12

RHCE Administrative Tasks

CERTIFICATION OBJECTIVES

The automation of system maintenance is an objective for both the RHCSA and RHCE exams. For the RHCE, you need to know how to create a shell script for that purpose. You'll study some example scripts used on RHEL 6. Standard scripts may be used on an hourly, daily, or even weekly basis. The Linux kernel is flexible and highly customizable. With different run-time parameters configured in the /proc directory, kernels can be modified to meet the needs of your users.

One of the special challenges of the RHCE exam is the configuration of a custom RPM package. It's easier than you think. RHEL 6 already includes excellent tools to help you create that custom RPM.

The RHCE objectives also include a number of special network options. You need to know how to set up a system as a Kerberos client. In a complex network, you may need to configure a special route between networks. Finally, with the variety of systems that can be configured, you should know how to connect to remote storage over a network; the RHCE specifies the Internet Small Computer Systems Interface (iSCSI) for that purpose.

INSIDE THE EXAM

This chapter directly addresses six RHCE objectives. The first is a basic skill for systems administration, specifically, to

- Use scripting to automate system maintenance tasks

While an earlier version of the objectives specified the use of the bash shell, some Linux users work with different shells. No matter. Administrative scripts combine a series of commands in a single executable file. Automated scripts are normally run on a regular schedule, which makes the cron daemon perfect for that purpose.

Some Linux administrative tasks can be met through kernel run-time parameters, how the kernel manages the operating system. That's made possible by the files in the /proc/sys directory and the **sysctl** command and is summarized by the following objective:

- Use /proc/sys and sysctl to modify and set kernel run-time parameters

As RHEL and other Linux distributions are configured with RPM packages, it can

be helpful to know how to create your own RPM. The following RHCE objective is step one in that process, to

- Build a simple rpm that packages a single file

This chapter also addresses several network tasks from the RHCE objectives. The configuration of an iSCSI initiator, as described in the following objective, is the configuration of a client:

- Configure system as an iSCSI Initiator persistently mounting existing Target

The following objective relates to the configuration of a system as a Kerberos client:

- Configure system to authenticate using Kerberos

Finally, the following objective is related to the nitty-gritty of enterprise networking:

- Route IP traffic and create static routes

CERTIFICATION OBJECTIVE 12.01

Automate System Maintenance

As discussed in Chapter 9, RHEL 6 includes a number of standard scripts for system maintenance, driven by the /etc/crontab and the /etc/anacrontab configuration files. In that chapter, you can review basic information on when scripts in various /etc/cron.* directories are run. In this chapter, you'll get into the details of some of the standard scripts in the /etc/cron.* directories and then analyze various script commands in detail. With that information in hand, you'll be able to create administrative scripts too.

Standard Administrative Scripts

Start with the scripts in the /etc/cron.hourly directory. While the intent of the 0anacron script was summarized in Chapter 9, you'll analyze it in detail here. But first, start with a simpler script, mcelog.cron, which logs machine check exception data on an hourly basis. That script has two lines. The first line is standard on most scripts, as it specifies that the commands that follow are governed by the syntax of the bash shell:

```
#!/bin/bash
```

Normally, lines that start with the pound symbol (#) are comments. This line is an exception. The second line is a straightforward execution of the **mcelog** command, which would exit silently if a device is not found (**--ignorenodev**) and filters out (**--filter**) known problems, before appending the messages (**>>**) to the end of the /var/log/mcelog file.

```
/usr/sbin/mcelog --ignorenodev --filter >> /var/log/mcelog
```

Next, examine the contents of the /etc/cron.daily directory. A slightly more complex script is logrotate. It starts with what seems to be a different shell operator:

There's actually a bug related to the 0anacron script; for more information, search for bug 675077 at https://bugzilla.redhat.com.

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

But that command is soft-linked to the /bin/bash command, the bash shell. The next line in the file is executed automatically. The **logrotate** command rotates logs as defined in the /etc/logrotate.conf file, as described in Chapter 9. Standard logging messages are sent to /dev/null, which is essentially the Linux trash bin. Error messages, as signified by the **2>**, are sent to the standard exit value, as indicated by the **&1**.

Some Linux distributions (not Red Hat) link the /bin/sh command to a shell other than bash. Unless #!/bin/bash is specified in the script, it may not be transferable to other distributions.

```
/usr/sbin/logrotate /etc/logrotate.conf >/dev/null 2>&1
```

The following line assigns the standard exit value to a variable named **EXITVALUE**:

```
EXITVALUE=$?
```

Success has an exit value of 0. If there's a problem, the exit value is some other number. The **if** command starts a conditional statement. The bang character (!), which looks like an exclamation point, in effect means "not" or "anything but." So the following if conditional is true when the value of **EXITVALUE** is something other than 0:

```
if [ $EXITVALUE != 0 ];
```

So if **EXITVALUE** is not 0, the command inside the **if** conditional is executed, which can help an administrator identify a problem with the logrotate script or related log files.

```
/usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
```

The **fi** command that follows ends the conditional statement that started with the **if**. The last directive returns 0, an indication of success:

```
exit 0
```

Since that may have been confusing for users who are newer to scripts, it's a good time to look at some of the basic commands available for scripts.

Script Commands

Scripts are filled with various command constructs. Some groups of commands are organized in a loop, which continues to run as long as the conditions are met. These command constructs are also known as *operators*. Common operators include **for**, **if**, and **test**. The end of a loop may be labeled with an operator such as **done** or **fi**. Some operators only exist in the context of others, which will be described in the subsections that follow.

Test Operators with if

The **if** operator is primarily used to check if a file is of a certain type. For example, the following command checks to see if the /var/cache/man/whatis file, a local database of man pages, is anything but a regular file:

```
if [ ! -f /var/cache/man/whatis ]
```

As suggested earlier, the bang (!) is "anything but." The **-f** checks to see if the filename that follows is a currently existing regular file. The key is the test operators common in bash shell scripts. Some of these operators are listed in Table 12-1.

The **if** operator normally is associated with a **then**, and possibly an **else** operator. For example, take the following hypothetical loop:

```
if [ -e /etc/inittab ];
then
    /bin/ls /home > /root/homedirs
else
    /bin/echo "Don't reboot, /etc/inittab is missing!"
fi
```

For this loop, if the /etc/inittab file exists (courtesy of the **-e**), the command associated with the **then** operator is run. If that file is missing, then the noted message is run.

TABLE 12-1

Test Operators for bash Scripts

Operator	Description
-b	Checks for a block file.
-d	Looks to see if the file is a directory.
-e	Asks if the file exists.
-eq	Checks for equality of the noted variables or values.
-f	Works if the file is a regular file.
-ge	Looks to see if the first value is greater than or equal to the second.
-le	Looks to see if the first value is less than or equal to the second.
-lt	Looks to see if the first value is less than the second.
-ne	Looks to see if the first value is not equal to the second.
-r	Checks the file for read permissions.
-s	Checks to see if the size of the file is greater than zero.
-w	Inspects the file for write permissions.
-x	Looks to the file for execute permissions.
||	Asks if the previous expression is false.
&&	Asks if the previous expression is true.

Test Operators withtest

The **test** operator is sometimes used as a conditional within the **if**. For example, the original version of the 0anacron script in the /etc/cron.hourly directory includes the following line:

```
if test -x /var/spool/anacron/cron.daily;
```

which is functionally equivalent to

```
if [ -x /var/spool/anacron/cron.daily ];
```

The doLoop

The **do** loop normally exists within other loops. It's fairly simple; the following example continues until some condition related to variable **n** is met:

```
do
     echo "I love Linux #$n"
done
```

A more complex example exists within the tmpwatch script, in the /etc/cron.daily directory, as it is combined with an **if** operator:

```
do
    if [ -d "$d" ]; then
        /usr/sbin/tmpwatch "$flags" -f 30d "$d"
    fi
done
```

This loop executes the noted **tmpwatch** command for all noted files from variable **d** that are confirmed as directories **[-d "$d"]**.

The fordirective with a doLoop

An example of a **for** directive exists in the tmpwatch script, in the /etc/cron.daily directory. It includes the **do** loop just described. The **for** directive specifies the value of variable **d** based on existing directories such as /var/cache/man/cat1.

```
for d in /var/{cache/man,catman}/{cat?,X11R6/cat?,local/cat?}; do
    if [ -d "$d" ]; then
        /usr/sbin/tmpwatch "$flags" -f 30d "$d"
    fi
done
```

That combination may seem complex. I've written a simpler script for Exercise 12-1.

Create Your Own Administrative Scripts

If this is the first time you're creating a script, keep it simple. There may be a command that you run often on a Linux system. For example, sometimes I configure my server to collect pictures from an outside camera every second. After a few days, that results in a lot of files. The **rm** command by itself can't handle too many files. I can never remember the exact command, so I've set up a script for this purpose:

```
#!/bin/sh
/usr/bin/find /home/camera/ -type f -name "outside*" -exec rm -f {} \
```

This particular script finds all regular files (**-type f**) with the given name and passes it off to the **rm** command. Commands of any such complexity are perfect candidates for scripts. I could set up that script to be run on a regular basis with a cron job associated with the user named camera, assuming the files in that directory are owned by that user. Such cron jobs were discussed in Chapter 9.

