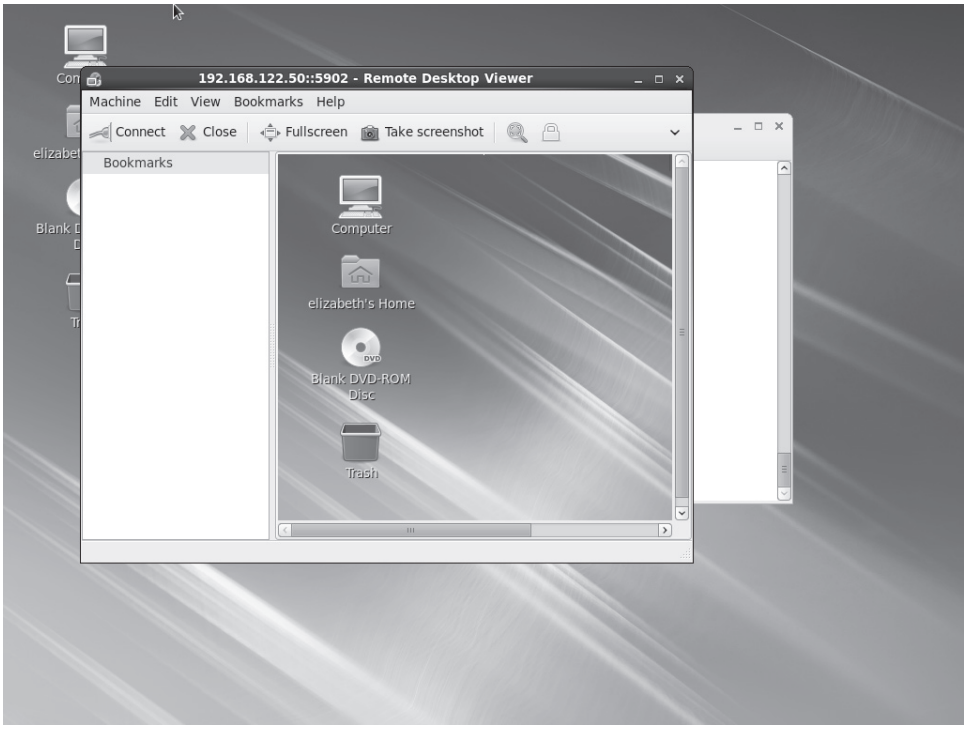
The options are as follows:

- **Protocol** VNC is assumed in this chapter; however, the Remote Desktop Viewer can be used with a variety of protocols.
- Host Specify the hostname or the IP address of the system to which you want to connect.
- **Full Screen** If selected, the remote connection occupies the full screen of the local system.
- **View Only** If selected, no actions are allowed over the connection.
- **Scaling** When selected, changes the screen to allow viewing in different-sized GUI environments.

If successful, it'll open up another Remote Desktop Viewer window, with a request for the password, and a Remember This Credential option, which would store that password. If a connection is made, you'll see the "remote" desktop on the local system, as shown in Figure 9-3.

FIGURE 9-3

A local connection to a "Remote" Desktop via VNC



But that's just a local connection. Before proceeding to Firewall Options, use this command to make sure this VNC server is running the next time this system is booted:

```
# chkconfig vncserver on
```

There's a limit. Only one connection can be made per port. For example, if the previous vncviewer command used port 5902, you can't use the Remote Desktop Viewer to connect to the same port.

Firewall Options

The ports to be open in a firewall depend on the number of connections that may be made to the local VNC server. At minimum, you'll need to open ports 5900 and 5901. The additional ports that you open depend on the number of connections needed from remote systems. While firewalls are covered in Chapters 4 and 10, the following is a brief description of one way you can open ports 5900 through 5905 through a firewall.

1. Run the **system-config-firewall-tui** command.
2. In the console window that appears, select Customize and press ENTER.
3. In the Trusted Services window that appears, select Forward and press ENTER.
4. In the Other Ports window, select Add to open the Port And Protocol window shown in Figure 9-4. The entries shown specify ports 5900 through 5905, over the TCP protocol. Type in those entries, select OK, and the press ENTER.
5. Select Close and press ENTER.
6. Back in the Firewall Configuration window, select OK and press ENTER.
7. When the warning appears about overriding the existing firewall configuration, select Yes and press ENTER.
8. To confirm the new firewall, run the **iptables -L** command. The following line from the output confirms that the ports associated with vnc-server in the /etc/services file, 5900, through 5905, are open in the firewall.

```
ACCEPT tcp -- anywhere anywhere state NEW tcp dpts:vnc-server:5905
```

The local system is now ready to transmit VNC server connections to remote systems, using connection tools such as the Remote Desktop Viewer and the **vncviewer** command.

FIGURE 9-4

Firewall Ports and Protocol

Confirm Access to a VNC Server

This is one case where the **telnet** and **nmap** tools discussed in Chapter 2 may be helpful. The messages associated with the start of various VNC servers may be cryptic. To confirm that a system is listening on appropriate ports from a local system, try the following command:

```
# nmap localhost
```

For my physical host system, that results in the following output:

```
Starting Nmap 5.21 ( http://nmap.org ) at 2011-01-26 12:18 PST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000011s latency).
rDNS record for 127.0.0.1: localhost.localdomain
Not shown: 993 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
5900/tcp open  vnc
5901/tcp open  vnc-1
5902/tcp open  vnc-2
```

The last three lines confirm the availability of three ports for connections to VNC servers. To reiterate, ports 5901 and 5902 are used for different VNC terminals. The ports can be confirmed with **telnet** commands such as the following:

```
# telnet localhost 5900
```

The following output indirectly confirms a connection to a VNC server, as the **RFB** shown in the output refers to a remote framebuffer, a protocol for network access to a GUI:

```
RFB 003.008
```

Of course, remote access to VNC servers won't work unless there's access through the firewall just configured earlier. You can use both the **nmap** and **telnet** commands for that purpose from remote systems. Just substitute the IP address or the hostname of the VNC server for localhost in the commands just shown.

Route Through a Secure Shell

For this example, assume you've configured a VNC server on the server1.example.com system. Furthermore, assume you've set up /etc/sysconfig/vncservers to point to the appropriate port, and have run the following command to set up user elizabeth's account for remote access over port 5903:

```
$ vncserver :3
```

Now navigate to a remote system such as tester1.example.com. From that system, run the following command to set up an SSH tunnel:

```
$ ssh -L 5903:server1.example.com:5903 elizabeth@server1.example.com
```

That **ssh** command with the **-L** specifies that communications from local port 5903 are to be bound to communications over port 5903 from the server1.example.com system. Since the secure shell requires a user login, the **ssh** command requires access via an account on that remote system. Since this communication proceeds over port 5903, it does not interfere with normal ssh communication. However, it also means that you would need to open port 5903 for this particular VNC server.

Once that communications tunnel is established, open up a second command line console, or the Remote Desktop Viewer on the local system. Connect to the **localhost:3** system. That connection is transmitted over port 5903 through the now established SSH tunnel.

More VNC Configuration

When a user establishes a connection by entering a command like **vncserver :3**, that action sets up configuration files in that user's home directory, in the .vnc/ subdirectory. That command sets up four files in the .vnc/ subdirectory: the password, a log file, a process identifier (PID) file, and a configuration file.

When the **vncserver :3** command prompts for a password, an encrypted version of that entry is stored in the passwd file.

The activity associated with a VNC server is a process, with a PID number. On the server1.example.com system, that file is server1.example.com.pid. That file includes the actual PID number. The **kill** command can be used to stop this process, as discussed later in this chapter.

The last two files are more important. The server1.example.com.log file provides information on what happened, and what is happening with the connection. The configuration file is xstartup, which is preconfigured to start the local default desktop environment.

A User VNC Configuration File

The following is a line-by-line analysis of the VNC configuration file that's created when a user activates a VNC server for her system. To review, the file is xstartup in the .vnc/ subdirectory of the target user. The first line is common to many scripts, as it establishes the bash shell via a soft link as the shell for the script:

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

The following line sets up the VNC window as an icon:

```
vncconfig -iconic
```

The **unset** command applied to the SESSION_MANAGER variable allows the new GUI to create its own communications socket:

```
unset SESSION_MANAGER
```

The unset command applied to the DBUS_SESSION_BUS_ADDRESS variable supports the creation of a new message bus for the GUI to be created:

```
unset DBUS_SESSION_BUS_ADDRESS
```

While it's useful to confirm the operating system is Linux with the OS=`uname -s` directive, the if loop that follows applies only to the PATH variable associated with the SUSE Linux distribution. Since this book covers RHEL, that loop is not shown here.

