Chapter 7

Package Management

CERTIFICATION OBJECTIVES

After installation is complete, systems are secured, filesystems are configured, and other initial setup tasks are completed, you still have work to do. Almost certainly before your system is in the state you desire, you will be required to install or remove packages. To make sure the right updates are installed, you need to know how to get a system working with Red Hat Subscription Management (RHSM) or the repository associated with a rebuild distribution.

To accomplish these tasks, you need to understand how to use the **rpm** and **yum** commands in detail. Although these are "just" two commands, they are rich in detail. Entire books have been dedicated to the **rpm** command, such as the *Red Hat RPM Guide* by Eric

Foster-Johnson. For many, that degree of in-depth knowledge of the **rpm** command is no longer necessary, given the capabilities of the **yum** command and the additional package management tools provided in RHEL 7.

CERTIFICATION OBJECTIVE 7.01

The Red Hat Package Manager

One of the major duties of a system administrator is software management. New applications are installed. Services are updated. Kernels are patched. Without the right tools, it can be difficult to figure out what software is on a system, what is the latest update, and what applications depend on other software. Worse, you may install a new software package only to find it has overwritten a crucial file from a currently installed package.

INSIDE THE EXAM

Administrative Skills

As the management of RPM packages is a fundamental skill for Red Hat administrators, it's reasonable to expect to use the **rpm**, **yum**, and related commands on the RHCSA exam. In fact, the RHCE exam effectively assumes knowledge of such commands and more as prerequisite skills. The RHCSA objectives include two specific requirements addressed in this chapter:

- Install and update software packages from Red Hat Network, a remote repository, or from the local filesystem
- Update the kernel package appropriately to ensure a bootable system

Another closely related objective is the **tar** archiving utility, which is covered in Chapter 9. Before Red Hat introduced RPM packages, tar archives were the standard method for distributing software.

Now let's break down these skills a bit. If you don't have access to the RHN, don't be intimidated. For RHEL 7, the RHN-hosted service has been phased out in favor of Red Hat Subscription Management (RHSM, which can be accessed via a web interface from the Red Hat Customer Portal). You can use **yum** to install and update packages from RHSM; you can use the same **yum** commands to install and update packages from a remote third-party repository.

The Red Hat Package Manager (RPM) was designed to alleviate these problems. With RPM, software is managed in discrete *packages.* An RPM package includes the software with instructions for adding, removing, and upgrading those files. When properly used, the RPM system can back up key configuration files before proceeding with upgrades and removals. It can also help you identify the currently installed version of any RPM-based application.

RPMs and the **rpm** command are very focused on individual packages, which in many cases is far from ideal and is why **rpm** has been supplemented with the **yum** command. With a connection to a repository such as that available from RHSM or third-party "rebuilds" such as Scientific Linux, you'll be able to use **yum** to satisfy dependencies automatically.

What Is a Package?

In the generic sense, an RPM package is a container of files. It includes the group of files associated with a specific program or application, which normally contains binary files, installation scripts, as well as configuration and documentation files. It also includes instructions on how and where these files should be installed and uninstalled.

An RPM package name usually consists of the version, the release, and the architecture for which it was built. For example, the fictional penguin-3.4.5-26.el7.x86_64.rpm package is version 3.4.5, release 26.el7. The x86_64 indicates that it is suitable for computers built to the AMD/Intel 64-bit architecture.

Many RPM packages include software compiled for a specific CPU type (for example, x86_64). You can identify the CPU type for the system with the uname -i **or** uname -p **command. More information about your processor can be found in /proc/cpuinfo.**

What Is the RPM Database?

At the heart of this system is the RPM database, which is stored locally on each machine in the /var/lib/rpm directory. Among other things, this database tracks the version and location of every file in each RPM. The RPM database also maintains an MD5 checksum of each file. With the checksum, you can use the **rpm -V *package*** command to determine whether any file from that RPM package has changed. The RPM database makes it easy to add, remove, and upgrade individual packages because it's configured to know which files to handle and where to put them.

RPM also manages conflicts between packages. For example, assume you have a package that installs a configuration file, and you want to update from an older to a newer version of the software. Call the original configuration file /etc/*someconfig*.conf. You've already installed package X. If you then try to install a more recent version of package X, the RPM

can be configured to preserve the original configuration file and install the new one as /etc/*someconfig*.conf.rpmnew.

Alternatively, the RPM creator can build the RPM in such a way that it will back up the original /etc/*someconfig*.conf file (with a filename such as /etc/*someconfig*.conf.rpmsave) before upgrading package X and then replace the configuration file with a new version. This may occur if the format of the old configuration file is incompatible with the new release of the software.

Although RPM upgrades are supposed to preserve or save existing configuration files, there are no guarantees, especially if the RPM is created by someone other than Red Hat. It's best to back up all applicable configuration files before upgrading any associated RPM package.

What Is a Repository?

RPM packages are frequently organized into repositories. Generally, such repositories include groups of packages with different functions. For example, the Red Hat Portal gives access to the following RHEL 7 Server repositories (additional repositories are available):

- **Red Hat Enterprise Linux Server** The main repository, which includes both the packages associated with the original installation of RHEL 7, along with updates.
- **RHEL Server Optional** A large group of open-source packages, provided with no support from Red Hat.
- **RHEL Server Supplementary** A collection of packages released under licenses other than open source, such as the IBM Java Runtime and Development Kit.
- **RHEL Extras** Includes Docker, a platform for packaging and managing applications using a lightweight form of virtualization known as Linux Containers.
- **RHN Tools** Client tools to subscribe to the Red Hat Network via a Satellite server, along with utilities for automating Kickstart installations.

In contrast, the repository categories for third-party Red Hat clones vary. Generally, they include categories such as main and extras. In most cases, whereas the main repository includes just the packages available from the released DVD, updated packages are configured in their own repository.

Each repository includes a database of packages in a repodata/ subdirectory. That database includes information on each package and allows installation requests to include all dependencies. If you have a subscription to RHSM, access to the Red Hat repositories is enabled in the product-id.conf and subscription-manager.conf files, in the /etc/yum/pluginconf.d directory. Those files are discussed later in this chapter.

Later in this chapter, you'll examine how to configure connections to repositories with the configuration files associated with the **yum** command.

A dependency is a package that needs to be installed to make sure all the features of a target package work as designed.