EXERCISE 12-3

Create a Script

In this lab, you'll create a simple script that lists the .doc files in the local home directory. Of course, there are easier ways to identify local .doc files. If there are no .doc files in an appropriate directory, you can use the **touch** command to create them, or substitute a different kind of file, such as those with a .pdf or a .conf extension. The purpose of this exercise is to help users who are newer to scripts understand how they work.

1. Use the /etc/cron.daily/cups script as a template. Copy it to your home directory with a command like **cp /etc/cron.daily/cups ~/testscript**.
2. Take ownership of the script from your regular user account with the **chown** command. For more information on **chown**, see Chapter 4. Confirm appropriate permissions in the newly copied file with the **ls -l ~/testscript** command. Scripts should be executable.
3. Open the script. Consider the first line. It's acceptable to leave it as is or change it to **#!/bin/bash**.
4. Consider the second line. It reads the file names in the /var/spool/cups/tmp directory. How would you change it to read .doc files in your local home directory? For user michael, one option is the following:

   ```
   for d in /home/michael/*.doc
   ```

   In some cases, a different directory such as /home/michael/Documents might be more appropriate.
5. Consider the **if** loop that follows. The **-d** operator checks to see if the file is a directory. An option like **-f** is more appropriate, as it checks to see if the contents of the variable is a regular file. While other options may work, change the noted line to read

   ```
   if [ -f "$d" ]; then
   ```

6. As the objective is simply to identify .doc files in the local directory, it's best to send the output to another file. One method is with the following line, where the >\> appends the output to the end of a file.

   ```
   /bin/ls -l "$d" >\> docfiles
   ```

7. You may retain the remaining lines; where the **fi** ends the **if** loop, the **done** ends the **do** loop, and the **exit 0** returns a success message:

```
    fi
done
exit0
```

8. Save the changes. Execute the script from the local directory. Since the script name is testscript, the following command executes its contents from the local directory:

```
$ ./testscript
```

9. Check the contents of the docfiles file. If the script worked, you'll find existing .doc files in that directory.

CERTIFICATION OBJECTIVE 12.02

Kernel Run-Time Parameters

Kernel run-time parameters, as defined in the RHCE objectives, relate to files in the /proc/sys directory and the **sysctl** command. Closely related is the /etc/sysctl.conf configuration file, as that's used by the **sysctl** command during the boot process to add parameters to various files in the /proc/sys directory. So it's appropriate to start this section with a look at that sysctl.conf file.

How sysctl Works with /etc/sysctl.conf

The /etc/sysctl.conf file was briefly discussed in Chapter 1, as it controls IPv4 forwarding. To review, you can enable IPv4 forwarding in two steps. First, change the following boolean directive to activate IPv4 forwarding in the configuration:

```
net.ipv4.ip_forward = 1
```

Then make the system re-read the configuration file with the following command:

```
# sysctl -p
```

Let's examine this process in a bit more detail. First, kernel run-time parameters are documented in various files in the /proc/sys directory. Add the net.ipv4.ip_forward variable to that directory. That's interpreted as the ip_forward file, in the net/ipv4/ subdirectory. In other words, IPv4 forwarding is documented in the ip_forward file, in the /proc/sys/net/ipv4 directory.

As that file contains either a 0 or a 1, it is a boolean variable. So the value 1 for the net.ipv4.ip_forward variable activates IPv4 forwarding.

What if you want to add IPv6 forwarding? While that's not configured in the /etc/sysctl.conf file, it's a feature that you can add. IPv6 forwarding can be set in a file named forwarding, in the /proc/sys/net/ipv6/conf/all directory. In other words, to set IPv6 forwarding on reboot, you'd include the following directive in /etc/sysctl.conf:

```
net.ipv6.conf.all.forwarding=1
```

Similar directives would work for other settings associated with files in the /proc/sys directory. Look at the icmp_* directives in the /proc/sys/net/ipv4 directory. Some of you may recognize that the Internet Control Message Protocol (ICMP) is most frequently associated with the **ping** command. In fact, a **ping** command is a request for an echo. So the icmp_echo_ignore_all and icmp_echo_ignore_broadcasts relate to a direct **ping** command, as well as a **ping** command associated with the broadcast address.

In other words, if you add the following directives to the /etc/sysctl.conf file:

```
net.ipv4.icmp_echo_ignore_all = 1
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

the local system won't respond to a direct **ping** command, nor will it respond to a request made by a **ping** to the broadcast address for the network.

Settings in the /etc/sysctl.conf File

The settings in the /etc/sysctl.conf file are a small fraction of what can be configured. It's fair to assume that RHEL 6 includes the options in that file for a reason, and those settings are most likely to be addressed in a RHCE exam. You've already examined the first directive for IPv4 forwarding. The next directive, if active, makes sure that packets that come in from an external network are in fact external:

```
net.ipv4.conf.default.rp_filter = 1
```

The following directive is normally disabled, to keep crackers on outside networks from routing data via third parties. Such routing is a common tactic in attacks, as it can help mask the identity of the attacker.

```
net.ipv4.conf.default.accept_source_route = 0
```

Also known as the kernel magic sysrq key, developers may enable this directive for development purposes. Generally, you should retain the following setting:

```
kernel.sysrq = 0
```

If there's a crash of the Linux kernel, this option includes the PID number with the kernel core dump file to help identify the culprit:

```
kernel.core_uses_pid = 1
```

Another standard method used by crackers to overload a system is a flood of SYN packets. It's similar to the so-called "ping of death." The following setting regulates their use:

```
net.ipv4.tcp_syncookies = 1
```

A bridge is an older term for a switch that can regulate traffic within a single network. The following directives disable the use of the noted **iptables**, **ip6tables**, and **arptables** commands on such bridges.

```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

Such bridges relate to virtualization on physical host systems; they don't apply within KVM-based virtual machines.

EXERCISE 12-2

Disable Responses to the ping Command

In this exercise, you'll use kernel parameters to disable responses to the **ping** command. While this exercise can be run on any two connected systems, this exercise assumes that you'll be configuring the server1.example.com system and testing the result from the tester1.example.com system.

1. On the server1.example.com system, review the current setting related to responses to **ping** messages with the following command:

   ```
   # cat /proc/sys/net/ipv4/icmp_echo_ignore_all
   ```

2. Assuming the output is a 0, try the **ping localhost** command. What happens? Don't forget to press CTRL-C to exit from the output stream. If the output is 1, skip to Step 5.
3. Confirm the result from a remote system such as tester1.example.com. In some situations, you may not have physical access to that system, so connect with the appropriate **ssh** command. From the remote system, try the **ping server1.example.com** or **ping 192.168.122.50** commands.
4. Return to the server1.example.com system. Change the kernel setting described in Step 1 with the following command:

   ```
   # echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all
   ```

   Confirm by repeating the command from Step 1. Try the **ping localhost** command again. What happens?
5. Restore the original 0 setting to the icmp_echo_ignore_all option.

CERTIFICATION OBJECTIVE 12.03

Create an RPM Package

If you've never created an RPM before, the RHCE objective to "build a simple RPM that packages a single file" may seem like a big challenge. With the help of some relatively new tools, it's easier than it looks.

While you won't have access to the Internet during the exam, you can download source RPMs to help study for the exam. Source RPMs, when installed, set up a structure of subdirectories. The location depends on the user. If the source RPM is installed by user michael (and yes, regular users can now load source code), source RPM subdirectories can be found in the /home/michael/rpmbuild directory.

In production, it's risky to install and compile a source RPM as the root administrative user. Mistakes in the source RPM could easily compromise a production system.

The source code unpacked from a source RPM (SRPM) can serve as a model for the one that you will create. RHEL 6 tools are now available to help set up a spec file. That spec file can then be used to build an actual binary RPM that others can use to install the single file on their own systems.

Source RPMs

On Red Hat systems, when the GNU General Public License (GPL) refers to source code, it refers to source RPMs. In fact, Red Hat complies with such open-source licenses in part by releasing source RPMs on their public servers at ftp.redhat.com. The source code includes the programs and files, as created by the developers. Source code can then be built into the binary packages used for installing RHEL 6 and additional software after installation is complete.

In this section, you'll download and install the source-code RPM associated with the vsFTP server. One way to do so is with the **lftp** command described in Chapter 2. To use that command to connect to the Red Hat FTP server, run the following command:

```
$ lftp ftp.redhat.com
```

That command connects to the Red Hat public FTP server anonymously and initiates the lftp ftp.redhat.com> prompt. To connect to the directory with source-code packages for the RHEL 6 server, run the following command from the noted prompt:

```
lftp ftp.redhat.com> cd /redhat/linux/enterprise/6Server/en/os/SRPMS
```

Command completion features work at the **lftp** prompt; in other words, if you're not sure what directories are available after typing in the **cd** command, just press TAB once or twice.