The two **if** loops that follow executes the commands within the /etc/X11/xinit/xinitrc script. The only difference is the **-x** and the **-f** switches, which execute the script depending on whether the noted file has executable permissions. You may already recognize the xinitrc script as the standard script associated with the **startx** command, which starts the GUI from the command line interface.

```
if [ -x /etc/X11/xinit/xinitrc ]; then
  exec /etc/X11/xinit/xinitrc
fi
if [ -f /etc/X11/xinit/xinitrc ]; then
  exec sh /etc/X11/xinit/xinitrc
fi
```

CERTIFICATION OBJECTIVE 9.02

Elementary System Administration Commands

There are several system administration commands in the RHCSA objectives not covered in previous chapters. They're associated with system resource management and archives. System resource management allows you to see what processes are running, to check the resources they're using, and to kill or restart those processes. Archive commands support the consolidation of a group of files in a single archive, which can then be compressed.

System Resource Management Commands

Linux includes a variety of commands that can help you identify those processes that are monopolizing the system. The most basic of those commands is **ps**, which provides a snapshot of currently running processes. Those processes can be ranked with the **top** command, which can display running Linux tasks in order of their resource usage. With **top**, you can identify those processes that are using the most CPU and RAM memory. Commands that can adjust process priority include **nice** and **renice**. Sometimes it's not enough to adjust process priority, at which point it

exam Watch

The objective related to system resource management is to "Identify CPU/memory intensive processes, adjust process priority with renice, and kill processes."

may be appropriate to stop a process with commands like **kill** and **killall**. If you need to monitor system usage, the **sar** and **iostat** commands can also be helpful.

Process Management with the psCommand

It's important to know what's running on a Linux computer. To help with that task, the **ps** command has a number of critical switches. When trying to diagnose a problem, it's common to get the fullest possible list of running processes, and then look for a specific program. For example, if the Firefox Web browser were to suddenly crash, you'd want to kill any associated processes. The **ps aux | grep firefox** command could then help you identify the process(es) that you need to kill.

The **ps** command by itself is usually not enough. All it does is identify those processes running in the current command line shell. Unless you've started a running process with the ampersand (&) to return the command to a shell prompt, that command typically returns just the process associated with the current shell, and the **ps** command process itself.

To identify those processes associated with a username, the **ps -u *username*** command can help. Sometimes there are specific users who may be problematic for various reasons. So if you're suspicious of a user like mjang, the following command can help you review every process currently associated with that user:

```
$ ps -u mjang
```

As an administrator, you may choose to focus on a specific account for various reasons, such as activity revealed by the **top** command, described in the next section. If a bigger picture view is required, you may want to audit all currently running processes with a command like the following:

```
$ ps aux
```

The output is a more complete database of currently running processes, in order of their PIDs. The **a** lists all running processes, the **u** classifies those processes by user, and the **x** lifts the standard limitation that listed processes must be associated with a terminal or console. One example is shown in Figure 9-5. While the output can include hundreds of processes and more, the output can be redirected to a file for further analysis with commands like **grep**. The output columns shown Figure 9-5 are described in Table 9-1.

FIGURE 9-5

Output from the ps aux command

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  19244  1340 ?        Ss   Jan19   0:05 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Jan19   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Jan19   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S    Jan19   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S    Jan19   0:00 [watchdog/0]
root         6  0.0  0.0      0     0 ?        S    Jan19   0:00 [migration/1]
root         7  0.0  0.0      0     0 ?        S    Jan19   0:01 [ksoftirqd/1]
root         8  0.0  0.0      0     0 ?        S    Jan19   0:00 [watchdog/1]
root         9  0.0  0.0      0     0 ?        S    Jan19   0:00 [migration/2]
root        10  0.0  0.0      0     0 ?        S    Jan19   0:05 [ksoftirqd/2]
root        11  0.0  0.0      0     0 ?        S    Jan19   0:00 [watchdog/2]
root        12  0.0  0.0      0     0 ?        S    Jan19   0:00 [migration/3]
root        13  0.0  0.0      0     0 ?        S    Jan19   0:01 [ksoftirqd/3]
root        14  0.0  0.0      0     0 ?        S    Jan19   0:00 [watchdog/3]
root        15  0.0  0.0      0     0 ?        S    Jan19   0:01 [events/0]
root        16  0.0  0.0      0     0 ?        S    Jan19   0:24 [events/1]
root        17  0.0  0.0      0     0 ?        S    Jan19   0:12 [events/2]
root        18  0.0  0.0      0     0 ?        S    Jan19   0:04 [events/3]
root        19  0.0  0.0      0     0 ?        S    Jan19   0:00 [cpuset]
root        20  0.0  0.0      0     0 ?        S    Jan19   0:00 [khelper]
root        21  0.0  0.0      0     0 ?        S    Jan19   0:00 [netns]
root        22  0.0  0.0      0     0 ?        S    Jan19   0:00 [async/mgr]
:
```

TABLE 9-1

Columns of Output from ps aux

Column Title	Description
USER	The username associated with the process
PID	Process Identifier
%CPU	Current CPU usage
%MEM	Current RAM usage
VSZ	Virtual memory size of the process in KB
RSS	Physical memory in use by the process, not including swap space, in KB
TTY	Associated terminal console
STAT	Process status
START	Start time of the process; if you just see a date, the process started more than 24 hours ago
TIME	Cumulative CPU time used
COMMAND	Command associated with the process

Incidentally, you may note that the **ps aux** command does not include the familiar dash in front of the **aux** switches. In this case, the command works with and without the dash; the following alternative includes current environmental variables.

```
$ ps eux
```

Processes can be organized in a tree format. Specifically, the first process, with a PID of 1, is init. That process is the base of the tree, which may be shown with the **pstree** command. In a few cases, it's not possible to use standard **kill** commands described shortly to kill a process. In such cases, it may be possible to kill a process by killing its "parent" in the tree. To that end, the following command identifies the parent of a process, known as the PPID:

```
$ ps axl
```

The **l** switch is not compatible with the **u** switch; in other words, you can't set up the output of the **ps** command to include both the user who started the process and the PPID. You can view the PID and PPIDs of all running processes in Figure 9-6.

With the -Z switch (that's an uppercase Z), the ps command can also identify the SELinux contexts associated with a process. For example, the following command includes the SELinux contexts of each process at the start of the output. If you've read Chapter 4, the contexts should already seem familiar. For example, contrast the context of the vsFTP server process, with the following excerpt:

```
system_u:system_r:ftpd_t:s0-s0:c0.c1023 2059 ? Ss     0:00 /usr/sbin/vsftpd
/etc/vsftpd/vsftpd.conf
```

FIGURE 9-6

Output from the ps axl command

```
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     1     0  20   0  19244  1340 poll_s Ss   ?          0:05 /sbin/init
1     0     2     0  20   0      0     0 kthrea S    ?          0:00 [kthreadd]
1     0     3     2 -100  -      0     0 migrat S    ?          0:00 [migration/0]
1     0     4     2  20   0      0     0 ksofti S    ?          0:00 [ksoftirqd/0]
5     0     5     2 -100  -      0     0 watchd S    ?          0:00 [watchdog/0]
1     0     6     2 -100  -      0     0 migrat S    ?          0:00 [migration/1]
1     0     7     2  20   0      0     0 ksofti S    ?          0:01 [ksoftirqd/1]
5     0     8     2 -100  -      0     0 watchd S    ?          0:00 [watchdog/1]
1     0     9     2 -100  -      0     0 migrat S    ?          0:00 [migration/2]
1     0    10     2  20   0      0     0 ksofti S    ?          0:05 [ksoftirqd/2]
5     0    11     2 -100  -      0     0 watchd S    ?          0:00 [watchdog/2]
1     0    12     2 -100  -      0     0 migrat S    ?          0:00 [migration/3]
1     0    13     2  20   0      0     0 ksofti S    ?          0:01 [ksoftirqd/3]
5     0    14     2 -100  -      0     0 watchd S    ?          0:00 [watchdog/3]
1     0    15     2  20   0      0     0 worker S    ?          0:01 [events/0]
5     0    16     2  20   0      0     0 worker S    ?          0:25 [events/1]
1     0    17     2  20   0      0     0 worker S    ?          0:12 [events/2]
1     0    18     2  20   0      0     0 worker S    ?          0:04 [events/3]
1     0    19     2  20   0      0     0 worker S    ?          0:00 [cpuset]
1     0    20     2  20   0      0     0 worker S    ?          0:00 [khelper]
:
```

Contrast that with the context of the actual daemon. The object role works with the actual daemon; you can review it with other daemons in the /usr/sbin directory. The vsftpd daemon works with the associated configuration file with the ftpd_t type. In contrast, the vsftpd daemon alone is executable with the ftpd_exec_t type.