Install an RPM Package

There are three basic commands that *may* install an RPM. They won't work if there are dependencies. For example, if you haven't installed the SELinux policy development tool package (policycoreutilis-devel) and try to install the SELinux configuration GUI (policycoreutilis-gui), you'll get the following message (version numbers may vary):

```
# rpm -i policycoreutils-gui-2.2.5-11.el7.x86_64.rpm
error: Failed dependencies:
        policycoreutils-devel = 2.2.5-11.el7 is needed by↵
policycoreutils-gui-2.2.5-11.el7.x86_64
```

One way to test this is to mount the RHEL 7 DVD with the **mount /dev/cdrom /media** command. Next, find the noted policycoreutils-gui package in the Packages/ subdirectory. Alternatively, you could download that package directly from the Red Hat Portal or a configured repository with the **yumdownloader policycoreutils-gui** command. This and other **yum** commands are discussed later in this chapter. Be aware that some Linux GUI desktop environments automatically mount a CD/DVD media that is inserted into an associated drive. If so, you'll see the mount directory in the output to the **mount** command.

When dependency messages are shown, **rpm** does not install the given package. Note the dependency messages: policycoreutils-gui requires a policycoreutils-devel package of the same version number.

Sure, you can use the --nodeps option to make rpm ignore dependencies, but that can lead to other problems, unless you install those dependencies as soon as possible. The best option is to use an appropriate yum command, described later in this chapter. In this case, a yum install policycoreutils-gui command would automatically install the other dependent RPM as well.

If you're not stopped by dependencies, the following three basic commands can install RPM packages:

```
# rpm -i packagename
# rpm -U packagename
# rpm -F packagename
```

The **rpm -i** option installs the package, if it isn't already installed. The **rpm -U** option upgrades any existing package or installs it if an earlier version isn't already installed.

The **rpm -F** option upgrades only existing packages. It does not install a package if it wasn't previously installed.

We like to add the **-vh** options with the **rpm** command. These options add verbose mode and use hash marks that can help monitor the progress of the installation. So when we use **rpm** to install a package, we run the following command:

```
# rpm -ivh packagename-version.arch.rpm
```

There's one more thing associated with a properly designed RPM package. When unpacking a package, the **rpm** command checks to see whether it would overwrite any configuration files. The **rpm** command tries to make intelligent decisions about what to do in this situation. As suggested earlier, if the **rpm** command chooses to replace an existing configuration file, it provides a warning (in most cases) similar to this:

```
# rpm -U penguin-3.26.x86_64.rpm
warning: /etc/someconfig.conf saved as /etc/someconfig.conf.rpmsave
```

The **rpm** command normally works in the same fashion when a package is erased with the **-e** switch. If a configuration file has been changed, it's also saved with an .rpmsave extension in the same directory.

It's up to you to look at both files and determine what modifications, if any, need to be made. Of course, as not every RPM package is perfect, there's always a risk that such an update would overwrite that critical customized configuration file. In that case, backups are important.

In general, the **rpm** commands to upgrade a package work only if the package being installed is of a newer version. Sometimes, an older version of a package is desirable. As long as there are no security issues with the older package, administrators may be more comfortable with slightly older releases. Bugs that may be a problem on a newer package may not exist in an older version of that package. So if you want to "downgrade" a package with the **rpm -i**, **-U**, or **-F** command, the **--force** switch can help.

on the job

If you've already customized a package and then upgraded it with rpm, check if there is a saved configuration file ending with an .rpmnew extension. Use it as a guide to change the settings in the new configuration file. But remember, with upgrades, there may be additional required changes. Therefore, you should test the result for every conceivable production environment.

Uninstall an RPM Package

The **rpm -e** command uninstalls a package. But first, RPM checks a few things. It performs a dependency check to make sure no other packages need what you're trying to uninstall. If dependent packages are found, **rpm -e** fails with an error message identifying these packages. With properly configured RPMs, if you have modified related configuration files,

RPM makes a copy of the file, adds an .rpmsave extension to the end of the filename, and then erases the original. It can then proceed with the uninstallation. When the process is complete, it removes the package from the database.

Be very careful about which packages you remove from a system. Like many other Linux utilities, RPM may silently let you shoot yourself in the foot. For example, if you were to remove the packages that include the running kernel, it could render that system unusable at the next boot.

Install RPMs from Remote Systems

With the RPM system, you can even specify package locations similar to an Internet address, in URL format. For example, if you want to apply the **rpm** command to the foo.rpm package on the /pub directory of the ftp.rpmdownloads.com FTP server, you can install this package with a command such as the following:

```
# rpm -ivh ftp://ftp.rpmdownloads.com/pub/foo.rpm
```

Assuming you have a network connection to that remote server, this particular **rpm** command logs on to the FTP server anonymously and downloads the file. Unfortunately, an attempt to use wildcards in the package name with this command leads to an error message associated with "file not found." The complete package name is required, which can be an annoyance.

If you installed RHEL 7 from an FTP server as instructed in Chapters 1 and 2, you could substitute the associated URL, along with the exact name of the package. For example, based on the FTP server configured in Chapter 1 and the aforementioned policycoreutils-gui package, the appropriate command would be

```
# rpm -ivh ftp://192.168.122.1/pub/inst/policycoreutils-gui ↵
-2.2.5-11.el7.x86_64.rpm
```

If the FTP server requires a username and password, you can include them in the following format

```
ftp://username:password@hostname:port/path/to/remote/package.rpm
```

where *username* and *password* are the username and password you need to log on to this system, and *port,* if required, specifies a nonstandard port used on the remote FTP server.

Based on the preceding example, if the username is **mjang** and the password is **Ila451MS**, you could install an RPM directly from a server with the following command:

```
# rpm -ivh ftp://mjang:Ila451MS@192.168.122.1/pub/inst/policycoreutils-gui ↵
-2.2.5-11.el7.x86_64.rpm
```

RPM Installation Security

Security can be a concern, especially with RPM packages downloaded over the Internet. If a "black hat" hacker were to somehow penetrate a third-party repository, how would you know that packages from those sources were genuine? The key is GNU Privacy Guard (GPG), which is the open-source implementation of Pretty Good Privacy (PGP). If an RPM file is signed using a private GPG key, the integrity of the package can be verified with the corresponding public GPG key. A valid signature also ensures that the package has been signed by an authorized party and does not come from a malicious hacker.