At the noted directory, you can then download the source code for the vsFTP server with the following command (the version number will vary with updates):

```
get vsftpd-2.2.2-6.el6.src.rpm
```

You can then run the **quit** command to exit from the **lftp** prompt and return to the regular command line interface. The source-code SRPM should now be available

in the local directory. Even as a regular user, you'll be able to unpack that SRPM with the **rpm** command. For example, the following command run from my regular account unpacks that source code into /home/michael/rpmbuild subdirectories:

```
$ rpm -ivh vsftpd-*.src.rpm
```

The command doesn't install the vsFTP server, as it would with a regular RPM package. It unpacks the SRPM into specific subdirectories.

The Directory Structure of an RPM Source

SRPMs, when installed, are unpacked. The source code from such packages are loaded onto the rpmbuild/SPECS and rpmbuild/SOURCES subdirectories. As suggested by the names, the source code is in the SOURCES subdirectory. Figure 12-1 illustrates the list of files unpacked from the vsFTP source code.

Most of the files in the noted directory are patches, updates to the source code. Some of the files are configuration files to be loaded into the /etc/vsftpd directory by the binary package. The actual source code is compressed in a gzip-compressed tar archive, in .tar.gz format. Note how the version number is included in the source-

FIGURE 12-1

Source code in an rpmbuild/SOURCES subdirectory

```
[michael@server1 SOURCES]$ ls
vsftpd-2.0.5-greedy.patch          vsftpd-2.2.2-clone.patch
vsftpd-2.1.0-build_ssl.patch       vsftpd-2.2.2-isolate.patch
vsftpd-2.1.0-configuration.patch   vsftpd-2.2.2.tar.gz
vsftpd-2.1.0-filter.patch          vsftpd-2.2.2-v6only.patch
vsftpd-2.1.0-libs.patch            vsftpd-close-std-fds.patch
vsftpd-2.1.0-pam_hostname.patch    vsftpd_conf_migrate.sh
vsftpd-2.1.0-tcp_wrappers.patch    vsftpd.ftpusers
vsftpd-2.1.0-trim.patch            vsftpd.init
vsftpd-2.1.0-userlist_log.patch    vsftpd.pam
vsftpd-2.1.1-daemonize_plus.patch  vsftpd.user_list
vsftpd-2.2.0-openssl.patch         vsftpd.xinetd
vsftpd-2.2.0-wildchar.patch
[michael@server1 SOURCES]$
```

code archive. Such archives can be created and unpacked with the help of the **tar** command.

on the job

Source code can also be compressed using the bzip2 algorithm.

To create an RPM, you're going to have to set up a similar archive in the same rpmbuild/SOURCES subdirectory. Once configured, you'll also have to create a . spec file, to be stored in the rpmbuild/SPECS subdirectory. You'll then be able to compile that package with the **rpmbuild** command.

Install and Analyze the rpmbuild Command

The **rpmbuild** command is used to build a binary RPM from source code, as configured with a .spec file. As the command is not available by default, you'll have to install it from the rpm-build package. You should also install the rpmdevtools package, which will be useful shortly. To speed up the process, it's more efficient to use administrative privileges to use the **yum** command to install all of these packages, using the procedures described in Chapter 7.

```
# yum install rpm-build rpmdevtools
```

In general, if you want more information on how it works, run the **rpmbuild** command with the **-v** or **-vv** switch. Most documentation assumes the **rpmbuild** command uses a spec file in the rpmbuild/SPECS directory. In that case, you'd use the **-b** switch. Options with the **-t** switch are associated with a spec file included in the actual source-code archive in .tar or compressed .tar.gz format. Given current documentation, it's assumed that you'll configure a separate .spec file and would therefore use one of the **-b** switches described in Table 12-2.

TABLE 12-2 Switches for the rpmbuild Command

Switch	Description
-ba	Builds both binary and source RPM packages.
-bb	Builds the binary RPM package.
-bc	Executes the %prep and %build commands from the .spec file.
-bi	Executes the %prep, %build, and %install commands from the .spec file.
-bl	Checks for the existence of cited files.
-bp	Executes just the %prep stage of the .spec file.
-bs	Builds just the source RPM package.

In general, you'd apply either the rpmbulid -ba or rpmbuild -bb command to the .spec file from the rpmbuild/SPECS subdirectory.

Use the rpmbuild Command

If you unpacked the vsFTP source RPM described earlier, there will be a .spec file in the rpmbuild/SPECS subdirectory. The following rpmbuild command can be used to build both the source and binary RPMs for the vsFTP server, from a regular user's home directory:

```
$ rpmbulid -ba rpmbuild/SPECS/vsftpd.spec
```

In many cases, the process of building an RPM from source code requires the installation of development packages. For example, when I ran the noted **rpmbuild** command on the standard server1.example.com system, it cited a need for four other packages:

```
pam-devel is needed by vsftpd-2.2.2-6.el6.x86_64
libcap-devel is needed by vsftpd-2.2.2-6.el6.x86_64
openssl-devel is needed by vsftpd-2.2.2-6.el6.x86_64
tcp_wrappers-devel is needed by vsftpd-2.2.2-6.el6.x86_64
```

The GNU C Compiler package, gcc, is also required. With dependencies, several other packages will probably be required on your system. So before the noted **rpmbuild** command works, you should install the noted packages with a command like

```
# yum install gcc pam-devel libcap-devel openssl-devel tcp_wrappers-devel
```

Once these developmental and related compiler packages are installed, a regular user will be able to run the aforementioned **rpmbuild** command to compile the source code for the vsFTP server to create a binary and a source RPM.

Once the build process is complete, the compiled binary RPM can be found in the rpmbuild/RPMS/x86_64 subdirectory; of course, the last bit would vary with the architecture. The collected source RPM can be found in the rpmbuild/SRPMS subdirectory

Create Custom Source Code

With what you now know about the build process for the vsFTP source RPM, you should be able to create a source-code package. For the purpose of this chapter, I've created a corporate_policies.pdf file. Since source code that's built into an RPM is configured as a gzip compressed tar archive that will be expanded into a directory, it

should be copied to a directory created for that purpose. In addition, the **rpmbuild** command, with standard .spec files, requires access to an executable file named configure. It can be an empty file.

For my own user account, I took the following steps:

1. I copied the given corporate_policies.pdf file to a newly created /home/michael/CorPor-1.0 directory. The -1.0 represents the version number of the package.
2. I created an empty file named *configure* and assigned it executable permissions with the following commands:

```
$ cd /home/michael/CorPor-1.0
$ touch configure
$ chmod u+x configure
$ cd /home/michael
```

3. I used the following command to create an appropriate gzip-compressed tar archive:

```
$ tar czvf CorPor-1.0.tar.gz CorPor-1.0
```

4. I then copied the archive to the rpmbuild/SOURCES subdirectory.

If you're following along in a regular account, download a PDF file, such as RHEL 6 documentation available from http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/index.html. Copy the downloaded PDF, and change that file name with a command such a mv to corporate_policies.pdf.

That rpmbuild/SOURCES subdirectory can also contain the source code for the previously discussed vsFTP server. Don't be concerned about all of the patch and other files, as the RHCE objective is to create an RPM with one file. With a properly configured spec file, the source code for multiple packages can co-exist in this directory.

One More Prep Package

The next step is to create a spec file. If you've looked at the vsftpd.spec file in the rpmbuild/SPECS subdirectory, don't be too concerned. First, your spec file doesn't have to be nearly so complex. Second, there's no need to memorize the format of a .spec file, as long as the rpmdevtools package is installed. It depends on the fakeroot package, which supports root-type privileges.

Once installed, two commands from the rpmdevtools package are of special interest. The **rpmdev-setuptree** command creates the rpmbuild/ subdirectory, with appropriate subdirectories. The **rpmdev-newspec** command creates a newpackage.spec file in the local directory, with a template that can be used to create an RPM.

Thus, if the RHCE exam expects you to build an RPM, you'd run the **rpmdev-setuptree** command to build the directory structure, and then copy the gzip compressed archive to the rpmbuild/SOURCES subdirectory. Now you can create a spec file to process that archive.

Create Your Own Spec File

You can create a newpackage.spec file with the **rpmdev-newspec** command. From that template, you can configure a spec file that can process the CorPor.tar.gz archive into an RPM package. The file as shown in Figure 12-2 essentially prompts for all needed information.

Compare this file to the contents of the vsftpd.spec file described earlier for hints. For this section, make a copy of that file as corpor.spec in the rpmbuild/SPECS subdirectory. Only the .spec file extension matters; it could be named mytest.spec. Examine the .spec file line by line.