```
-rwxr-xr-x. root root system_u:object_r:ftpd_exec_t:s0 /usr/sbin/vsftpd
```

The role of different daemons and their corresponding processes should match and contrast in a similar fashion. If they don't, the daemon should not work, and the problem should be documented in the SELinux audit log described in Chapter 4 in the /var/log/audit directory.

View Loads with the top Task Browser

The **top** command sorts active processes first by their CPU load and RAM memory usage. Take a look at Figure 9-7. It provides an overview of the current system status, starting with the current up time, number of connected users, active and sleeping tasks, CPU load, and more. The output is in effect a task browser.

The default sort field is CPU load. In other words, the process that's taking the most in CPU resources is listed first. You can change the sort field with the help of the left and right directional (<, >) keys. Most of the columns are the same as shown in Figure 9-6, as detailed in Table 9-1. The additional columns are described in Table 9-2.

FIGURE 9-7

Output from the top command

```
top - 20:10:14 up 4 days, 20:25,  7 users,  load average: 0.05, 0.04, 0.05
Tasks: 256 total,   1 running, 255 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.2%us,  2.8%sy,  0.0%ni, 94.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7985904k total,  6257164k used,  1728740k free,   205204k buffers
Swap:  9775512k total,    25400k used,  9750112k free,  2429716k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1231 qemu      20   0 1137m 633m 3048 S  6.6  8.1 117:24.28 qemu-kvm
 2619 root      20   0  230m  64m  28m S  4.0  0.8 156:45.17 Xorg
 2971 michael   20   0 1405m 544m  29m S  3.3  7.0 417:44.30 firefox
18809 michael   20   0 1256m  81m  16m S  2.3  1.0 126:40.65 python
 2874 michael   20   0  334m  15m  10m S  2.0  0.2   5:36.19 wnck-applet
 1063 root      20   0  592m  16m 4516 S  1.3  0.2  18:33.44 libvirtd
 3077 michael   20   0 1834m 106m  24m S  1.0  1.4 439:11.63 plugin-containe
 2932 michael   20   0 4784m 775m  51m S  0.7  9.9  31:14.37 swriter.bin
 6137 michael   20   0 15072 1308  900 R  0.7  0.0   0:13.75 top
 2827 michael   20   0  575m  18m 9508 S  0.3  0.2   1:00.75 gnome-settings-
 2844 michael   20   0  433m  19m  10m S  0.3  0.3   2:24.75 metacity
 2893 michael   20   0  289m  15m 9628 S  0.3  0.2   0:37.65 gnome-terminal
 2957 michael   20   0  256m  11m 5580 S  0.3  0.1   0:33.13 gnome-screensav
10870 michael   20   0  244m 9544 7516 S  0.3  0.1   0:00.06 screenshot
    1 root      20   0 19244 1276 1056 S  0.0  0.0   0:05.30 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.04 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.24 migration/0
```

TABLE 9-2 Additional Columns of Output from top

Column Title	Description
PR	The priority of the task; for more information, see the **nice** and **renice** commands
NI	The nice value of the task, an adjustment to the priority
VIRT	The virtual memory used by the task
RES	Physical memory in use by the process, not including swap space, in KB (similar to RSS in the output to the **ps aux** command)
SHR	Shared memory used by a task, which can be reallocated
S	Process status (same to STAT in the output to the **ps aux** command)

One problem with the **top** and **ps** commands is that they display the status of processes on a system as a snapshot in time. That may not be enough. Processes may load a system for just a blip of time, or even periodic blips in time. One way to find more information about the overall load on a system is with two commands from the sysstat package: **sar** and **iostat**. That system activity information is logged courtesy of the **sa1** and **sa2** commands associated with the /etc/cron.d/sysstat script, which will be described shortly.

System Activity Reports with the sarCommand

The **sar** command, in essence, can be used to provide a system activity report. For example, Figure 9-8 shows the output of the **sar -A** command. As you can see, the output shows various CPU measures at different points in time. The default settings measures CPU load at ten-minute intervals. There are four CPU cores on this system, which are measured individually and as a whole. The large idle numbers shown in the figure are a good sign that the CPU is not being overloaded; however, the figure shows the load for less than an hour.

The ten-minute intervals associated with the **sar** command output are driven by a regular job in the /etc/cron.d directory. The output from those reports are collected in log files in the /var/log/sa directory. The filenames are associated with the numeric day of the month; for example, system activity report status for the fifteenth of the month can be found in the sa15 file in the noted directory. However, such reports are normally stored for the last seven days, based on the following default in the /etc/sysconfig/sysstat file:

```
HISTORY=7
```

FIGURE 9-8 Output from the sar -A command

```
Linux 2.6.32-71.el6.x86_64 (Maui)       01/24/2011      _x86_64_        (4 CPU)

12:00:01 AM     CPU      %usr     %nice      %sys   %iowait    %steal      %irq     %soft    %guest     %idle
12:10:01 AM     all      5.09      0.00      2.57      0.18      0.00      0.01      0.01      0.02     92.13
12:10:01 AM       0      4.69      0.00      4.15      0.07      0.00      0.02      0.01      0.04     91.01
12:10:01 AM       1      6.86      0.00      1.40      0.03      0.00      0.02      0.01      0.01     91.67
12:10:01 AM       2      4.00      0.00      3.44      0.55      0.00      0.00      0.01      0.02     91.97
12:10:01 AM       3      4.79      0.00      1.28      0.06      0.00      0.00      0.01      0.00     93.85
12:20:01 AM     all      5.41      0.00      2.74      0.24      0.00      0.02      0.01      0.02     91.56
12:20:01 AM       0      4.39      0.00      4.29      0.10      0.00      0.01      0.01      0.02     91.17
12:20:01 AM       1      7.98      0.00      1.54      0.03      0.00      0.04      0.01      0.00     90.40
12:20:01 AM       2      3.60      0.00      3.81      0.70      0.00      0.01      0.02      0.04     91.83
12:20:01 AM       3      5.69      0.00      1.33      0.11      0.00      0.00      0.01      0.00     92.85
12:30:01 AM     all      6.20      0.00      2.61      0.19      0.00      0.02      0.02      0.02     90.94
12:30:01 AM       0      5.31      0.00      4.53      0.05      0.00      0.03      0.02      0.03     90.03
12:30:01 AM       1      7.84      0.00      1.52      0.02      0.00      0.05      0.02      0.01     90.54
12:30:01 AM       2      4.36      0.00      3.21      0.63      0.00      0.00      0.03      0.03     91.75
12:30:01 AM       3      7.29      0.00      1.21      0.06      0.00      0.00      0.01      0.00     91.43
12:40:01 AM     all      4.80      0.00      2.38      0.22      0.00      0.02      0.01      0.02     92.56
12:40:01 AM       0      4.45      0.00      3.92      0.07      0.00      0.02      0.01      0.03     91.49
12:40:01 AM       1      6.31      0.00      1.27      0.01      0.00      0.07      0.01      0.01     92.32
12:40:01 AM       2      3.39      0.00      3.28      0.67      0.00      0.00      0.02      0.02     92.61
12:40:01 AM       3      5.03      0.00      1.06      0.11      0.00      0.00      0.00      0.00     93.79
--More--
```

CPU and Storage Device Statistics with iostat

In contrast to **sar**, the **iostat** command reports more general input/output statistics for the system, not only for the CPU, but also for connected storage devices, such as local drives and mounted shared NFS directories. An example shown in Figure 9-9 displays information for the CPU and the storage devices on the server1.example.com system.

Variations on sar with sa1 and sa2

The **sa1** and **sa2** commands are often used to collect system activity report data. In the /etc/cron.d/sysstat script, the **sa1** command is used to gather disk activity data every ten minutes. In that same script, the **sa2** command collects all **sar** data in a daily report. As noted in the script, that report is processed every day, at seven minutes before midnight.

nice and renice

The **nice** and **renice** commands can be used to manage the priority of different processes. While the **nice** command is used to start a process with a different priority, the **renice** command is used to change the priority of a currently running process.

FIGURE 9-9

CPU and storage device statistics

```
[root@server1 ~]# iostat
Linux 2.6.32-71.el6.x86_64 (server1.example.com)        01/26/2011      _x86_64_
(2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.05    0.00    0.03    0.03    0.00   99.88

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.01         0.00       2368          0
sdb               0.00         0.01         0.00       2144          0
scd0              0.00         0.03         0.00       7392          0
vda               0.21         2.70         1.97     765626     556858
dm-0              0.00         0.00         0.00        256          0

[root@server1 ~]#
```

Process priorities in Linux specify numbers that seem counterintuitive. The range of available nice numbers can vary from -20 to 19. A process given a priority of -20 takes precedence over all other processes. In contrast, a process given a priority of 19 will wait until the system is almost completely free before taking any resources. The default nice number of a process is 0.

The **nice** command prefaces other commands. For example, if you have an intensive script to be run at night, you might choose to start it with a command like the following:

```
$ nice -n 19 ./intensivescript
```

This command starts the noted script with the lowest possible priority. If started at night (or at some other time when a system is not loaded by other programs), the script is run until just about any other jobs, such as a script in the /etc/cron.* directories, are scheduled for execution. As such scripts are run on a schedule, they normally should take priority over some user-configured programs.

Sometimes a program is just taking up too many resources. If you need to make sure that program continues to run, one step before killing the associated process is to lower its priority with the **renice** command. Normally, the easiest way to identify a process that's taking up too many resources is by its PID in the output to the **top** command. That PID number is in the left-hand column of the output. For example, if you identify a process that's monopolizing current CPU and memory resources, copy the PID number of that process. If that number were 1234, the following command would change the nice number of that process to -10, which gives that process a higher priority than the default of 0.