If you haven't imported or installed the Red Hat public GPG keys, you might have noticed something similar to the following message when packages are installed:

```
warning: vsftpd-3.0.2-9.el7.x86_64.rpm: Header V3 RSA/SHA256
Signature, key ID fd431d51: NOKEY
```

If you're concerned about security, this warning should raise alarm bells. During the RHEL 7 installation process, GPG keys are stored in the /etc/pki/rpm-gpg directory. Take a look at the contents of this directory. You'll find files such as RPM-GPG-KEY-redhat-release. To actually use the key to verify packages, it has to be imported—and the command to import the GPG key is fairly simple:

```
# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

If there's no output, the **rpm** command probably successfully imported the GPG key. Even if this command succeeds, if you repeat it, an "import failed" message will appear. In addition, the GPG key is now included in the RPM database, which can be verified with the **rpm -qa gpg-pubkey** command.

In the /etc/pki/rpm-gpg directory, there are normally five GPG keys available, as described in Table 7-1.

Later in this chapter, you'll see how GPG keys are imported automatically from remote repositories when new packages are installed.

Special RPM Procedures with the Kernel

Updated kernels incorporate new features, address security issues, and generally help Linux systems work better. However, kernel updates can go wrong and prevent systems booting or cause applications to break; this is particularly common if specialized packages that depend on an existing version of a kernel have been installed.

If you are aware of any software that relies on a custom kernel module, don't upgrade a kernel if you're not ready to repeat every step taken to customize software with the existing kernel, whether that be obtaining new closed-source kernel modules from the vendor for the new version, rebuilding specialized modules for the new kernel, or other manual work. For example, the drivers for a few wireless network cards and printers

TABLE 7-1 GPG Keys to Verify Software Updates

GPG Key	Description
RPM-GPG-KEY-redhat-beta	Packages built for the RHEL 7 beta
RPM-GPG-KEY-redhat-legacy-former	Packages for pre–January 2007 releases (and updates)
RPM-GPG-KEY-redhat-legacy-release	Packages for post–January 2007 releases
RPM-GPG-KEY-redhat-legacy-rhx	Packages associated with Red Hat Exchange
RPM-GPG-KEY-redhat-release	Released packages for RHEL 7

without in-tree open-source drivers may be tied to a specific version of a kernel. Some virtual machine software components (not including KVM) may be installed against a specific version of a kernel.

If you see an available update for a kernel RPM, the temptation is to run the **rpm -U *newkernel*** command. Don't do it! It overwrites your existing kernel, and if the updated kernel doesn't work with the system, you're out of luck. (Well, not completely out of luck, but if you reboot and have problems, you'll have to use rescue mode, discussed in Chapter 5, to boot the system and reinstall the existing kernel. In the days where there were separate Troubleshooting and System Maintenance sections on the Red Hat exams, that might have made for an interesting test scenario.) The best option for upgrading to a new kernel is to install it, specifically with a command such as this:

```
# rpm -ivh newkernel
```

If you're connected to an appropriate repository, the following command works equally well:

```
# yum install kernel
```

This installs the new kernel, along with related files, side by side with the current working kernel. One example of the result is shown in Figure 7-1, in the output to the **ls /boot** command.

on the job

It is also safe to install a new kernel by running yum update kernel**. In fact, by default, yum is configured to always install a kernel package and leave any old kernel in place. This applies to a maximum of three kernels installed at the same time.**

Table 7-2 briefly describes the different files for the various parts of the boot process in the /boot directory.

FIGURE 7-1

New and existing kernel files in the /boot directory

```
[root@server1 ~]# ls /boot
config-3.10.0-123.13.2.el7.x86_64
config-3.10.0-123.el7.x86_64
grub2
initramfs-0-rescue-b37be8dd26f97ac4ba4a6152f5e92b44.img
initramfs-3.10.0-123.13.2.el7.x86_64.img
initramfs-3.10.0-123.el7.x86_64.img
initrd-plymouth.img
symvers-3.10.0-123.13.2.el7.x86_64.gz
symvers-3.10.0-123.el7.x86_64.gz
System.map-3.10.0-123.13.2.el7.x86_64
System.map-3.10.0-123.el7.x86_64
vmlinuz-0-rescue-b37be8dd26f97ac4ba4a6152f5e92b44
vmlinuz-3.10.0-123.13.2.el7.x86_64
vmlinuz-3.10.0-123.el7.x86_64
[root@server1 ~]#
```

The installation of a new kernel adds options to boot the new kernel in the GRUB configuration file (/boot/grub2/grub.cfg), without erasing existing options. A condensed version of the revised GRUB configuration file is shown in Figure 7-2.

A careful reading of the two "menuentry" stanzas reveals that the only difference is in the version numbers listed in the title for the Linux kernel and for the initial RAM disk filesystem. By default, the system will boot with the newly installed kernel. Therefore, if that kernel does not work, you can restart the system, access the GRUB menu, and then boot from the older, previously working kernel.

TABLE 7-2

Files in the /boot Directory

File	Description
config-*	Kernel configuration settings; a text file
grub2/	Directory with GRUB configuration files
initramfs-*	The initial RAM disk filesystem, a root filesystem called during the boot process to help load other kernel components, such as block device modules
initrd-plymouth.img	RAM disk filesystem containing files for the graphical animation displayed at boot by Plymouth
symvers-*	List of modules
System.map-*	Map of system names for variables and functions, with their locations in memory
vmlinuz-*	The actual Linux kernel

FIGURE 7-2 GRUB with a second kernel

```
menuentry 'Red Hat Enterprise Linux Server (3.10.0-123.13.2.el7.x86_64) 7.0 (Maipo)' --class
red --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.
10.0-123.el7.x86_64-advanced-d055418f-1ff6-46bf-8476-b391e82a6f51' {
        # output removed for brevity
        set root='hd0,msdos1'
        linux16 /vmlinuz-3.10.0-123.13.2.el7.x86_64 root=/dev/mapper/rhel-root ro rd.lvm.lv=r
hel/root vconsole.font=latarcyrheb-sun16 rd.lvm.lv=rhel/swap crashkernel=auto  vconsole.keyma
p=uk rhgb quiet LANG=en_GB.UTF-8
        initrd16 /initramfs-3.10.0-123.13.2.el7.x86_64.img
}
menuentry 'Red Hat Enterprise Linux Server (3.10.0-123.el7.x86_64) 7.0 (Maipo)' --class red -
-class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-
123.el7.x86_64-advanced-d055418f-1ff6-46bf-8476-b391e82a6f51' {
        # output removed for brevity
        set root='hd0,msdos1'
        linux16 /vmlinuz-3.10.0-123.el7.x86_64 root=/dev/mapper/rhel-root ro rd.lvm.lv=rhel/r
oot vconsole.font=latarcyrheb-sun16 rd.lvm.lv=rhel/swap crashkernel=auto  vconsole.keymap=uk
rhgb quiet LANG=en_GB.UTF-8
        initrd16 /initramfs-3.10.0-123.el7.x86_64.img
}
```

CERTIFICATION OBJECTIVE 7.02

More RPM Commands

The **rpm** command is rich with details. All this book can do is cover some of the basic ways **rpm** can help you manage RHEL. You've already read about how **rpm** can install and upgrade packages in various ways. Queries can help you identify what's installed in detail. Verification tools can help you check the integrity of packages and individual files. You can use related tools to help identify the purpose of different RPMs, as well as a full list of those RPMs already installed.

Package Queries

The simplest RPM query verifies whether a specific package is installed. The following command verifies the installation of the systemd package (the version number may vary):