The first line is the name of the spec file. While the given Name doesn't matter, it's becomes the name for the RPM package:

```
Name: CorPor
```

The version number is also added to the RPM; 1.0 is standard:

```
Version: 1.0
```

The release number is added afterward; this default adds a **1.el6** for RHEL 6:

```
Release: 1%(?dist)
```

While the summary should describe the contents of the package, it doesn't affect how the RPM is created.

```
Summary: A package with one file
```

The Group is related to the package groups listed in the XML file described in Chapter 7. To set up a package as part of a package group, you'd need to assign a real package group; for example, the vsFTP server is part of the System Environment /Daemons package group. But for our purposes, the assigned package group does not matter.

```
Group:  Miscellaneous
```

FIGURE 12-2

A newpackage.spec file

```
Name:
Version:
Release:        1%{?dist}
Summary:

Group:
License:
URL:
Source0:
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -
n)

BuildRequires:
Requires:

%description


%prep
%setup -q


%build
%configure
make %{?_smp_mflags}


%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT


%clean
rm -rf $RPM_BUILD_ROOT


%files
%defattr(-,root,root,-)
%doc
"newpackage.spec" 42L, 488C
```

Open-source software may be released under a variety of different licenses. Most RHEL 6 software is released under the GPL.

```
License: McGraw-Hill
```

You can either comment out the URL, or add an appropriate address, as shown here:

```
URL: http://www.mheducation.com/
```

The Source0 is perhaps most important. It specifies the name of the gzip compressed tar archive file, either from a remote URL, or from the local rpmbuild/SOURCES

subdirectory. While the vsftpd.spec file includes Source1, Source2, and later directives, it's not required. All that's needed is the full name of the compressed archive file.

```
Source0: CorPor-1.0.tar.gz
```

Most packages don't include a BuildArch directive, as most packages are built to the architecture of the local system. However, for those packages that are supposed to be architecture-independent, it's appropriate to include the following BuildArch line:

```
BuildArch: noarch
```

If you forget, don't worry. The BuildArch line is not required unless you're told to set up a single RPM package that can be installed on systems with all architectures.

The lines that follow are dependencies. The BuildRequires line specifies packages that must be installed before the RPM and SRPM packages can be built with this spec file. Review the vsftpd.spec file for this line. There are actually four BuildRequires lines in that file, which specify the developmental packages listed as dependencies when you applied the **rpmbuild -ba** command to that vsftpd.spec file. In contrast, the Requires line would specify actual dependencies. For an RPM package from a single file, especially one related to documentation, there's generally no reason to have BuildRequires or Requires dependencies. In that case, you'd comment out these lines:

```
#BuildRequires
#Requires
```

The %description directive that follows allows you to include a brief paragraph description of the package. One word would be sufficient.

```
%description
This is a package with one file, associated with corporate policies.
```

The %prep section is associated with any commands required to prepare the source code. the line that follows is a bit strange. Even though it starts with a %, it is a command. The **%setup -q** is a command macro that unpacks the compressed tar archive in the rpmbuild/SOURCES directory.

```
%prep
%setup -q
```

The %build section that follows includes commands that configure and compile the source code. It normally points to scripts within the source code, commonly set up with script filenames like **configure** and **make**. With a one-file RPM, there's nothing to configure or compile, so it is appropriate to comment out or erase these options.

```
%build
#%configure
#make %{?_smp_mflags}
```

The %install section includes commands that actually adds the files from the package to noted directories. It normally starts with the following command to delete the directory tree associated with any previous builds of this package:

```
%install
rm -rf $RPM_BUILD_ROOT
```

As noted before, the **make** command is normally used to compile actual source code. Unless you've included some script, command, or executable in the source code, it's also not necessary. For RPM packages with one file, you should in fact erase or comment out that line:

```
#make install DESTDIR=$RPM_BUILD_ROOT
```

However, that means you need to configure a directory where the specified package is actually getting installed. I actually found an excellent model for this purpose in a package for fonts for a language from Ethiopia, in the abyssinica-fonts.spec file. The key excerpt is shown in Figure 12-3.

The directives from Figure 12-3 also provide a slightly different perspective. For example, %{buildroot} is equivalent to $RPM_BUILD_ROOT. As there's nothing to compile in a group of fonts, there's no "make" command in that file.

I've modified those options a bit. The install -d directive creates a directory, with (-m) mode (chmod-style octal permissions) of 0755. The system is built on some hypothetical root to be defined, in the /opt/CorPor-1.0 subdirectory.

FIGURE 12-3 An RPM model for file placement

```
%install
rm -rf %{buildroot}

#fonts
install -d -m 0755 %{buildroot}%{_fontdir}
install -m 0644 *.ttf %{buildroot}%{_fontdir}
```

The corporate_policies.pdf file is then installed, with 0644 permissions, in the directory just created.

```
install -d -m 0755 $RPM_BUILD_ROOT/opt/CorPor-1.0
install -m 0644 corporate_policies.pdf $RPM_BUILD_ROOT/opt/CorPor-1.0/corporate_
policies.pdf
```

After the source code is compiled, it is cleaned with the following directive:

```
%clean
rm -rf $RPM_BUILD_ROOT
```

The locations of the files and directories can be confirmed:

```
%files
%dir /opt/CorPor-1.0
%defattr(-,root,root,-)
/opt/CorPor-1.0/corporate_policies.pdf
```

If desired, you can set up different ownership. For example, if user and group michael were substituted for root in the %defattr(-,root,root,-) directive, the corporate_policies.pdf file that follows would be owned by the user and group michael.

Once the changes are made, you can proceed with the building of an RPM.

Build Your Own RPM

With the packages installed so far in this chapter, the **rpmbuild -ba** command can be run by a regular user to create binary and source RPM packages. For the corpor.spec file just described, user michael, from his home directory, could build the RPM from the directives specified in that file with the following command:

```
$ rpmbuld -ba rpmbuild/SPECS/corpor.spec
```

If successful, the last message should indicate success, as follows:

```
+ exit 0
```

If there's a problem, the last message might provide a clue. For example, the following message is straightforward, associated with a typographical error in a file. It also shows the location of the $RPM_BUILD_ROOT variable in the .spec file.

```
RPM build errors:
    File not found: /home/michael/rpmbuild/BUILDROOT/CorPor-1.0-1.el6.x86_64/
opt/CorPor-1.0/corporate_policies.pdf
```

Some error messages are even more straightforward. When I avoided an entry in the License field, the **rpmbuild** command returned the following message:

```
error: License field must be present in package: (main package)
```

Other error messages may seem more cryptic, but they point to a file which actually generates the RPM:

```
make: *** No targets specified and no makefile found.  Stop.
|error: Bad exit status from /var/tmp/rpm-tmp.tF080m (%build)
```

The extension of the rpm-tmp file varies. For more information, examine the contents of the latest rpm-tmp.* file the /var/tmp directory. Sometimes, the error message includes a line number. In some cases, the **rpmbuild -ba -vv** command provides very verbose (thus the **-vv** switch) information.

The Built RPMs

Once built, you'll be able to find the RPMs in the same rpmbuild/ subdirectory tree. With the **rpmbuild -ba** command described previously, both source and binary RPMs are built in that tree. Source RPMs can be found in the rpmbuild/SRPMS subdirectory; binary RPMs can be found in a subdirectory like rpmbuild/RPMS/noarch or rpmbuild/RPMS/x86_64. The actual directory classifies RPMs built in an architecture-independent manner, or for systems with 64-bit CPUs.

To view the differences, I built the CorPor RPM with and without the **BuildArch: noarch** directive in the .spec file. The resulting packages are

```
rpmbuild/RPMS/noarch/CorPor-1.0-1.el6.noarch.rpm
rpmbuild/RPMS/x86_64/CorPor-1.0-1.el6.x86_64.rpm
```

While the fakeroot package made it possible to run the **rpmbuild** command as a regular user, installation of the new package still requires administrative privileges. When installed, it loaded the noted corporate_policies.pdf file in the /opt/CorPor-1.0 directory. And an **rpm -qi** command applied to that package led to the output shown in Figure 12-4.

Compare that output to the information added to the corpor.spec file described earlier. That should help you better understand the purpose of each entry in the .spec file.

FIGURE 12-4

Information on the single-file RPM package

```
[michael@server1 ~]$ rpm -qi CorPor
Name        : CorPor                       Relocations: (not relocatable)
Version     : 1.0                               Vendor: (none)
Release     : 1.el6                         Build Date: Wed 16 Feb 2011 09:2
2:44 AM PST
Install Date: Wed 16 Feb 2011 09:23:04 AM PST      Build Host: server1
Group       : Miscellaneous                 Source RPM: CorPor-1.0-1.el6.src
.rpm
Size        : 7702360                          License: McGraw-Hill
Signature   : (none)
URL         : http://www.mheducation.com
Summary     : A package with one file
Description :
This is a package with one file, associated with corporate policies.
[michael@server1 ~]$
```

CERTIFICATION OBJECTIVE 12.04

Special Network Options

This section relates to three different network options, each associated with an objective for the RHCE exam. The first is related to routing, which requires an understanding of routing tables, the **route** command, and associated configuration tools and files. The second relates to the configuration of a system as a Kerberos client. It's not the first time client side–only configuration has been required on Red Hat exams, as the previous version of the RHCE exam required only the configuration of a system as a Network Information Service (NIS) or a Lightweight Directory Access Protocol (LDAP) client. Just be aware, a Kerberos client does not work unless Network Time Protocol (NTP) clients on both systems are configured to synchronize with the same NTP server.