```
# renice -10 1234
```

Even though the output refers to the "priority," it really is just listing the old and new "nice" numbers for the process:

```
1234: old priority 0, new priority, -10
```

The new nice number is shown in the output to the **top** command, under the NI column.

Process Killing Commands

Sometimes, it's not enough to reprioritize a process. Some processes can just overwhelm a system. With some other operating systems, such situations require a reboot. Linux is different. In most cases, such you can stop such difficult processes with the **kill** and **killall** commands. In many cases, you can kill a process directly from the **top** task browser.

If there's a situation where a process is taking up a lot of memory or CPU, it's probably slowing down everything else running on that system. As shown in Figure 9-10, Firefox has loaded the CPU of the noted system pretty heavily. If it were slowing down my system, I'd press **k** from the top task browser.

FIGURE 9-10

The top task browser with heavy Firefox load

```
top - 09:45:25 up 5 days, 10:00,  7 users,  load average: 0.05, 0.14, 0.10
Tasks: 262 total,   2 running, 260 sleeping,   0 stopped,   0 zombie
Cpu(s): 13.1%us,  3.4%sy,  0.0%ni, 83.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7985904k total,  6257932k used,  1727972k free,   134668k buffers
Swap:  9775512k total,    26272k used,  9749240k free,  2319512k cached
PID to kill: 2971
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2971 michael   20   0 1426m 584m  29m S 40.2  7.5 484:48.60 firefox
 2619 root      20   0  242m  65m  29m S  8.3  0.8 174:59.88 Xorg
 2932 michael   20   0 4787m 773m  45m S  6.0  9.9  36:40.56 swriter.bin
 1231 qemu      20   0 1127m 635m 3024 S  5.0  8.2 172:23.96 qemu-kvm
 3077 michael   20   0 1847m 119m  24m S  1.7  1.5 458:21.41 plugin-containe
18809 michael   20   0 1263m  78m  15m S  1.7  1.0 145:13.97 python
 2844 michael   20   0  435m  21m  10m S  1.0  0.3   2:44.51 metacity
 1063 root      20   0  592m  16m 4504 S  0.7  0.2  27:23.08 libvirtd
17881 michael   20   0  266m 157m  21m S  0.7  2.0  33:26.73 acroread
19436 michael   20   0 15072 1316  900 S  0.7  0.0   2:56.68 top
 2874 michael   20   0  334m  15m 9996 S  0.3  0.2   5:57.49 wnck-applet
 4039 michael   20   0  649m 117m  17m S  0.3  1.5   0:16.04 opera
    1 root      20   0 19244 1272 1052 S  0.0  0.0   0:05.94 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.04 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.25 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:01.04 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
```

As shown in the figure, the **k** command reveals the PID To Kill: prompt, where I enter the PID of the Firefox process, 2971. It applies the **kill** command to the process with that PID number.

Of course, you could apply the **kill** command directly to a PID number. For example, the following command is equivalent to the steps just described in the **top** task browser:

```
# kill 2971
```

The **kill** command can be run by the owner of a process from his account. Thus, user michael could run the **kill 2971** command from his regular account, as he has administrative privileges over processes associated with his username.

The **kill** command can send a wide variety of signals to different processes. For a full list, run the **kill -l** command. Before the advent of scripts in the /etc/init.d directory, the **kill -1** command was used to send a restart signal to service daemons. For example, if the PID number of the process associated with the vsFTP server is 2059, the following command is functionally equivalent to the **/etc/init.d/vsftpd restart** command:

```
# kill -1 2059
```

Without the **-1** switch, the **kill** command, under normal circumstances, would stop the given process. In this case, it would stop the vsFTP server. But sometimes, processes get stuck in loops. In some such cases, the **kill** command does not work by itself. The process continues running. In that case, you can try two things.

First, you could try the **kill -9** command, which attempts to stop a process "uncleanly." If it is successful, other related processes may still remain in operation.

Sometimes there are a number of processes running under the same name. For example, as you'll see in Chapter 14, the Apache Web server starts several processes that run simultaneously. It's at best inefficient to kill just one process; the following command would kill all currently running server processes, assuming no other issues:

```
# killall httpd
```

Archives and Compression

Linux includes a variety of commands to archive groups of files. Some archives can be reprocessed into packages such as RPMs. Other archives are just used as backups. In either case, archives can be a terrific convenience, especially when compressed. To that end, this section explores those archive and compression commands

specifically cited in the RHCSA objectives. These "essential tools" include the **gzip**, **bzip2**, **tar**, and **star** commands.

gzip and bzip2

The **gzip** and **bzip2** commands are functionally similar, as they compress and decompress files, using different algorithms. The **gzip** command uses the Lempel-Ziv algorithm, found in some Microsoft compression algorithms. The **bzip2** command uses the Burrows-Wheeler block sorting algorithm. While they both work well, the **bzip2** command makes a big file a bit smaller. For example, either of the two following commands could be used to compress a big picture file named big.jpg:

```
# gzip big.jpg
# bzip2 big.jpg
```

It adds a .gz or a .bz2 suffix to the file, compressed to the associated algorithms. With the **-d** switch, you can use the same commands to reverse the process:

```
# gzip -d big.jpg.gz
# bzip2 -d big.jpg.bz2
```

tar

The **tar** command was originally developed for archiving data to tape drives. However, it's commonly used today for collecting a series of files, especially from a directory. For example, the following command backs up the information from the /home directory in the home.tar.gz file:

```
# tar czvf home.tar.gz /home
```

Like the **ps** command, this is one of the few commands that does not require a dash in front of the switch. This particular command creates (**c**) an archive, compresses (**z**) it, in verbose (**v**) mode, with the filename (**f**) that follows. Alternatively, you can extract (**x**) from that file with the following command:

```
# tar xzvf home.tar.gz /home
```

The compression specified (**z**) is associated with the **gzip** command; if you wanted to use **bzip2** compression, substitute the **j** switch. But there are drawbacks to the **tar** command, as such archives do not store access control list settings or SELinux attributes. But if a tar archive is all that's available, you can use commands like **restorecon**, as described in Chapter 4, to restore the contexts of an archive that have been restored to their original directories.

star

The **star** command is more appropriate for archiving files in a SELinux system. As the **star** command is not normally installed, you'll need to install it; one method is with the following command:

```
# yum install star
```

Unfortunately, the **star** command doesn't quite work in the same fashion as **tar**. If you ever have to use the **star** command, some practice is appropriate. For example, the following command would create an archive, with all SELinux contexts, from the current /home directory:

```
# star -xattr -H=exustar -c -f=home.star /home/
```

The **-xattr** switch saves the extended attributes associated with SELinux. The **-H=exustar** records the headers associated with ACLs. The **-c** creates a new archive file. The **-f** specifies the name of the archive file.

Once the archive is created, it can be unpacked with the following command, which extracts the archive:

```
# star -x -f=home.star
```

If desired, the archive can be compressed with the aforementioned **gzip** or **bzip2** commands. However, with an archive created with the **star** command, such compressed files can't be uncompressed with the **gzip -d** or **bzip2 -d** command. Nevertheless, the **star -x** command can detect and restore files from archives configured with various compression schemes. For example, based on a gzip-compressed archive, the **star** command unpacks that archive, as noted by the following log information message:

```
star: WARNING: Archive is 'gzip' compressed, trying to use the -z option.
```

Control Services Through Daemons

A *daemon* is a process that runs in the background. It is resident in system RAM and watches for signals before it goes into action. For example, a network daemon such as httpd, the Linux Web server known as Apache, waits for a request from a browser before it actually serves a Web page.

Many Linux daemons are designed to work on a network. Unfortunately, networks don't always work. Sometimes you need to restart a network daemon to implement a configuration change. As discussed in Chapter 5, RHEL makes it easy to control network service daemons through the scripts in the /etc/rc.d/init.d directory. This

directory includes scripts that can control installed Linux network services (and more) for everything from the Network File System (NFS) to sendmail. The actual daemon itself is usually located in the /sbin or /usr/sbin directory.

CERTIFICATION OBJECTIVE 9.03

Automate System Administration: cron and at

The cron system is essentially a smart alarm clock. When the alarm sounds, Linux runs the commands of your choice automatically. You can set the alarm clock to run at all sorts of regular time intervals. Many cron jobs are scheduled to run during the middle of the night, when user activity is lower. Of course, that timing can be adjusted. Alternatively, the at system allows users to run the commands of their choice, once, at a specified time in the future.

RHEL installs the cron daemon by default. There have been significant changes in cron since RHEL 5, as the cron daemon now incorporates the anacron system. The cron daemon starts jobs on a regular schedule. The anacron system helps the cron daemon work on systems that are powered off at night. This helps enterprises that want to save energy.

exam Watch

Because cron always checks for changes, you do not have to restart cron every time a change has been made.

It's configured to check the /var/spool/cron directory for jobs by user. In addition, it incorporates jobs defined in the /etc/anacrontab file, based on the 0anacron script in the /etc/cron.hourly directory. It also checks for scheduled jobs for the computer described in the /etc/crontab file and in the /etc/cron.d directory.

The System crontab and Components

The /etc/crontab file is set up in a specific format. Each line can be blank, a comment (which begins with #), a variable, or a command. Naturally, blank lines and comments are ignored. Through RHEL 5, that file included a schedule of jobs. At this point, the crontab file just includes the format for other related configuration files.

Users run regular commands. Anyone who runs a regular command, whether it be you or a daemon, is limited by various environmental variables. To see the

environmental variables for the current user, run the **env** command. If that user is your account, some of the standard variables in RHEL include **HOME**, which should match your home directory, **SHELL**, which should match the default shell, and **LOGNAME** as the username.

Other variables can be set in the /etc/crontab and related files such as /etc/anacrontab and /etc/cron.d/0hourly in the following format:

```
Variable=Value
```

Some variables are already set for you. For example, **MAIL** for me is /var/spool /mail/michael, **LANG** is en_US.UTF-8, and **PATH** is where the shell looks for commands. You can set these variables to different values in various cron configuration files. For example, the default /etc/crontab file includes the following variables:

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

Note that the values of **PATH**, **MAILTO**, and **HOME** are different from standard environment variables. The **PATH** variable in a cron configuration file may be different from the **PATH** variable associated with a shell. In fact, the two variables are independent. Therefore, you'll want to know the exact path of every command in each cron configuration file. Specify the absolute path with the command if it isn't in the crontab **PATH**.

The* MAILTO *variable can help you administer several Linux systems. The* cron *daemon sends output by e-mail. Just add a line such as* MAILTO=me@example.net *to route all* cron *messages associated with that file to that e-mail address.

The format of a line in /etc/crontab is now detailed in comments, as shown in Figure 9-11. Each of these columns is explained in more detail in Table 9-3.

If you see an asterisk in any column, the **cron** daemon runs that command for all possible values of that column. For example, an * in the minute field means that the command is run every minute during the specified hour(s). Consider another example, as shown here:

```
1  5  3  4  *  ls
```

FIGURE 9-11

The format of a crontab

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

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,f
ri,sat
# |  |  |  |  |
# *  *  *  *  *  command to be executed

"/etc/crontab" 16L, 448C
```

This line runs the **ls** command every April 3 at 5:01 A.M. The asterisk in the day of week column simply means that it does not matter what day of the week it is; crontab still runs the **ls** command at the specified time.

The entries associated with the cron daemon are flexible. For example, a 7–10 entry in the hour field would run the specified command at 7:00 A.M., 8:00 A.M., 9:00 A.M., and 10:00 A.M. A list of entries in the minute field such as 0,5,10,15,20,25,30,35,40,45,50,55 would run the specified command every five minutes. But that's a lot of numbers. The */5 in the minute field would lead to the same result. The **cron** daemon also recognizes abbreviations for months and the day of the week.

TABLE 9-3

Columns in a cron Configuration File

Field	Value
minute	0–59
hour	Based on a 24-hour clock; for example, 23 = 11 P.M.
day of month	1–31
month	1–12, or jan, feb, mar, etc.
day of week	0–7; where 0 and 7 are both Sunday; or sun, mon, tue, etc.
command	The command to be executed, sometimes listed with the username to run the command

The actual command is the sixth field. You can set up new lines with a percent (%) symbol. This is useful for formatting standard input. The example of a cron file follows formats input for an e-mail message:

```
# crontab -l
# Sample crontab file
#
# Force /bin/sh to be my shell for all of my scripts.
SHELL=/bin/bash
# Run 15 minutes past Midnight every Saturday
15 0 * * sat   $HOME/scripts/scary.script
# Do routine cleanup on the first of every Month at 4:30 AM
30 4 1 * *     /usr/scripts/removecores >> /tmp/core.tmp 2>>&1
# Mail a message at 10:45 AM every Friday
45 10 * * fri  mail -s "Project Update employees%Can I have a status
update on your project?%%Your Boss.%
# Every other hour check for alert messages
0 */2 * * * /usr/scripts/check.alerts
```

Hourly cron Jobs

Now it's time for some sample cron files. The files and scripts discussed are limited to those seen on the server1.example.com system. A number of different packages add their own cron jobs. Certain jobs associated with the cron daemon are run every hour, based on the 0hourly script in the /etc/cron.d directory. It includes the same variables as the /etc/crontab file just described. For hourly jobs, it includes one line:

```
01 * * * * root run-parts /etc/cron.hourly
```

Given the information provided in the preceding section, you should be able to read this line. The **run-parts** command executes each script in the directory that follows; the scripts in that directory are executed as the root user. Of course, the first five columns specify the time; the scripts are run at one minute past the hour, every hour, every day, every month, on each day of the week.

The script of interest in the /etc/cron.hourly directory is 0anacron. That script reviews the contents of the /var/spool/anacron/cron.daily file, to see if the **anacron** command has been run in the past day. If not, and if the system is running on AC power, the **/usr/sbin/anacron -s** command is executed, which runs scripts defined in the /etc/anacrontab configuration file.

The system status script described earlier is stored in the /etc/cron.d/sysstat file. There are two active commands in that file. The first command, **sa1**, is run every ten minutes, as depicted by the */10. That command is run every hour, every day, etc.

```
*/10 * * * * root /usr/lib64/sa/sa1 -S DISK 1 1
```

The second command, sa2, is run at 53 minutes after the hour, on the 23rd hour of each day. In other words, the system activity report is not collected until 11:53 P.M. at night.

```
53 23 * * * root /usr/lib64/sa/sa2 -A
```

Regular Anacron Jobs

The 0anacron script in the /etc/cron.hourly directory described earlier executes the **anacron** command after a system has been powered up. That command executes three scripts defined in the /etc/anacrontab file. It includes three environment variables that should seem familiar:

```
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
```

The SHELL directive may appear a bit different, but the **ls -l /bin/sh** command should confirm a soft link to the /bin/bash command, which starts the default bash shell. The following directive means that scripts are run at a random time of up to 45 minutes after the scheduled time:

```
RANDOM_DELAY=45
```

With the following directive, anacron jobs are run only between the hours of 3 A.M. and 10:59 P.M.

```
START_HOURS_RANGE=3-22
```

While the format of /etc/anacrontab is similar to those listed in a script for a regular cron job, there are differences. The order of data in each line is specified by the following comment:

```
#period in days   delay in minutes   job-identifier   command
```

The period in days is 1, 7, or @monthly, since the number of days in a month varies. The delay in minutes is associated with the **RANDOM_DELAY** directive. Since

the /etc/anacrontab file is executed through the /etc/cron.d/0hourly script, the clock starts one minute after the hour, after the system has been started. The delay in minutes comes before the **RANDOM_DELAY** directive.

In other words, based on the following line, the scripts in the /etc/cron.daily directory may be run anywhere from 5 to 50 minutes after the **anacron** command is run, or 6 to 51 minutes after the hour.

```
1   5   cron.daily      nice run-parts /etc/cron.daily
```

For more examples, review some of the scripts in the /etc/cron.daily directory. Three key scripts include *logrotate*, for rotating log files; *mlocate.cron*, which updates the *locate* file database; and *tmpwatch*, which wipes files from /tmp and /var/tmp after a specific amount of time.