```
# rpm -q systemd
systemd-208-11.el7.x86_64
```

You can do more with RPM queries, as described in Table 7-3. Note how queries are associated with **-q** or ***--query***; full-word switches such as ***--query*** are usually associated with a double-dash.

TABLE 7-3

rpm --query Options

rpm Query Command	Meaning
rpm -qa	Lists all installed packages.
rpm -qf /path/to/file	Identifies the package associated with /path/to/file.
rpm -qc *packagename*	Lists only configuration files from *packagename.*
rpm -qd *packagename*	Lists only documentation files from *packagename.*
rpm -qi *packagename*	Displays basic information for *packagename.*
rpm -ql *packagename*	Lists all files from *packagename.*
rpm -qR *packagename*	Notes all dependencies; you can't install *packagename* without them.
rpm -q --changelog *packagename*	Displays change information for *packagename.*

If you want to query an RPM package file rather than the local RPM database, all you have to do is add the **-p** switch and specify the path or URL of the package file. As an example, the following command lists all the files of the RPM package epel-release-7-5.noarch.rpm:

```
# rpm -qlp epel-release-7-5.noarch.rpm
```

Package Signatures

RPM uses several methods for checking the integrity of a package. You've seen how to import the GPG key. Some of the available methods are shown when you verify a package with the **rpm --checksig *pkg.rpm*** command. (The **-K** switch is equivalent to **--checksig**.) For example, if you've downloaded a package from a third party such as the hypothetical pkg-1.2.3-4.noarch.rpm package and want to check it against the imported GPG keys, run the following command:

```
# rpm --checksig pkg-1.2.3-4.noarch.rpm
```

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

```
pkg-1.2.3-4.noarch.rpm: rsa sha1 (md5) pgp md5 OK
```

This guarantees that the package is authentic and the RPM file was not modified by a third party. You may already recognize the algorithms used to verify package integrity:

- **rsa** Named for its creators, Rivest, Shamir, and Adleman, it's a public key encryption algorithm.
- **sha1** A 160-bit message digest Secure Hash Algorithm; a cryptographic hash function.
- **md5** Message Digest 5, a cryptographic hash function.
- **pgp** PGP, as implemented in Linux by GPG.

File Verification

The verification of an installed package compares information about that package with information from the RPM database on a system. The ***--verify*** (or ***-v***) switch checks the size, MD5 checksum, permissions, type, owner, and group of each file in the package. Verification can be done in a number of ways. Here are a few examples:

- Verify all files. Naturally, this may take a long time on your system. (Of course, the **rpm -Va** command performs the same function.)

```
# rpm --verify –a
```

- Verify all files within a package against a downloaded RPM.

```
# rpm --verify -p vsftpd-3.0.2-9.el7.x86_64.rpm
```

- Verify a file associated with a particular package.

```
# rpm --verify --file /bin/ls
```

If the integrity of the files or packages is verified, you will see no output. Any output means that a file or package is different from the original. There's no need to panic if certain changes appear; after all, administrators do edit configuration files. There are eight tests. If there has been a change, the output is a string of up to eight failure code characters, each of which tells you what happened during each test.

If you see a dot (.), that test passed. The following example shows /bin/vi with an incorrect group ID assignment:

```
# rpm --verify --file /bin/vi
......G.   /bin/vi
```

Table 7-4 lists the failure codes and their meanings.

Now here's an interesting experiment: When you have one version of a package installed, use the **rpm *--verify* -p** command with a second version of the same package. Finding such

TABLE 7-4

rpm --verify Codes

Failure Code	Meaning
5	MD5 checksum
S	File size
L	Symbolic link
T	File modification time
D	Device
U	User
G	Group
M	Mode

a package should not be too difficult because Red Hat updates packages for feature updates, security patches, and, yes, bug fixes frequently. For example, when we wrote this book for RHEL 7, we had access to both sssd-client-1.11.2-65.el7.x86_64.rpm and sssd-client-1.11.2-28 .el7.x86_64.rpm. When the latter version was installed, we ran the command

```
# rpm --verify -p sssd-client-1.11.2-65.el7.x86_64.rpm
```

and got a whole list of changed files, as shown in Figure 7-3. This command provides information on what was changed between different versions of the sssd-client package.

FIGURE 7-3

Verifying changes between packages

```
[root@server1 Packages]# rpm --verify -p sssd-client-1.11.2-65.el7.x86_64.rpm
S.5....T.    /usr/lib64/krb5/plugins/authdata/sssd_pac_plugin.so
S.5....T.    /usr/lib64/krb5/plugins/libkrb5/sssd_krb5_locator_plugin.so
S.5....T.    /usr/lib64/libnss_sss.so.2
S.5....T.    /usr/lib64/security/pam_sss.so
missing     /usr/share/doc/sssd-client-1.11.2
missing   d /usr/share/doc/sssd-client-1.11.2/COPYING
missing   d /usr/share/doc/sssd-client-1.11.2/COPYING.LESSER
missing   d /usr/share/man/ca/man8/pam_sss.8.gz
missing   d /usr/share/man/es/man8/pam_sss.8.gz
S.5....T.  d /usr/share/man/es/man8/sssd_krb5_locator_plugin.8.gz
S.5....T.  d /usr/share/man/fr/man8/pam_sss.8.gz
S.5....T.  d /usr/share/man/fr/man8/sssd_krb5_locator_plugin.8.gz
missing   d /usr/share/man/ja/man8/pam_sss.8.gz
S.5....T.  d /usr/share/man/ja/man8/sssd_krb5_locator_plugin.8.gz
S.5....T.  d /usr/share/man/man8/pam_sss.8.gz
S.5....T.  d /usr/share/man/man8/sssd_krb5_locator_plugin.8.gz
S.5....T.  d /usr/share/man/uk/man8/pam_sss.8.gz
S.5....T.  d /usr/share/man/uk/man8/sssd_krb5_locator_plugin.8.gz
[root@server1 Packages]#
```

CERTIFICATION OBJECTIVE 7.03

Dependencies and the yum Command

The **yum** command makes it easy to add and remove software packages to and from a system. It maintains a database regarding the proper way to add, upgrade, and remove packages. This makes it relatively simple to add and remove software with a single command. That single command overcame what was known as "dependency hell."

The **yum** command was originally developed for Yellow Dog Linux. The name is based on the Yellow Dog updater, modified. Given the trouble associated with dependency hell, Linux users were motivated to find a solution. It was adapted for Red Hat distributions with the help of developers from Duke University.

The configuration of **yum** depends on package libraries known as repositories. Red Hat repositories are available through the Red Hat Portal, while repositories of third-party rebuild distributions use their own publicly available servers. In either case, it's important to know the workings of the **yum** command as well as how it installs and updates individual packages and package groups.

An Example of Dependency Hell

To understand more about the need for the **yum** command, examine Figure 7-4. You do not need the kernel.spec file. The packages listed in that figure are what's required to build an RPM. Although the building of an RPM package is not an exam requirement, the associated packages illustrate the need for **yum**.

You could try to use the **rpm** command to install each of these packages. To do so, take the following steps:

1. Include the RHEL 7 DVD. Insert it into its drive, or make sure it's included in the configuration for the target virtual machine.

FIGURE 7-4

Packages required to build RPMs