For the third option, you need to know how to set up an iSCSI client, which can connect to storage over a standard network connection. Just to be clear, as an iSCSI client does not require SCSI hardware, there's no need to look for that now-older type of server storage media.

Configure Special IP Routes

As described in the RHCE objectives, you need to know how to "Route IP traffic and create static routes." That's really two tasks. First, it's a standard part of network configuration to set up a default route to an outside network. But there's also the

related task, when a system has two or more network devices, of setting up a special route, using a nondefault device, to a specific network.

Configure a Default Route

The default route is the path taken by a network message, when the destination address is on an outside network. When a Dynamic Host Configuration Protocol (DHCP) server is working, and is configured to assign gateway addresses, a default route is assigned with the dynamic IP address. That's normally evident in the output to the **route -n** and **netstat -nr** commands discussed in Chapter 5. One sample of such output for a system that uses a DHCP server is shown here:

```
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0      0 eth0
0.0.0.0         192.168.122.1   0.0.0.0         UG    0      0      0 eth0
```

To review, the default IPv4 address is 0.0.0.0, so the default route goes through the gateway address of 192.168.122.1. In a similar fashion, the default route for a statically configured network system is configured with the GATEWAY directive in its configuration file. Such configuration files are stored in the /etc/sysconfig/network-scripts directory, with names like ifcfg-eth0.

But there are situations, such as a temporary disconnect on a network cable, where the default route is not given by a DHCP server. Perhaps the DHCP server has to be replaced, and you'll have to set up static IP address information. In such cases, the output to the **route -n** command might look more like the following:

```
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0      0 eth0
```

In other words, since the default IP address of 0.0.0.0 is missing, there is no default route. That route can be added temporarily with the **route add** command. For example, the following command would restore the default route shown earlier:

```
# route add default gw 192.168.122.1
```

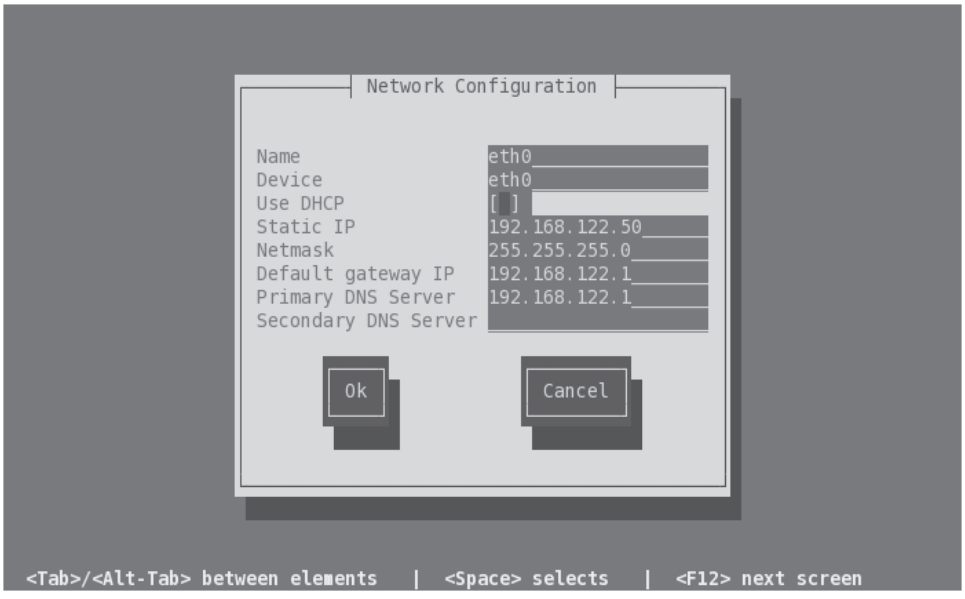
If multiple network devices exist on the local system, you can specify it; just for variety, the following command specifies the second Ethernet adapter, eth1:

```
# route add default gw 192.168.122.1 dev eth1
```

To make sure that default route survives a reboot, you'll need to make sure either the system configures that default gateway IP address as part of a static configuration, or the DHCP server used for the network can assign that gateway IP address. To review, Figure 12-5 reflects the way the default gateway IPv4 address is configured

FIGURE 12-5

A static network configuration with a default gateway



with the Network Configuration tool. In addition, you'll need to make sure the added default route survives a reboot, either by a direct change to the ifcfg-eth*x* configuration file, or indirectly with the Network Configuration tool.

Some systems may have multiple network devices connected to the same network. In that case, you may need to configure a special route. As you might see, special routes are not possible with the console-based Network Configuration tool.

Configure a Special Route

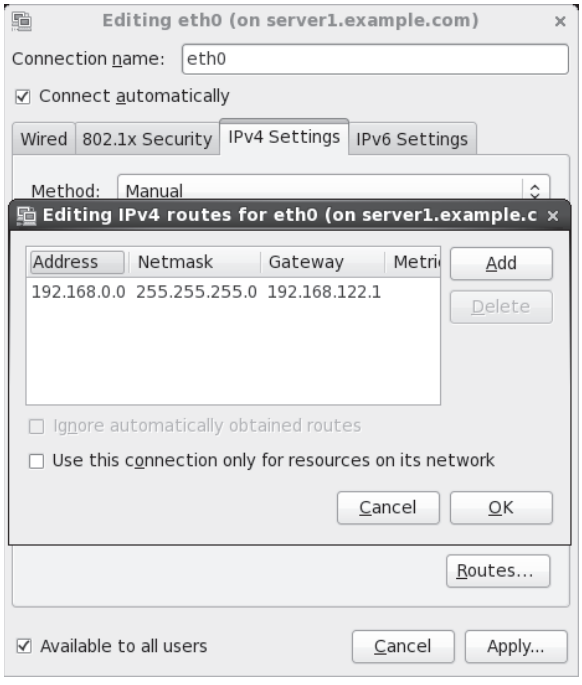
One way to configure a special route is with the Network Connections tool. As discussed in Chapter 5, you can start it from a GUI console with the **nm-connection-editor** command. Select an existing wired or wireless network device, and click Edit. Under either the IPv4 or IPv6 tab, there's a Routes button for special routes. Click it to see the window shown in Figure 12-6.

on the job

The Network Connections tool does not work unless the NetworkManager service in the /etc/init.d directory is active.

FIGURE 12-6

A special route for a specific network device



When applied, it writes a route-eth0 file in the /etc/sysconfig/network-scripts directory. The following is the complete contents of that file:

```
ADDRESS0=192.168.0.0

NETMASK0=255.255.255.0

GATEWAY0=192.168.122.1
```

When the network service is restarted, it's applied to the routing table. Based on the previously configured routing table, the following is the result:

```
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.0.0     192.168.122.1 255.255.255.0 UG    0      0      0 eth0
192.168.122.0 0.0.0.0         255.255.255.0 U     1      0      0 eth0
0.0.0.0         192.168.122.1 0.0.0.0         UG    0      0      0 eth0
```

Set Up a Kerberos Client

For the purpose of an exam, as well as on the job, it's almost always best to keep the solutions as simple as possible. That's where the Authentication Configuration tool can help. To see what this tool does to help configure a Kerberos client, you could back up the files in the /etc/sssd directory, along with the /etc/nsswitch.conf configuration file. It's related to the System Security Services Daemon, controlled by the /etc/init.d/sssd script. If you've previously configured a system to authenticate via an LDAP server in Chapter 8, the sssd service may already be running.

As there is no RHCE objective related to the configuration of a Kerberos server, that process will not be covered in this book. What you should know is that Kerberos servers as configured on RHEL 6 don't have their own authentication databases. So for a valid client connection to a Kerberos server, you'll also need a connection to a network authentication database such as LDAP.

on the Job

If you configure a Kerberos server, make sure port 88 is open in the relevant firewalls.

What Is Kerberos

Kerberos is a network authentication protocol originally developed at the Massachusetts Institute of Technology (MIT) that supports secure identification of networked systems. Since it works with systems, a separate protocol such as LDAP is required for user-based authentication.

Two systems configured with and confirmed by Kerberos can communicate in encrypted format, with a symmetric key. That key is granted by a Key Distribution Center (KDC), which consists of an Authentication Server (AS) and a Ticket Granting Server (TGS). When authentication is confirmed, the Kerberos client gets a ticket good for a limited time, typically eight hours.