The only SELinux settings associated with* cron *support automated relabeling and enable access for the* fcron *scheduler, associated with the* cron_can_relabel *and* fcron_crond *booleans.

Setting Up cron for Users

Each user can use the **crontab** command to create and manage **cron** jobs for their own accounts. There are four switches associated with the **crontab** command:

- ***-u user*** Allows the root user to edit the crontab of another specific user.
- **-l** Lists the current entries in the crontab file.
- **-r** Removes **cron** entries.
- **-e** Edits an existing **crontab** entry. By default, **crontab** uses vi.

To set up **cron** entries on your own account, start with the **crontab -e** command. Normally, it opens a file in the vi editor, where you can add appropriate variables and commands, similar to what you've seen in other cron job files.

Once the cron job is saved, you can confirm the job with either the **crontab -l** command, or by reading the contents of a file in the /var/spool/cron directory associated with a username. All current cron jobs for a user can be removed with the **crontab -r** command.

EXERCISE 9-1

Create a cron Job

In this exercise, you will modify the basic Red Hat **cron** job settings to read a text file at 1:05 P.M. every Monday in the month of January. To do so, you'll need to create a directory for yearly **cron** jobs. To do this, use the following steps:

1. Log in as a regular user.
2. Create a /etc/cron.yearly directory. Add a file called taxrem, which reads a text file from your home directory. A command such as the following in the taxrem file should suffice:

   ```
   cat ~/reminder
   ```

   Make sure to add appropriate lines to the reminder file in your home directory, such as "Don't forget to do your taxes!" Make sure the taxrem file is executable with the **chmod +x /etc/cron.yearly/taxrem** command.
3. Open up the crontab for your account with the **crontab -e** command.
4. Add an appropriate command to the crontab. Based on the conditions described, it would read as follows:

   ```
   5 13 * 1 1 root run-parts /etc/cron.yearly
   ```

5. Don't forget directives such as **SHELL=/bin/bash** at the start of the script.
6. Save and exit. Confirm the existence of the user cron file in the /var/spool/cron directory. That file should have the same name as the user.

Running a Job with the at System

Like **cron**, the **at** daemon supports job processing. However, you can set an **at** job to be run once. Jobs in the **cron** system must be set to run on a regular basis. The **at** daemon works in a way similar to the print process; jobs are spooled in the /var/spool/at directory and run at the specified time.

You can use the **at** daemon to run the command or script of your choice. For the purpose of this section, assume that user michael has created a script named 797 in his home directory to process some airplane sales database to another file in the same directory called sales.

From the command line, you can run the **at *time*** command to start a job to be run at a specified *time*. That *time* can be now; in a specified number of minutes, hours, or days; or at the time of your choice. Several examples are illustrated in Table 9-4.

You can use one of the example commands shown in Table 9-4 to open an **at** job. It opens a different command line interface, where you can specify the command of your choice. For this example, assume you're about to leave work and want to start the job in an hour. From the conditions specified, run the following commands:

```
$ at now + 1 hour
at> /home/michael/797 > /home/michael/sales
at> Ctrl-D
```

The CTRL-D command exits the **at** shell and returns to the original command line interface. The **atq** command, as shown here, checks the status of current at jobs. All jobs that are pending are listed in the output to the **atq** command:

```
$ atq
1       2012-12-21 03:00 a michael
```

If there's a problem with the job, you can remove it with the **atrm** command. For example, you can remove the noted job, labeled as job 1, with the following command:

```
$ atrm 1
```

Secure cron and at

You may not want everyone to be able to run a job in the middle of the night. If the system has a security flaw, someone may download important data or worse, and it could be days before the security breach is found.

TABLE 9-4 Examples of the at Command

Time Period	Example	Start Time for jobs
Minutes	**at now + 10 minutes**	**In 10 minutes**
Hours	**at now + 2 hours**	**In 2 hours**
Days	**at now + 1 day**	**In 24 hours**
Weeks	**at now + 1 week**	**In 7 days**
n/a	**at teatime**	**At 4:00 P.M.**
n/a	**at 3:00 12/21/12**	**On December 21, 2012, at 3:00 A.M.**

TABLE 9-5

Security Effects of cron.allow and cron.deny

	/etc/cron.deny exists	/etc/cron.deny does not exist
/etc/cron.allow exists	Only users listed in /etc/cron.allow can run crontab -e; contents of /etc/cron.deny are ignored.	Only users listed in /etc/cron.allow can run crontab -e.
/etc/cron.allow does not exist	All users listed in /etc/cron.deny cannot use crontab -e.	Only the root user can run crontab -e.

Users can be configured in /etc/cron.allow and /etc/cron.deny files. If neither of these files exist, **cron** usage is restricted to the root administrative user. If the /etc/cron.allow file exists, only users named in that file are allowed to use **cron**. If there is no /etc/cron.allow file, only users named in /etc/cron.deny can't use **cron**.

These files are formatted as one line per user; if you include the following entries in /etc/cron.deny, and the /etc/cron.allow file does not exist, users elizabeth and nancy aren't allowed to set up their own **cron** scripts:

```
elizabeth
nancy
```

However, if the /etc/cron.allow file does exist, with the same list of users, it takes precedence. In that case, both users elizabeth and nancy are allowed to set up their own cron scripts. The range of possibilities is summarized in Table 9-5.

User security for the at system is almost identical. The corresponding security configuration files are /etc/at.allow and /etc/at.deny. The range of possibilities is summarized in Table 9-6.

If you're paranoid about security, it may be appropriate to include only desired users in the /etc/cron.allow and /etc/at.allow files. Otherwise, a security breach in a service account may allow a cracker to run a cron or at script from the associated account.

TABLE 9-6

Security Effects of at.allow and at.deny

	/etc/at.deny exists	/etc/at.deny does not exist
/etc/at.allow exists	Only users listed in /etc/at.allow can run the at command; contents of /etc/at.deny are ignored.	Only users listed in /etc/at.allow can run the at command.
/etc/at.allow does not exist	All users listed in /etc/at.deny cannot run the at command.	Only the root user can run the at command.

CERTIFICATION OBJECTIVE 9.04

Local Log File Analysis

An important part of maintaining a secure system is monitoring those activities that take place on the system. If you know what usually happens, such as understanding when users log in to a system, you can use log files to spot unusual activity. Red Hat Enterprise Linux comes with new system monitoring utilities that can help identify the culprit if there is a problem.

RHEL 6 comes with an enhanced logging daemon known as rsyslog. It includes the functionality of the kernel and system logging services used through RHEL 5. The rsyslogd service logs all process activity. You can use the log files so generated to track activities on a system. The configuration of rsyslog as a log server for multiple systems is an RHCE skill covered in Chapter 17.

The rsyslog daemon is active by default and can be activated by the /etc/init.d/rsyslog script. The way it logs files is based on the configuration defined in the /etc/rsyslog.conf file. If you're familiar with the RHEL 5 syslogd and klogd daemons, the concepts in the rsyslog.conf file should be familiar.

In many cases, services such as SELinux, Apache, and Samba have their own log files, defined within their own configuration files. Details are addressed with the chapters associated with those services.

System Log Configuration File

You can configure what is logged through the /etc/rsyslog.conf configuration file. As shown in Figure 9-12, it includes a set of rules for different facilities (if the corresponding packages are installed): authpriv, cron, kern, mail, news, user, and uucp.

Each facility is associated with several different levels of logging, known as the priority. In ascending order, log priorities are debug, info, notice, warn, err, crit, alert, emerg. There's also a generic none priority that logs no messages of the specific facility; for example, a authpriv.none directive would omit all authentication messages.

For each facility and priority, log information is sent to a specific log file. For example, consider the following line from /etc/syslog.conf:

```
*.info;mail.none;news.none;authpriv.none;cron.none /var/log/messages
```

FIGURE 9-12

The rsyslog.conf log configuration file

```
#### RULES ####

# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -/var/log/maillog


# Log cron stuff
cron.*                                                  /var/log/cron

# Everybody gets emergency messages
*.emerg                                                 *

# Save news errors of level crit and higher in a special file.
uucp,news.crit                                          /var/log/spooler

# Save boot messages also to boot.log
local7.*                                                /var/log/boot.log
```

This line sends log information from all of the given facilities to the /var/log/messages file. This includes

- All facility messages of info level and higher
- Except for log messages related to the **mail**, **news**, **authpriv** (authentication), and **cron** services

You can use the asterisk as a wildcard in /etc/syslog.conf. For example, a line that starts with ***.*** tells the **rsyslogd** daemon to log everything. A line that starts with **auth.*** means you want to log all messages from the **authpriv** service.

By default, **rsyslogd** logs all messages of a given priority or higher. In other words, a **cron.err** line will include all log messages from the **cron** daemon at the **err**, **crit**, **alert**, and **emerg** levels.

Most messages from the **rsyslogd** daemon are written to files in the /var/log directory. You should scan these logs on a regular basis and look for patterns that could indicate a security breach. It's also possible to set up cron jobs to look for such patterns.

Log File Management

Logs can easily become very large and difficult to read. By default, the logrotate utility creates a new log file on a weekly basis, using the directives in the /etc/logrotate.conf file, which also pulls in directives from files in the /etc/logrotate.d directory. As shown in Figure 9-13, the directives in the file are straightforward and well explained by the comments.

Specifically, the default settings rotate log files on a weekly basis, storing the past four weeks of logs. New log files are created during the rotation, and older files have the date of rotation as a suffix. Different provisions are given to wtmp and btmp logs, related to authentication.

FIGURE 9-13

Log rotation configured in /etc/logrotate.conf

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp and btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
        minsize 1M
    rotate 1
}

/var/log/btmp {
    missingok
    monthly
    create 0600 root utmp
    rotate 1
}

# system-specific logs may be also be configured here.
```

A Variety of Log Files

Various log files and their functionality are described in Table 9-7. These files are created based on the previously described configuration of the /etc/rsyslog.conf file. All files shown are in the /var/log directory. If you haven't installed, activated, or used the noted service, the associated log file may not appear. In contrast, you may see log files not shown here based on additional installed services.

TABLE 9-7 Standard Red Hat Log Files

Log Files	Description
anaconda.*	Specifies six log files: anaconda.log for installation messages; anaconda.program.log for storage detection messages; anaconda.storage.log for format messages; anaconda.syslog for the first **dmesg**, anaconda.xlog for the first start of the GUI server; and anaconda.yum.log for package installation
audit/	Includes the audit.log file, which collects messages from the kernel 2.6 audit subsystem
boot.log	Associated with services that start and shut down process
btmp	Lists failed login attempts; readable with the **utmpdump btmp** command
ConsoleKit/	Tracks user logins with consoles and input device hardware
cron	Collects information from scripts run by the **cron** daemon
cups/	Directory of printer access, page, and error logs
dmesg	Includes basic boot messages
gdm/	Directory of messages associated with starting via the GNOME Display Manager; includes login failures
httpd/	Directory of log files associated with the Apache Web server
lastlog	Lists login records; readable with the **lastlog** command
maillog	Collects log messages related to e-mail servers
mcelog	Specifies machine check exception data on 64-bit systems
messages	Includes messages from other services as defined in /etc/syslog.conf
ntpstats/	Directory with NTP server data
pm-*	Specifies two log files related to power management
ppp/	Directory with Point to Point Protocol statistics; usually associated with telephone modems
prelink/	Directory with logs of prelinked libraries and binaries designed to speed the boot process
rpmpkgs	Current list of installed RPM packages
sa/	Directory with system activity reports
samba/	Directory of access and service logs for the Samba server

TABLE 9-7 Standard Red Hat Log Files (*continued*)

Log Files	Description
scrollkeeper.log	Notes log information related to GNOME documentation
secure	Lists login and access messages
setroubleshoot/	Directory of messages associated with the SELinux troubleshooting tool
spooler	Shows a log file that might include critical messages
squid/	Directory of files related to Squid Proxy Server access, cache, and storage
sssd/	Directory of messages associated with the System Security Services Daemon
tallylog	Supports pam_tally, which locks out a user after excessive login attempts
up2date	Includes access messages to a Red Hat Network update server
wtmp	List of logins, in binary format; can be read with the **utmpdump** command
xferlog	Adds messages associated with file transfers from a local FTP server
Xorg.0.log	Notes setup messages for the X Window System; may include configuration problems
yum.log	Specifies logs packages installed, updated, and erased with **yum**

Service Specific Logs

As suggested earlier, a number of services control their own log files. The log files for the vsFTP server, for example, are configured in the vsftpd.conf file in the /etc/vsftpd directory. As noted from that file, the following directive enables the logging of both uploads and downloads in the /var/log/xferlog file:

```
xferlog_enable=YES
```

The logging of other services may be more complex. For example, separate log files are configured for access and errors in the Apache Web server in the /var/log/httpd directory.

EXERCISE 9-2

Learn the Log Files

In this exercise, you'll inspect the log files on a local system to try to identify different problems.

1. Restart the Linux computer. Log in as the root user. Use the wrong password once.

2. Log in properly with the correct password as the root user.
3. In a console, navigate to the /var/log directory and open the file named secure. Navigate to the "FAILED LOGON" message closest to the end of the file. Close the file.
4. Review other logs in the /var/log directory. Use Table 9-7 for guidance. Look for messages associated with hardware. What log files are they in? Does that make sense?
5. Most, but not all log files are text files. Try reading the lastlog file in the /var/log directory as a text file. What happens? Try the **lastlog** command. Are you now reading the contents of the /var/log/lastlog file? Can you confirm this from the associated man page?

SCENARIO & SOLUTION

Can't connect to a local VNC server	Make sure the VNC server is running with the **/etc/init.d/vncserver restart** or **vino-preferences** command.
Can't connect to a remote VNC server	Make sure you're connecting to the right port; for example, a connection to 192.168.100.1:1 requires open ports 5900 and 5901. Review open ports with appropriate **nmap** and **telnet** commands.
Regular users can't access the crontab command or the at prompt	Review the cron.allow and cron.deny files in the /etc/ directory. If all regular users are allowed access, make sure the cron.deny file exists but is empty and to delete cron.allow. (Similar guidelines apply for the at prompt.)
Log files don't include sufficient information	Revise /etc/rsyslog.conf. Focus on the desired facility such as authpriv, mail, or cron, and revise the priority to include more detailed information

CERTIFICATION SUMMARY

RHEL 6 includes two VNC servers that can help you configure remote connections to a local GUI desktop environment. They require open ports starting with 5900 and up, depending on the number of remote GUI connections that you want to allow. Such communication can be encrypted over an SSH connection. Be aware, the viewer for the KVM graphical console also uses VNC.

A variety of system administration commands can help you as an administrator to monitor and manage the resources used on a system. These commands include **ps**, **top**, **kill**, **nice**, and **renice**. In addition, with the right commands, you can create archives. However, special commands are required to back up files with specialized attributes such as those based on ACLs and SELinux.

The cron and at daemons can help you manage what jobs are run on a system on a schedule. With related configuration files, access to these daemons can be limited to certain users. While cron configuration files follow a specific format documented in /etc/crontab, those configuration directives have been integrated with the anacron system that supports job management on systems that are powered off on a regular basis.

RHEL 6 includes the rsyslog daemon, configured primarily for local systems in the /etc/rsyslog.conf file. Log files are normally collected in the /var/log directory. The rsyslog daemon also supports the creation of a logging server that can collect log file information from a variety of systems.

TWO-MINUTE DRILL

Here are some of the key points from the certification objectives in Chapter 9.

Configure Access with VNC

- VNC communication is normally configured over ports 5900 up to 5909; to enable remote access, configured firewalls need to let such traffic through.
- On RHEL 6, VNC servers can be configured with the TightVNC server and the vino server.
- RHEL 6 also uses VNC to provide a graphical view of KVM-based virtual machines.
- Capable VNC client software includes the **vncviewer** command and the Remote Desktop Viewer that you can start with the **vinagre** command.
- VNC communication can be encrypted by routing it through a SSH connection.

Elementary System Administration Commands

- The **ps** command can identify currently running processes.
- The **top** command starts a task browser that can identify processes taking excessive load on a system.
- The **sar** and related commands provide system activity reports.
- The **iostat** command can provide CPU and storage device statistics.
- The **nice** and **renice** commands can be used to reprioritize processes.
- The **kill** and **killall** commands can be used to stop currently running processes and even daemons with a variety of signals.
- Archives can be created, extracted, and compressed with the **gzip**, **bzip2**, **tar**, and **star** commands.
- The **chkconfig** command can help control services at the daemon level.

Automate System Administration: cron and at

- The cron system allows users to schedule jobs so they run at given intervals.
- The at system allows users to configure jobs to run once at a scheduled time.
- The **crontab** command is used to work with cron files. Use **crontab -e** to edit, **crontab -l** to list, or **crontab -r** to delete cron files.
- The /etc/cron.allow and /etc/cron.deny files are used to control access to the cron job scheduler; the /etc/at.allow and /etc/at.deny files are used to control access to the at job scheduler in a similar fashion.

Local Log File Analysis

- Red Hat Enterprise Linux includes the rsyslog daemon, which monitors a system for kernel messages as well as other process activity, as configured in /etc/rsyslog.conf.
- You can use log files generated in the /var/log directory to track activities on a system.
- Other log files may be created and configured through service configuration files.
- Log files may be rotated on a regular basis, as configured in the /etc/logrotate.conf file.

SELF TEST

The following questions will help measure your understanding of the material presented in this chapter. As no multiple-choice questions appear on the Red Hat exams, no multiple-choice questions appear in this book. These questions exclusively test your understanding of the chapter. It is okay if you have another way of performing a task. Getting results, not memorizing trivia, is what counts on the Red Hat exams.

Configure Access with VNC

1. What two port numbers are associated with the first connection to a VNC server?

   _____

   _____

2. What command would you use to connect to the third VNC server window, where the VNC Server is on IP address 192.168.200.1?

   _____

3. Name a software package associated with the VNC server. Version numbers are not required, just the name of the package that can be used with the **yum** command for installation.

   _____

Elementary System Administration Commands

4. What command identifies all running processes in the current terminal console?

   _____

5. What is the highest priority number that you can set for a process with the **nice** command?

   _____

6. What command can be used to archive the files of an existing directory while saving its SELinux contexts?

   _____

Automate System Administration: cron and at

7. You want to schedule a maintenance job, maintenance.pl, to run from your home directory on the first of every month at 4:00 A.M. You've run the **crontab -e** command to open your personal

job file. Assume you've added appropriate **PATH** and **SHELL** directives. What directive would you add to run the specified job at the specified time?

_____

8. If you see the following entry in the output to the **crontab -l** command,