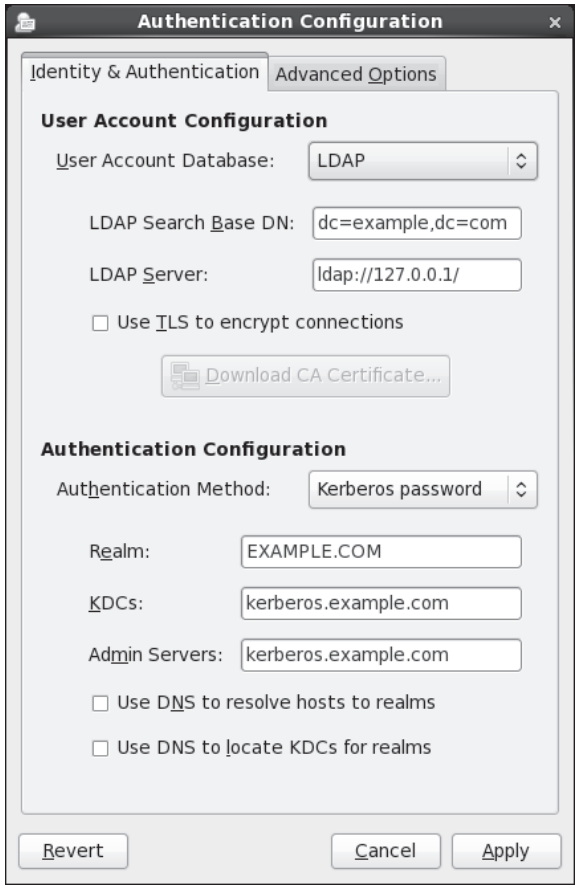
Given the importance of time to Kerberos, it may not work unless configured clients and servers also use NTP services, as discussed in Chapters 5 and 17.

The Graphical Authentication Configuration Tool

One way to open the GUI version of the Authentication Configuration tool is with the **authconfig-gtk** command. That should open the Authentication Configuration tool with the two tabs shown in Figure 12-7. While other authentication databases are supported, the focus is on LDAP, especially since LDAP is the selected network client authentication option for the RHCSA exam. The options in the LDAP half of the Identity And Authentication tab were discussed in Chapter 8.

FIGURE 12-7

Configure a Kerberos-based client with the graphical Authentication Configuration Tool.



The focus of this section is on the second half of the tab. For a Kerberos-based client, you'd retain Kerberos Password as the Authentication Method. The other options are

- **Realm** By convention, the Kerberos realm is the same as the domain name for the network, in uppercase letters. It's necessary if you configure DNS support for Kerberos.
- **KDCs** The KDC is the Kerberos Key Distribution Center. The entry here should correspond either to the Fully Qualified Domain Name (FQDN) or the IP address of the actual Kerberos server.

- **Admin Servers** The administrative server associated with the KDC is frequently located on the same system. On the Kerberos administrative server, the kadmind daemon is running.
- **Use DNS To Resolve Hosts To Realms** Where a trusted DNS server exists for the local network, you can allow the local system to use a DNS server to find the realm. If this option is activated, the Realm text box will be blanked out.
- **Use DNS To Locate KDCs For Realms** Where a trusted DNS server exists for the local network, you can allow the local system to use a DNS server to find the KDC and administrative server. If this option is activated, the KDCs and Admin Servers text boxes will be blanked out.

For the purpose of this section, accept the default options as shown in Figure 12-7. Click Apply. After a few moments, the Authentication Configuration window will close and changes will be made to the aforementioned configuration files. In addition, the sssd service will be started.

The Console Authentication Configuration Tool

Perhaps a weakness of the console-based version of the Authentication Configuration tool is how it's possible to set up a system as a Kerberos client without a connection to a network authentication database. If you're using the console tool, keep in mind the need for an actual user authentication database to go with Kerberos.

To start the text-mode version of the Authentication Configuration tool, run the **authconfig-tui** command. As shown in Figure 12-8, you'll need to activate LDAP at least for authentication.

After selecting Next, the tool prompts for LDAP settings information, as discussed in Chapter 8. After that screen, you'll see the Kerberos Settings screen shown in Figure 12-9. The default options shown here are the same as those shown in the graphical version of the tool from Figure 12-7.

You may also need to set up changes to configuration files, described next.

Changes to Configuration Files

Several changes are made to local configuration files, specifically /etc/nsswitch.conf and /etc/sssd/sssd.conf, based on the configuration of a Kerberos client.

FIGURE 12-8

Configure a Kerberos-based client with the console Authentication Configuration tool.

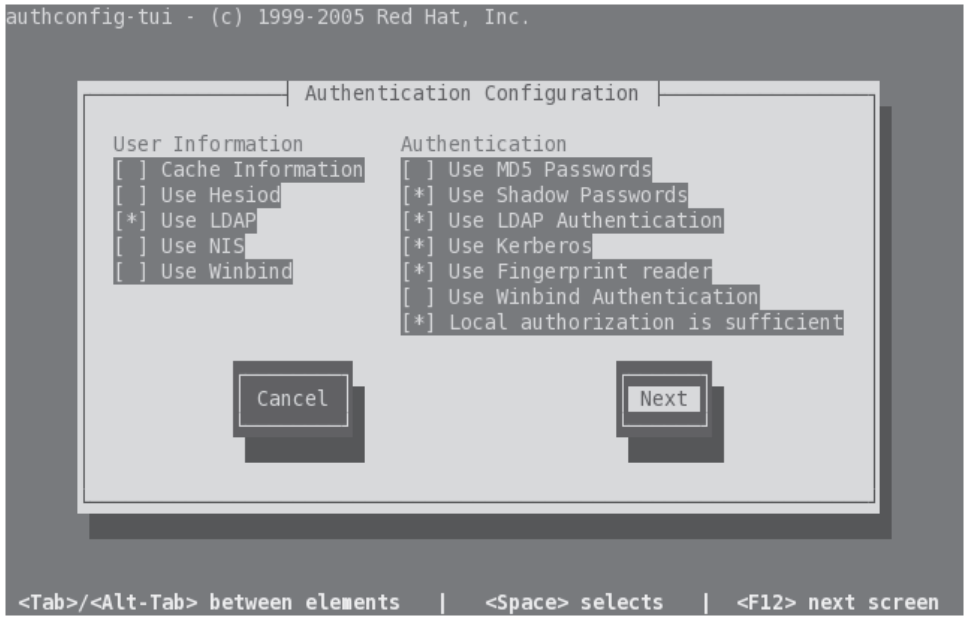
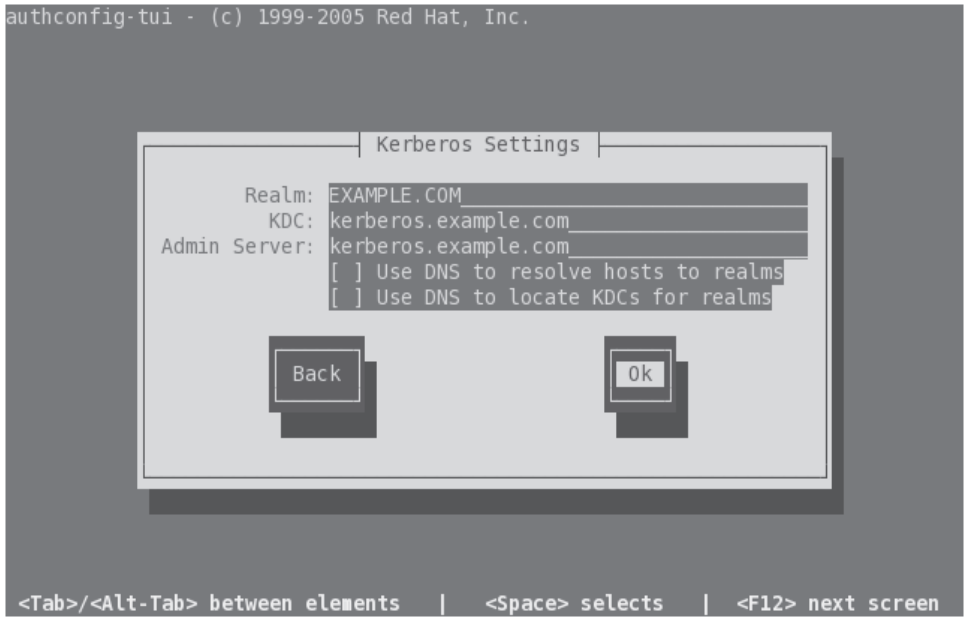


FIGURE 12-9

Specify Kerberos client settings.



First, look at the changes to the /etc/nsswitch.conf file, where the **sss** setting is added to the authentication files shown here:

```
passwd:   files sss
shadow:   files sss
group:    files sss
```

In other words, when the local system looks for usernames and passwords, the files of the local shadow password suite are checked. If the username is not found there, control is turned over to the sssd daemon described earlier. Those files are configured in the /etc/sssd directory.

Changes to the sssd daemon configuration are added to the end of /etc/sssd/sssd.conf file. The added directives are shown in Figure 12-10 and are explained in Table 12-3. The directives are listed in the order that they're shown in the figure.

Connect to Remote iSCSI Storage

The relevant RHCE objective is to "configure a system as an iSCSI initiator that persistently mounts an iSCSI target." The iSCSI initiator is a client. The iSCSI target is the shared storage on the server, which communicates with the client over port 3260. Once the client is configured, you'll have access to the iSCSI target; that target will look like just another hard drive. Of course, the response will probably be slower, but that depends on the speed of and the traffic on that network.