```
42 4 1 * * root run-parts /etc/cron.monthly
```

when is the next time Linux will run the jobs in the /etc/cron.monthly directory?

_____

9. If the users tim and stephanie are listed in both the /etc/cron.allow and the /etc/cron.deny files, and users donna and elizabeth are listed only in the /etc/cron.allow file, which of those users is allowed to run the **crontab -e** command?

_____

10. What file documents how log files are managed over time?

_____

Local Log File Analysis

11. What entry in the /etc/rsyslog.conf file would notify logged-in users whenever there is a serious problem with the kernel?

_____

12. There are several files in the /var/log directory related to what happened during the installation process. What is the first word shared by the name of these log files?

_____

LAB QUESTIONS

Several of these labs involve exercises that can seriously affect a system. You should do these exercises on test machines only. The second Lab of Chapter 1 sets up KVM for this purpose. However, some readers may not have hardware that supports KVM. Alternatives to KVM include virtual machine solutions such as VMware, available from www.vmware.com, or Virtualbox, open source edition, available from www.virtualbox.org.

Red Hat presents its exams electronically. For that reason, the labs for this chapter are available on the CD that accompanies the book, in the Chapter9/ subdirectory. It's available in .doc, .html, and .txt formats, in the filename starting with 56509-labs. In case you haven't yet set up RHEL 6 on a system, refer to the first lab of Chapter 2 for installation instructions. However, the answers for each lab follows the Self Test answers for the fill-in-the-blank questions.

SELF TEST ANSWERS

Configure Access With VNC

1. The two port numbers associated with the first connection to a VNC server are 5900 and 5901.
2. The appropriate command to connect to the third VNC server window on the given IP address is vncserver 192.168.200.1:3
3. Two software packages that install VNC servers are tigervnc and vino.

Elementary System Administration Commands

4. This is a bit of a trick question, as the **ps** command by itself identifies any currently running processes in the current console.
5. The highest priority number that can be used with the **nice** command is -20. Remember, priority numbers for processes are counter-intuitive.
6. The command that preserves SELinux contexts in an archive is **star**.

Automating System Administration: cron and at

7. The directive that runs the maintenance.pl script from a home directory at the noted time is

```
00 4 1 * * ~/maintenance.pl
```

8. Based on the noted entry in /etc/crontab, the next time Linux will run the jobs in the /etc/cron.monthly directory is on the first of the upcoming month, at 4:42 A.M.
9. When usernames exist in both the /etc/cron.allow and /etc/cron.deny files, users listed in /etc/cron.deny are ignored. Thus, all four users listed are allowed to run various **crontab** commands.
10. The file associated with the management of log files over time is /etc/logrotate.conf.

Local Log File Analysis

11. There's a commented entry in the /etc/rsyslog.conf file that meets the requirements of the question. Just activate it to notify you (and everyone) whenever a serious problem with the kernel occurs:

```
kern.*      /dev/console
```

Of course, that means there are other acceptable ways to meet the requirements of the question.

12. The log files in /var/log that are most relevant to the installation process start with **anaconda**.

LAB ANSWERS

Lab 1

One way to modify the login messages as noted is with the following steps (I can think of at least one other method, related to the /etc/cron.d directory):

1. Log in as the root user.
2. Run the **crontab -e** command.
3. Add appropriate environment variables, at least the following:

```
SHELL=/bin/bash
```

4. Add the following commands to the file to overwrite /etc/motd at the appropriate times:

```
0 7  * * * /bin/echo 'Coffee time!' > /etc/motd
0 13 * * * /bin/echo 'Want some ice cream?' > /etc/motd
0 18 * * * /bin/echo 'Shouldn't you be doing something else?' > /etc/motd
```

5. Save the file. As long as the **cron** daemon is active (which it is by default), the next user who logs into the console after one of the specified times should see the message upon a successful login. If you want to test the result immediately, the **date** command can help. For example, the following command

```
# date 06120659
```

sets a date of June 12, at 6:59 A.M., just before the **cron** daemon should execute the first command in the list. (Of course, you'll want to substitute today's date, and wait one minute before logging in to this system from another console.)

Lab 2

To set up an at job to start 24 hours from now, start with the **at** command. It'll take you to an at> prompt.

Currently installed RPMs are shown in the output to the **rpm -qa** command. Since there is no PATH given at the at> prompt, you should include the full path. So one way to create a list of cur-

rently installed RPMs in the /root/rpms.txt file, in a one-time job starting five minutes from now, is with the following commands:

```
# at now + 5 min
at> /bin/rpm -qa > /root/rpms.txt
at> Ctrl+d
#
```

Within five minutes, you should see an rpms.txt file in the home directory of the root user, /root. If five minutes is too long to wait (as it might be during the RHCSA exam), proceed to Lab 3 and come back to this problem afterward. Don't forget to set up the other at job to be run in 24 hours.

Lab 3

Given the hardware discussed so far in this book, successful configuration of a remote VNC server can only be confirmed indirectly. However, if you have a second physical system with RHEL installed, it's not hard to confirm the availability of a VNC server remotely. For example, if the VNC server can be found on the 192.168.122.200 system, on the second terminal, you can connect to that VNC server with the **vncviewer 192.168.122.200:2** command.

Lab 4

There are no secret solutions in this lab; the intent is to get you to review the contents of key log files to see what should be there.

When you review the anaconda.* files in /var/log and compare them to other files, you may gain some insight on how to diagnose installation problems. In future chapters, you'll examine some of the log files associated with specific services; many are located in subdirectories such as /var/log/samba/ and /var/log/httpd/.

The failed login should be readily apparent in the /var/log/secure file. You may be able to get hints in the output to the **utmpdump btmp** command.

When you review the /var/log/cron file, you'll see when standard **cron** jobs were run. Most of the file should be filled (by default) by the standard hourly job, **run-parts /etc/cron.hourly**, from the /etc/crontab configuration file. If you've rebooted, you may see the anacron service, and you should be able to search for the job of the same name.

While /var/log/dmesg includes the currently booted kernel, it may be the same kernel as the one associated with /var/log/anaconda.syslog, if you haven't upgraded kernels. At the end of /var/log/dmesg, you can find the filesystems mounted to the EXT4 format, as well as currently mounted swap partitions. For example, the following lists partitions from a KVM-based virtual drive:

```
EXT4-FS (vda1): mounted filesystem with ordered data mode.
SELinux: initialized (dev vda1, type ext4), uses xattr
```

```
EXT4-FS (vda5): mounted filesystem with ordered data mode.
SELinux: initialized (dev vda5, type ext4), uses xattr
Adding 1023992k swap on /dev/vda3. Priority:-1 extents:1 across:979956k
```

As you've hopefully discovered, the /var/log/maillog file does not include any information on mail clients, but only servers.

Red Hat included a GUI configuration tool in RHEL 5. The automatic configuration for hardware graphics is now sufficiently reliable; there's no longer even a standard xorg.conf configuration file.