To set up an iSCSI client, you'll need the iscsi-initiator-utils packages, along with any dependencies. Then you'd use the **iscsiadm** command to discover available iSCSI targets. One method is with the following command:

```
# iscsiadm -m discoverydb -t st -p 192.168.122.1 -D
```

FIGURE 12-10

Kerberos Configuration in /etc/sssd/sssd.conf

```
[domain/default]
auth_provider = krb5
cache_credentials = True
ldap_id_use_start_tls = False
debug_level = 0
krb5_kpasswd = kerberos.example.com
ldap_search_base = dc=example,dc=com
krb5_realm = EXAMPLE.COM
chpass_provider = krb5
id_provider = ldap
ldap_uri = ldap://127.0.0.1/
krb5_kdcip = kerberos.example.com
ldap_tls_cacertdir = /etc/openldap/cacerts
```

TABLE 12-3

Kerberos-Based Directives in /etc/sssd/sssd.conf

Directive	Description
auth_provider	Set to krb5 for Kerberos authentication.
cache_credentials	Stores authentication information locally if set to true.
ldap_id_use_start_tls	Requires Transport Layer Security (TLS) to encrypt connections to the LDAP server, if set to true.
debug_level	Configures when messages are logged; may be set between 0 to limit logging to critical messages and 10 for more information.
krb5_kpasswd	Can specify the FQDN of the applicable server.
ldap_search_base	Specifies the domain components of the LDAP server.
krb5_realm	Notes the name of the Kerberos realm, should be in all uppercase letters.
chpass_provider	Specifies the provider for the Kerberos password.
id_provider	Configures the identity provider for the domain, usually ldap.
ldap_uri	Includes the Uniform Resource Identifier (URI) for the LDAP server.
krb5_kdcip	Notes the IP addresses or FQDN of Kerberos servers; multiple servers can be included.
ldap_tls_cacertdir	Specifies the directory with a secure certificate for the LDAP server.

To interpret, this **iscsiadm** command queries iSCSI targets. It works in discovery database (**discoverydb**) mode (**-m**), where the discovery type (**-t**) requests that iSCSI servers actually send available targets (**sendtargets** or **st**), with a portal (**-p**) of the noted IP address, to discover (**-D**) shared storage.

If successful, you'll see output similar to the following:

```
192.168.122.1:3260,1 iqn.2011-02.com.example:for.all
```

You should then be able to start the iSCSI service with a command like

```
# /etc/init.d/iscsi start
```

If successful, you'll be able to review the available iSCSI shared storage, with the **/etc/init.d/iscsi status** command. The output should be similar to that shown in Figure 12-11.

FIGURE 12-11 Discovered iSCSI storage

```
[root@server1 ~]# /etc/init.d/iscsi status
iSCSI Transport Class version 2.0-870
version 2.0-872
Target: iqn.2011-02.com.example:for.all
        Current Portal: 192.168.122.1:3260,1
        Persistent Portal: 192.168.122.1:3260,1
                **********
                Interface:
                **********
                Iface Name: default
                Iface Transport: tcp
                Iface Initiatorname: iqn.1994-05.com.redhat:dd2d45d064b4
                Iface IPaddress: 192.168.122.50
                Iface HWaddress: <empty>
                Iface Netdev: <empty>
                SID: 6
                iSCSI Connection State: LOGGED IN
                iSCSI Session State: LOGGED_IN
                Internal iscsid Session State: NO CHANGE
                ************************
                Negotiated iSCSI params:
                ************************
                HeaderDigest: None
                DataDigest: None
                MaxRecvDataSegmentLength: 262144
                MaxXmitDataSegmentLength: 8192
                FirstBurstLength: 65536
                MaxBurstLength: 262144
                ImmediateData: Yes
                InitialR2T: Yes
                MaxOutstandingR2T: 1
                ************************
                Attached SCSI devices:
                ************************
                Host Number: 7  State: running
                scsi7 Channel 00 Id 0 Lun: 0
[root@server1 ~]#
```

exam Watch

To set up an iSCSI client, you'll need access to an iSCSI target, acting as a server. Normally, port 3260 will be open on that server, making it accessible with the* iscsiadm *command described in this chapter.

You should then be able to manage the shared storage as if it were a new hard drive on the local system. The hard drive device file will show up in the /var/log/messages file with information like the following, which points to device file /dev/sg3.

```
server1 kernel: scsi 7:0:0:0: Attached scsi generic sg3 type 12
```

You should then be able to create partitions and more on the new /dev/sg3 drive just as if it were a local drive, based on the techniques discussed in Chapter 6. Of course, a "persistent mount" as described in the relevant RHCE objective requires that you make sure the iSCSI service starts the next time the system is rebooted with a command like

```
# chkconfig iscsi on
```

To make sure there's an actual mount, you may also need to set up a partition that's actually mounted in the /etc/fstab file. In practice, the actual device file for the iSCSI drive may vary on each reboot. Therefore, such mounts should be configured with the Universally Unique Identifier (UUID) numbers described in Chapter 6.

You do not need to create an iSCSI target storage server for the RHCE exam. But as the configuration of an iSCSI server is relatively simple, you might consider creating one for that purpose. Red Hat developer Daniel Berrangé has created an excellent introduction to the creation of an iSCSI storage server at http://berrange.com/tags/tgtadm/.

SCENARIO & SOLUTION

You need to set up a daily task to back up files in the /home directory	Set up a script in the /etc/cron.daily directory with appropriate backup commands to copy files from /home.
You've been told to set up IPv6 forwarding on a system	Include the **net.ipv6.conf.all.forwarding=1** setting in /etc/sysctl.conf, and activate it with the **sysctl -p** command.
You need to set up source code for a single-file RPM	Set up the single file, along with an executable **configure** script in a dedicated directory; then set it up in a compressed tar archive.
You need to set up a special static route over device eth1	Use the Network Connections tool to set up that special route, given the network address, subnet mask, and desired gateway IP address.
You need to set up a system as a Kerberos client	Use the GUI Authentication Configuration tool; the realm should be the uppercase listing for the domain. You'll also need the FQDN for the KDC and Kerberos administration servers (which may be the same).
You need to set up an iSCSI initiator	Install the iscsi-initiator-utils package, use the **iscsiadm** command to discover available iSCSI targets, and make sure the iscsi service is active on reboot.

CERTIFICATION SUMMARY

Linux administrators need to configure scripts on a regular basis. Sample scripts are already available in different /etc/cron.* directories. Normally, scripts start with the **#!/bin/bash** line, which sets up the language for the rest of the script. Administrative scripts can use bash commands, along with operators such as **for**, **if**, **do**, and **test**. Kernel run-time parameters can be found in the /proc/sys directory. But changes to such files are temporary. For more permanent changes, you'd set up options in the /etc/sysctl.conf file. Changes to that file can be implemented with the **sysctl -p** command. Many standard options relate to networking.

One new RHCE requirement is to create an RPM from a single file. To do so, you need to know how to set up a source-code archive. The rpmdevtools and rpm-build packages can help. The **rpmdev-setup** tree command can help set up the needed directories. The **rpmdev-newspec** command can help create a template for a .spec file. The **rpmbuild** command can then be used to process the instructions in the spec file along with the packaged source code into an SRPM and an RPM package that can be installed on different systems.

The RHCE objectives include requirements for several special network options. With the help of the Network Connections tool, special IP routes can be configured in a file in the /etc/sysconfig/network-scripts directory. Kerberos clients can be configured in the /etc/sssd/sssd.conf file, referenced through the /etc/nsswitch.conf file. Perhaps the easiest way to configure a Kerberos client is with the GUI Authentication Configuration tool.

TWO-MINUTE DRILL

Here are some of the key points from the certification objectives in Chapter 12.

Automate System Maintenance

- ❑ Standard administrative scripts can provide a model for custom scripts to automate system maintenance tasks.
- ❑ Various command operators within scripts include **do**, **for**, **if**, and **test**.
- ❑ Many Linux administrative scripts start with **#!/bin/bash**, a reference to the bash shell.

Kernel Run-Time Parameters

- ❑ Kernel run-time parameters are located in the /proc/sys directory.
- ❑ Many kernel run-time parameters relate to network options such as forwarding and security.
- ❑ Kernel run-time parameters can be configured on a permanent basis with the help of the /etc/sysctl.conf file.

Create an RPM Package

- ❑ Available source RPMs can be used as a model to help create your own RPMs.
- ❑ RPM source-code components can be found in user home directories, in the rpmbuild/ subdirectory, as configured with the **rpmdev-setuptree** command.
- ❑ The actual source code can be found in the rpmbuild/SOURCES subdirectory.
- ❑ RPMs are built with the **rpmbuild** command, based on a .spec file in the rpmbuild/SPECS subdirectory. You can create a standard .spec template with the **rpmdev-newspec** command.
- ❑ Built RPMs can be found in the rpmbuild/SRPMS and rpmbuild/RPMS subdirectories.

Special Network Options

- The default network route to an outside network goes through a gateway IP address.
- Special routes to different networks can be configured through certain IP specific network devices.
- To configure a Kerberos client, you need to modify the /etc/sssd/sssd.conf file.
- A connection to the Kerberos client also requires a connection to a network authentication service such as LDAP.
- To configure an iSCSI client, you need the iscsi-initiator-utils package, which can be used to connect to iSCSI storage with the **iscsiadm** command.
- To make sure the iSCSI connection survives a reboot, you'll need to activate the iscsi service.

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.

Automate System Maintenance

1. What exit number is associated with success in a script?

   _____

2. What operator means "anything but"?

   _____

Kernel Run-Time Parameters

3. What's the full path to the file associated with the **net.ipv4.ip_forward** parameter?

   _____

Create an RPM Package

4. When source code is installed from an SRPM by user stephanie in her home directory, what is the full path to the actual source code?

   _____

5. What's the name of the package with a command that can be used to create a spec file template?

   _____

6. What common file is needed in a source-code directory to process standard source-code packages? Bonus: what should be different about permissions on that file, when compared to a regular file?

   _____

7. What command can be used to create just a regular RPM from source code, based on the test.spec file in the rpmbuild/SPECS subdirectory? Assume all required source code is in the correct location.

_____

8. When user tim builds a regular RPM associated with 64-bit systems, what's the full path to the directory with that RPM?

_____

Special Network Options

9. What are the three types of IP addresses associated with a special route?

_____

10. In what file is a Kerberos client configured?

_____

11. What is the standard Kerberos realm for the server1.example.com system?

_____

12. What service script that should be running on reboot on a properly configured iSCSI target?

_____

LAB QUESTIONS

Several of these labs involve configuration exercises. You should do these exercises on test machines only. It's assumed that you're running these exercises on virtual machines such as KVM.

Red Hat presents its exams electronically. For that reason, the labs in this and future chapters are available from the CD that accompanies the book, in the Chapter12/ subdirectory. In case you haven't yet set up RHEL 6 on a system, refer to Chapter 1 for installation instructions.

The answers for the labs follow the Self Test answers for the fill-in-the-blank questions.

SELF TEST ANSWERS

Automate System Maintenance

1. The exit number associated with success on a script is 0.In a script, the operator that means anything but is the exclamation point (**!**). (It's also known as a "bang" in the world of Linux.)

System Run-Time Parameters

2. The full path to the file associated with the **net.ipv4.ip_forward** parameter is /proc/sys/net/ipv4/ip_forward.

Create an RPM Package

3. The full path to the actual source code, installed from an SRPM by user stephanie, is /home/stephanie/rpmbuild/SOURCES.
4. The name of the package with a command that can be used to create a spec file template is **rpmdev-newspec**.
5. The common file needed in standard source-code packages is named configure. Bonus: That file should also have executable permissions.
6. The command that can be used to create just a regular RPM from source code under the given conditions is **rpmbuild -ba rpmbuild/SPECS/test.spec**.
7. When user tim builds a regular RPM associated with 64-bit systems, the full path to the directory with that RPM is /home/tim/rpmbuild/RPMS/x86_64.

Special Network Options

8. The three types of IP addresses associated with a special route are the network address, the network or subnet mask, and the gateway address.
9. On RHEL 6, Kerberos clients are normally configured in the /etc/sssd/sssd.conf file.
10. The standard Kerberos realm for the server1.example.com system is EXAMPLE.COM.
11. The service script that should be active on reboot on a properly configured iSCSI target is iscsi.

LAB ANSWERS

Lab 1

Success in this lab should be straightforward. If you've run the **date** command as suggested in the body of the lab, the files from the /etc directory should be copied to the /backup directory a minute later.

The simplest way to set up that script to be run on an hourly basis is to configure it in the /etc/cron.hourly directory. The script needs only two lines. The following is one example of how that script could be configured:

```
#/bin/bash
# /bin/cp -ar /etc /backup
```

Of course, for that script to work, it requires executable permissions. The name of the file does not matter, as long as it's in the /etc/cron.hourly directory. For the purpose of this lab, I created a script named whatever.cron in that directory with the two lines just shown. The name of the script does not matter, as long as it's saved to the /etc/cron.hourly directory.

As defined in the 0hourly script in the /etc/cron.d directory, hourly scripts are executed one minute past every hour. Since I'm by nature impatient (and tested this file on April 14), I ran the following command to advance the clock to the next hour (11 A.M.):

```
# date 04141100
```

One minute later, I found the contents of the /etc directory in the /backup directory.

Given the importance of the system clock for Kerberos-based authentication, you should restore the original time.

Lab 2

If you've followed the instructions in this lab, the /etc/sysctl.conf file should now have the following entry:

```
net.ipv4.icmp_echo_ignore_all = 1
```

That just makes sure the new setting survives a reboot. You may have also set the associated file, /proc/sys/net/ipv4/icmp_echo_ignore_all, to 1, or run the **sysctl -p** command to implement the change before the system is rebooted.

Of course, success can be confirmed with a **ping** command both from local and remote systems. If you want to restore the original configuration, return to the server1.example.com system, and then remove the net.ipv4.icmp_echo_ignore_all option from the /etc/sysctl.conf file.

Lab 3

If successful, you'll have a dedicated .spec file in the rpmbuild/SPECS directory, and at least a binary RPM in a directory like rpmbuild/RPMS/noarch. Yes, you'll have to take administrative privileges to run the **rpm** command; when installed, the vsftpd.conf file should be installed in the /opt/sampleftp directory.

If that doesn't work, you may need to refer back to the body of the chapter for more information. To summarize, remember to perform the following steps:

1. Set up a compressed archive for the directory with the vsftpd.conf file.
2. Install the rpm-build, rpmdevtools, and gcc packages, along with the dependencies shown when running the **rpmbuild -ba** command.
3. Use the **rpmdev-setuptree** command to configure a directory tree to build the new RPM.
4. Use the **rpmdev-newspec** command to set up a newpackage.spec file to process the RPM source code.
5. In the .spec file, make sure to fill in the name, version, release number, summary, group, license, and a description. You should comment out the BuildRequires and Requires options, as there are no dependencies for this one file RPM.
6. Make sure to set up %install directives at the end of the file; for example, the following directives would make sure the binary RPM, when installed, sets up the file in the specified directory.

```
install -d -m 0755 $RPM_BUILD_ROOT/opt/sampleftp
install -m 0644 vsftpd.conf $RPM_BUILD_ROOT/opt/sampleftp/vsftpd.conf
```

7. Make sure the following directives confirm the noted directories, ownership, and file name from the source code. Of course, you can change the ownership of the loaded file with the help of the **defattr** directive.

```
%files
%dir /opt/sampleftp
%defattr(-,root,root,-)
/opt/sampleftp/vsftpd.conf
```

8. Save the file and apply the **rpmbuild -ba** command to the .spec file.
9. Address any build errors that may appear in the **rpmbuild -ba** command output, or the latest /var/tmp/rpm-tmp.* file.
10. Use the **rpm** command to install the newly configured RPM from an rpmbuild/RPMS/norarch (or rpmbuild/RPMS/x86_64) subdirectory.

Lab 4

As with Lab 3, you'll have a different dedicated .spec file in the rpmbuild/SPECS directory, and at least a binary RPM in a directory like rpmbuild/RPMS/x86_64. Yes, you'll have to take administrative privileges to run the **rpm** command; when installed, the OVERVIEW file should be installed in the /opt/postfixinfo directory.

If you're ready, this can be a bit of a preview of the next chapter. The OVERVIEW file provides background information on the philosophy behind the Postfix service.

Lab 5

If you use the Network Connections tool to set up a special route, it should set up a special file in the /etc/sysconfig/network-scripts directory. If the specified network adapter is eth0, that special file would be route-eth0. Given the parameters used for the outsider1.example.org network as discussed in Chapter 1, that file would contain the following three lines:

```
ADDRESS0=192.168.100.0

NETMASK0=255.255.255.0

GATEWAY0=192.168.122.1
```

Of course, if the outsider1.example.org system is on a different network, the contents of the route-eth0 file would change accordingly.

Lab 6

Success in this lab means the following:

1. The sssd service is running, and is set to run the next time the system is booted, with commands like **/etc/init.d/sssd start** and **chkconfig sssd on**.
2. The /etc/nsswitch.conf file includes the following entries for the shadow password suite:

```
passwd:   files sss
shadow:   files sss
group:    files sss
```

3. The /etc/sssd/sssd.conf file should include the following entries:

```
[domain/default]

ldap_id_use_start_tls = False

cache_credentials = True

auth_provider = krb5

debug_level = 0

krb5_kpasswd = maui.example.org

ldap_schema = rfc2307

ldap_search_base = dc=example,dc=org

krb5_realm = EXAMPLE.ORG

chpass_provider = krb5

id_provider = ldap

ldap_uri = ldap://192.168.100.1

krb5_kdcip = maui.example.org

ldap_tls_cacertdir = /etc/openldap/cacerts
```