```
[root@server1 SPECS]# rpmbuild -ba kernel.spec
error: Failed build dependencies:
        gcc >= 3.4.2 is needed by kernel-3.10.0-123.13.2.el7.x86_64
        xmlto is needed by kernel-3.10.0-123.13.2.el7.x86_64
        hmaccalc is needed by kernel-3.10.0-123.13.2.el7.x86_64
        elfutils-devel is needed by kernel-3.10.0-123.13.2.el7.x86_64
        binutils-devel is needed by kernel-3.10.0-123.13.2.el7.x86_64
        python-devel is needed by kernel-3.10.0-123.13.2.el7.x86_64
        perl(ExtUtils::Embed) is needed by kernel-3.10.0-123.13.2.el7.x86_64
        bison is needed by kernel-3.10.0-123.13.2.el7.x86_64
        audit-libs-devel is needed by kernel-3.10.0-123.13.2.el7.x86_64
        numactl-devel is needed by kernel-3.10.0-123.13.2.el7.x86_64
[root@server1 SPECS]#
```

2. Unless it's already mounted, mount that DVD with the following command. Of course, a different empty directory can be substituted for /media.

   ```
   # mount /dev/cdrom /media
   ```

3. Navigate to the directory where the DVD is mounted (that is, /media or some subdirectory of /media).
4. The RPM packages on the RHEL 7 DVD can be found in the Packages/ subdirectory of the DVD. Navigate to that subdirectory.
5. Enter the **rpm -ivh** command, and then type in the names of the packages listed in Figure 7-4. It may be easiest to use command completion for this purpose; for example, if you were to type in

   ```
   # rpm -ivh gcc-
   ```

   you could then press the TAB key twice and review available packages that start with **gcc-**. You could then enter additional keys and press the TAB key again to complete the name of the package. After a bit of work, you'd end up with something similar to the command and results shown in Figure 7-5. What actually appears depends on the current revision level of each package, as well as what's already installed on the local system.
6. The next step is to try to include the missing dependencies in the list of packages to be installed. When we try this step, it leads to more dependencies, as shown in Figure 7-6.

At this point, the addition of more packages to the installation becomes somewhat more difficult. How would you know, except from experience, that the mpfr-* package would

FIGURE 7-5 These packages have dependencies.

```
[root@server1 Packages]# rpm -ivh gcc-4.8.2-16.el7.x86_64.rpm xmlto-0.0.25-7.el7.x86_64.rpm
hmaccalc-0.9.13-4.el7.x86_64.rpm elfutils-devel-0.158-3.el7.x86_64.rpm binutils-devel-2.23.5
2.0.1-16.el7.x86_64.rpm python-devel-2.7.5-16.el7.x86_64.rpm perl-ExtUtils-Embed-1.30-283.el
7.noarch.rpm bison-2.7-4.el7.x86_64.rpm audit-libs-devel-2.3.3-4.el7.x86_64.rpm numactl-deve
l-2.0.9-2.el7.x86_64.rpm
error: Failed dependencies:
        cpp = 4.8.2-16.el7 is needed by gcc-4.8.2-16.el7.x86_64
        glibc-devel >= 2.2.90-12 is needed by gcc-4.8.2-16.el7.x86_64
        libmpc.so.3()(64bit) is needed by gcc-4.8.2-16.el7.x86_64
        libmpfr.so.4()(64bit) is needed by gcc-4.8.2-16.el7.x86_64
        flex is needed by xmlto-0.0.25-7.el7.x86_64
        elfutils-libelf-devel(x86-64) = 0.158-3.el7 is needed by elfutils-devel-0.158-3.el7.
x86_64
        perl-devel is needed by perl-ExtUtils-Embed-0:1.30-283.el7.noarch
        kernel-headers >= 2.6.29 is needed by audit-libs-devel-2.3.3-4.el7.x86_64
[root@server1 Packages]#
```

FIGURE 7-6 There are even more dependencies.

```
[root@server1 Packages]# rpm -ivh gcc-4.8.2-16.el7.x86_64.rpm xmlto-0.0.25-7.el7.x86_64.rpm
hmaccalc-0.9.13-4.el7.x86_64.rpm elfutils-devel-0.158-3.el7.x86_64.rpm binutils-devel-2.23.5
2.0.1-16.el7.x86_64.rpm python-devel-2.7.5-16.el7.x86_64.rpm perl-ExtUtils-Embed-1.30-283.el
7.noarch.rpm bison-2.7-4.el7.x86_64.rpm audit-libs-devel-2.3.3-4.el7.x86_64.rpm numactl-deve
l-2.0.9-2.el7.x86_64.rpm cpp-4.8.2-16.el7.x86_64.rpm glibc-devel-2.17-55.el7.x86_64.rpm libm
pc-1.0.1-3.el7.x86_64.rpm mpfr-3.1.1-4.el7.x86_64.rpm flex-2.5.37-3.el7.x86_64.rpm elfutils-
libelf-devel-0.158-3.el7.x86_64.rpm perl-devel-5.16.3-283.el7.x86_64.rpm kernel-headers-3.10
.0-123.el7.x86_64.rpm
error: Failed dependencies:
        glibc-headers is needed by glibc-devel-2.17-55.el7.x86_64
        glibc-headers = 2.17-55.el7 is needed by glibc-devel-2.17-55.el7.x86_64
        gdbm-devel is needed by perl-devel-4:5.16.3-283.el7.x86_64
        libdb-devel is needed by perl-devel-4:5.16.3-283.el7.x86_64
        perl(ExtUtils::Installed) is needed by perl-devel-4:5.16.3-283.el7.x86_64
        perl(ExtUtils::MakeMaker) is needed by perl-devel-4:5.16.3-283.el7.x86_64
        perl(ExtUtils::ParseXS) is needed by perl-devel-4:5.16.3-283.el7.x86_64
        systemtap-sdt-devel is needed by perl-devel-4:5.16.3-283.el7.x86_64
[root@server1 Packages]#
```

satisfy the "Failed Dependencies" message for libmpfr.so.4()(64bit)? Even if you do already understand, the inclusion of such packages is not enough. There's even one more level of dependent packages. This pain is known as dependency hell.

Relief from Dependency Hell

Before **yum**, some attempts to use the **rpm** command were stopped by the dependencies described earlier. Sure, you could install those dependent packages with the same command, but what if those dependencies themselves have dependencies? That perhaps is the biggest advantage of the **yum** command.

Before **yum**, RHEL incorporated dependency resolution into the update process. Through RHEL 4, this was done with **up2date**. Red Hat incorporated **yum** starting with RHEL 5. The **yum** command uses subscribed Red Hat Portal channels and any other repositories configured in the /etc/yum.repos.d directory.

All you need to do to install the packages listed in Figure 7-4 is run the following command:

```
# yum install gcc xmlto hmaccalc elfutils-devel binutils-devel \
> python-devel perl-ExtUtils-Embed bison audit-libs-devel numactl-devel
```

If so prompted, accept the request to install additional dependent packages, and then all of the noted dependencies are installed automatically. (Yes, the **-y** switch would perform the same function.) If updates are available from connected repositories, the latest available

version of each package is installed. The **yum** command is described in more detail later in this chapter.

But if you're running RHEL 7 without a connection to Red Hat Portal, nothing happens. Shortly, you'll see how to create a connection between **yum** and the installation server created in Chapter 1.

A number of third-party repositories are available for RHEL. They include several popular applications that are not supported by Red Hat. For example, one of the authors of this book uses an external repository to install packages associated with his laptop wireless network card.

Although the owners of these repositories work closely with some Red Hat developers, there are some reports where dependencies required from one repository are unavailable from other repositories, leading to a different form of "dependency hell." However, the more popular third-party repositories are excellent; we have never encountered "dependency hell" when using these repositories.

There are two main reasons why Red Hat does not include most proven and popular packages available from third-party repositories. Some are not released under open-source licenses, and others are packages that Red Hat simply chooses not to support.

Basic yum Configuration

Relief from dependency hell depends on the proper configuration of **yum**. Not only do you need to know how to configure **yum** to connect to repositories over the Internet, but also you need to know how to configure **yum** to connect to repositories on a local network. With this knowledge, you can connect **yum** to repositories on Red Hat Portal, to repositories configured by third parties, and to custom repositories configured for specialized networks. And remember, during the Red Hat exams, you won't have access to the Internet.

To that end, you have to understand how yum is configured in some detail. It starts with the /etc/yum.conf configuration file and continues with files in the /etc/yum and /etc/yum .repos.d directories. To get the full list of yum configuration directives and their current values, run the following command:

```
# yum-config-manager
```

This command requires the installation of the yum-utils package.

The Basic yum Configuration File: yum.conf

This section analyzes the default version of the /etc/yum.conf file, line by line. Although you won't make changes to this file in most cases, you need to understand at least the standard directives in this file if something goes wrong. The following lines are straight excerpts from the default version of this file. The first directive is a header; the **[main]** header suggests that all directives that follow apply globally to **yum**:

```
[main]
```

The **cachedir** directive specifies where caches of packages, package lists, and related databases are to be downloaded. Based on the standard 64-bit architecture for RHEL 7, this translates to the /var/cache/yum/x86_64/7Server directory.

```
cachedir=/var/cache/yum/$basearch/$releasever
```

The **keepcache** boolean directive specifies whether **yum** actually stores downloaded headers and packages in the directory specified by **cachedir**. The standard shown here suggests that caches are not kept, which helps make sure that a system is kept up to date with the latest available packages (at the expense of slightly slower executions of **yum** as metadata is pulled down on each execution).

```
keepcache=0
```

The **debuglevel** directive is closely related to the **errorlevel** and **logfile** directives, as they specify the detail associated with debug and error messages. Even though the **errorlevel** directive is not shown, both it and **debuglevel** are set to 2 by default. The available range is 0–10, where 0 provides almost no information, and 10 provides perhaps too much information, even for developers.

```
debuglevel=2
logfile=/var/log/yum.log
```

The **exactarch** boolean directive makes sure the architecture matches the actual processor type, as defined by the **arch** command.

```
exactarch=1
```

The **obsoletes** boolean directive can support the uninstallation of obsolete packages in conjunction with a **yum update** command.

```
obsoletes=1
```

The **gpgcheck** boolean directive makes sure the **yum** command actually checks the GPG signature of downloaded packages.

```
gpgcheck=1
```

The **plugins** boolean directive provides a necessary link to Python-based RHN plugins in the /usr/share/yum-plugins directory. It also refers indirectly to plugin configuration files in the /etc/yum/pluginconf.d directory.

```
plugins=1
```

The **installonly_limit** directive specifies how many of the packages listed in the **installonlypkgs** option (usually the kernel) can be installed at the same time:

```
installonly_limit=3
```

To make sure the header data downloaded from the RHN (and any other repositories) is up to date, the **metadata_expire** directive specifies a lifetime for headers. Although the comments in yum.conf state that the default value is 90 minutes, the actual default on RHEL 7 is six hours. In other words, if you haven't used the **yum** command in the last six hours, the next use of the **yum** command downloads the latest header information.

```
#metadata_expire=90m
```

The final directive of interest, in comments, happens to be the default; it's a reference to the noted directory for actual configuration information for repositories:

```
# PUT YOUR REPOS HERE OR IN separate files named file.repo
# in /etc/yum.repos.d
```

Configuration Files in the /etc/yum/pluginconf.d Directory

The default files in the /etc/yum/pluginconf.d directory configure a connection between **yum** and the Red Hat Portal or a local Satellite server. If you're studying from a RHEL rebuild distribution such as CentOS, you'll see a different set of files in this directory. In CentOS, the files in this directory are focused on connecting the local system to better repositories over the Internet. This is a Red Hat book, however, so the focus will be on the two basic files in the RHEL 7 installation.

Red Hat Network Plugins

If you have a subscription to the RHN via an old version of Red Hat Satellite Server, the rhnplugin.conf file in this directory is especially important. Although the directives, shown next, may seem simple, they enable access and check GPG signatures:

```
[main]
enabled = 1
gpgcheck = 1
timeout = 120
```

In comments, this file suggests that different settings can be configured for different repositories. The repositories enclosed in brackets should match those associated with the actual RHN repositories.

Red Hat Subscription Management Plugins

The subscription-manager.conf and product-id.conf files are designed to connect the **yum** system to Red Hat Portal using Subscription Manager. As discussed later in this chapter, Subscription Manager is a system designed to replace RHN for system updates. The file subscription-manager.conf is very simple, with two directives that enable a connection between **yum** and the Subscription Manager plugin:

```
[main]
enabled=1
```

Configuration Files in the /etc/yum.repos.d Directory

The configuration files in the /etc/yum.repos.d directory are designed to connect systems to actual repositories. If you're running a rebuild distribution such as CentOS, you'll see files that connect to public repositories on the Internet. If you're running RHEL 7, this directory may be empty, unless the system was registered with Red Hat Subscription Management. In that case, you'll see a redhat.repo file in that directory, which is designed to get further updates from the Red Hat Portal.

A couple of elements in common for configuration files in the /etc/yum.repos.d directory are the file extension (.repo) and the documentation, available with the **man yum.conf** command.

A properly configured .repo file in the /etc/yum.repos.d directory can be a terrific convenience to enable the installation of groups of packages with the **yum** command. As the /etc/yum.repos.d directory may be empty on a RHEL 7 system, you should know how to create that file from scratch, using data for the installation server and information available in the yum.conf man page.

Understand /etc/yum.repos.d Configuration Files for Rebuild Distributions

If you're running a rebuild distribution, the files in the /etc/yum.repos.d directory may connect the local system to one or more remote repositories. One example comes from CentOS 7, as shown in Figure 7-7. Although it includes a number of different repositories, you can learn from the pattern of directives configured for each repository.

FIGURE 7-7 Several repositories configured in one file

```
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=updates
#baseurl=http://mirror.centos.org/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=extras
#baseurl=http://mirror.centos.org/centos/$releasever/extras/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that extend functionality of existing packages
[centosplus]
name=CentOS-$releasever - Plus
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=centosplus
#baseurl=http://mirror.centos.org/centos/$releasever/centosplus/$basearch/
gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

There are four stanzas in Figure 7-7. Each stanza represents a connection to a CentOS repository. For example, the first stanza includes the basic elements of a repository and more. The first line, in brackets, provides a name for the repository. In this case, **[base]** just stands for the base repository used by the CentOS 7 distribution. It doesn't represent the directory where the associated packages are installed.

```
[base]
```

However, when you run the **yum update** command to update the local database of those remote packages, it includes **base** as the name of the repository in output similar to the following, which suggests that it took one second to download the 3.6KB database of existing repository data:

```
base            | 3.6 kB  00:00:01
```

Although the name of the repository follows, it's just for documentation purposes and does not affect how packages or package databases are read or downloaded. However, the inclusion of the **name** directive does avoid a nonfatal error message.

```
name=CentOS-$releasever - Base
```

Note the **mirrorlist** directive that follows. It specifies a URL to a file that contains a list of multiple URLs to the closest remote servers with a copy of the actual repository of packages. It commonly works with either the HTTP or FTP protocol. (It can even work with local directories or mounted Network File System shares, as described in Exercise 7-1.)

```
mirrorlist=http://mirrorlist.centos.org/?release=$releasever ↵
&arch=$basearch&repo=os
```

Alternatively, these repositories can be set up in a file downloaded with the **baseurl** directive:

```
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
```

Repositories configured in .repo files in the /etc/yum.repos.d directory are **enabled** by default. The following directive provides an easy way to deactivate a connection to such (**enabled=1** would activate the connection):

```
enabled=0
```

If you want to disable the GPG signatures of each package to be downloaded, the following command puts that wish into effect:

```
gpgcheck=0
```

Of course, if you enable **gpgcheck**, any GPG check requires a GPG key; the following directive specifies one key from the local /etc/pki/rpm-gpg directory for that purpose:

```
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

Create Your Own /etc/yum.repos.d Configuration File

You'll want to know how to create a local configuration file in the /etc/yum.repos.d directory. It enables the use of the **yum** command, which is the easiest way to install groups of packages such as the Apache web server or any of the groups of packages discussed in the book.

To do so, you'll need to set up a text file with a .repo extension in the /etc/yum.repos.d directory. All that file needs is three lines. In fact, if you're willing to accept some nonfatal errors, two lines are sufficient.

On RHEL 7, especially during an exam, the /etc/yum.repos.d directory may be empty. So you may not have access to examples such as those available for CentOS, as shown in Figure 7-7. The first guidance comes from the following comments at the bottom of the /etc/yum.conf file, which confirm that the file must have a .repo extension in the /etc/yum.repos.d directory:

```
# PUT YOUR REPOS HERE OR IN separate files named file.repo
# in /etc/yum.repos.d
```

FIGURE 7-8

Excerpt from man yum.conf to configure a new repository

```
[repository] OPTIONS
       The repository section(s) take the following form:

              Example: [repositoryid]
              name=Some name for this repository
              baseurl=url://path/to/repository/

              repositoryid  Must  be  a  unique  name for each repository, one
              word.

              name A human readable string describing the repository.

              baseurl Must be a URL to the directory  where  the  yum  reposi-
              tory's  `repodata' directory lives. Can be an http://, ftp:// or
              file:// URL. You can specify multiple URLs in one baseurl state-
              ment. The best way to do this is like this:
              [repositoryid]
              name=Some name for this repository
              baseurl=url://server1/path/to/repository/
                      url://server2/path/to/repository/
                      url://server3/path/to/repository/

              If you list more than one baseurl= statement in a repository you
              will find yum will ignore the  earlier  ones  and  probably  act
              bizarrely. Don't do this, you've been warned.
```

In addition, you could configure the three lines in the /etc/yum.conf file. If you forget what three lines to add, there is an example in the man page for the yum.conf file, as shown in Figure 7-8.

If you forget what to do, run the **man yum.conf** command and scroll down to this part of the man page. The identifier for the repository is shown in brackets. Unless specified by the RHCSA exam, it doesn't matter what single word you put between the brackets as the identifier.

For the purpose of this chapter, we open a new file named whatever.repo in the /etc/yum .repos.d directory. (To some extent, the filename of the .repo file does not matter, as long as it has a .repo extension in the /etc/yum.repos.d directory.) In that file, we add the following identifier:

```
[test]
```

exam Watch

You should learn how to create a working .repo file in the /etc/yum .repos.d directory during Red Hat exams. It can be a big time saver when you need to install additional packages.

Next comes the **name** directive for the repository. As suggested by the listing in the man page, that name should be "human readable." In Linux parlance, that also means the name does not affect the functionality of the repository. To demonstrate, we add the following directive:

```
name=somebody likes Linux
```

Finally, there's the **baseurl** directive, which can be configured to point to an installation server. The RHCSA requirements imply that you need to know how to install Linux from a remote server. They also suggest that you need to know how to install and update packages from a remote repository. To meet either objective, you need to know the URL of that remote server or repository. It's reasonable to expect that URL to be provided during the exam. In Chapter 1, you created FTP and HTTP installation servers on the host system for virtual machines, which are "remote" from those systems.

The FTP and HTTP installation servers that you created in Chapter 1 can also be used as remote repositories. To set up access to those repositories, all you need to include is one of the following **baseurl** directives:

```
baseurl=ftp://192.168.122.1/pub/inst
baseurl=http://192.168.122.1/inst
```

As suggested in the yum.conf man page, you should not include both URLs in separate **baseurl** directives. Make a choice and save the resulting file. That's all you need. There's no reason (except for better security) to include the **enabled**, **gpgcheck**, or **gpgkey** directive described earlier. Of course, security is important in real life, but if your focus is on the exam, the best advice is often to keep things as simple as possible.

Once the file is saved, run the following commands to first clear out databases from previously accessed repositories and then to update the local database cache from the repository newly configured in the /etc/yum.repos.d/*whatever*.repo file:

```
# yum clean all
# yum makecache
```

For a system not registered with Red Hat Subscription Management, it'll lead to the following output:

```
Loaded plugins: langpacks, product-id, subscription-manager
test                                          | 3.7 kB     00:00
test/primary_db                               | 2.9 MB     00:00
Metadata Cache Created
```

The system is now ready for the installation of new packages. Try running the following command:

```
# yum install system-config-date
```

Given the virtual machines configured earlier in this book, you might see the result shown in Figure 7-9. If confirmed, the **yum** command would download and then install not only the system-config-date RPM, but also the dependent package shown in the figure to make sure the system-config-date package is fully supported.

FIGURE 7-9 Installation of one package can include dependencies.

```
Dependencies Resolved

================================================================================
 Package                    Arch          Version              Repository    Size
================================================================================
Installing:
 system-config-date         noarch        1.10.6-2.el7         test         619 k
Installing for dependencies:
 system-config-date-docs    noarch        1.0.11-4.el7         test         527 k

Transaction Summary
================================================================================
Install  1 Package (+1 Dependent package)

Total download size: 1.1 M
Installed size: 3.5 M
Is this ok [y/d/N]:
```

EXERCISE 7-1

Create a yum Repository from the RHEL 7 DVD

This exercise requires access to the RHEL 7 DVD. If you don't have a lot of space for this exercise, it's acceptable to set up the repository directly on the mounted DVD. Alternatively, you can copy the contents to a specified directory. It also assumes an available installation repository, such as one of those created in Chapter 1.

This exercise assumes you'll be starting with no files in the /etc/yum.repos.d directory described in this chapter.

1. If there are existing files in the /etc/yum.repos.d directory, copy them to a backup location such as the root user's home directory, /root. Delete any existing .repo files in the /etc/yum.repos.d directory.

   ```
   # cp  -a /etc/yum.repos.d /root/
   # rm –f /etc/yum.repos.d/*.repo
   ```

2. Mount the RHEL 7 DVD on the /mnt directory with the following command (you may need to substitute /dev/sr0 or /dev/dvd for /dev/cdrom):

   ```
   # mount /dev/cdrom /mnt
   ```

   Alternatively, if you have only the RHEL 7 DVD as an ISO file, mount it with the following command:

   ```
   # mount -o loop rhel-server-7.0-x86_64-dvd.iso /mnt
   ```

Of course, if desired, you can copy the files from a different mount point, such as from /mnt to the /opt/repos/rhel7 directory, with a command like this:

```
# mkdir -p /opt/repos/rhel7
# cp -a /mnt/. /opt/repos/rhel7
```

The dot (.) in front of the /mnt directory ensures the copying of the contents of the directory, rather than of the directory itself.

3. Navigate to the /etc/yum.repos.d directory.
4. Open a new file in a text editor. Use a name such as rhel7.repo.
5. Edit the rhel7.repo file. Create a new stanza of directives. Use an appropriate stanza title such as **[rhel]**.
6. Specify an appropriate **name** directive for the repository.
7. Include a **baseurl** directive set to **file:///opt/repos/rhel7/**. Include an **enabled=1** directive.
8. Save and close the file.
9. Assuming you're running RHEL 7 (and not a rebuild distribution), open the subscription-manager.conf file in the /etc/yum/pluginconf.d directory and set **enabled=0**.
10. Run the **yum clean all** and **yum update** commands.
11. If successful, you should see the following output:

```
Loaded plugins: langpacks, product-id
rhel                                          | 3.8 kB  00:00:00
(1/2): rhel/group_gz                          | 133 kB  00:00:00
(2/2): rhel/primary_db                        | 3.4 MB  00:00:00
No packages marked for update
```

You've now set up a repository on the local /opt/repos/rhel7 directory.

12. Restore the original files. Open the subscription-manager.conf file in the /etc/yum/pluginconf.d directory and then set **enabled=1**. Move the backed-up files from the /root directory to /etc/yum.repos.d. If you want to restore the original configuration, delete or move the rhel7.repo file from that directory. Run the **yum clean all** command again.

Third-party Repositories

Other groups of third-party developers create packages for RHEL 7. They include packages for some popular software not supported by Red Hat. The websites for two of these third parties can be found at https://fedoraproject.org/wiki/EPEL and http://repoforge.org.

To add third-party repositories to a system, you'd create a custom .repo file in the /etc/yum.repos.d directory.

Some repositories, such as EPEL (Extra Packages for Enterprise Linux), simplify the configuration by providing an RPM package that includes a .repo configuration file and a GPG key to verify the packages. To configure the repository, all you have to do is install that RPM file:

```
# rpm -ivh https://dl.fedoraproject.org/pub/epel/7/x86_64/e/ ↵
epel-release-7-5.noarch.rpm
```

If you want to disable any repository in the /etc/yum.repos.d directory, add the following directive to the applicable repository file:

```
enabled=0
```

Basic yum Commands

If you want to learn more about the intricacies of the **yum** command, run the command by itself. You'll see the following output scroll by, probably far too fast. Of course, you can pipe the output to the **less** command pager with the **yum | less** command.

```
# yum
Loaded plugins: langpacks, product-id, subscription-manager
You need to give some command
usage: yum [options] COMMAND

List of Commands
...
```

You'll examine how a few of these commands and options work in the following sections. Although you won't have Internet access during a Red Hat exam, you might have a network connection to a locally configured repository, which you should be ready to configure via the appropriate file in the /etc/yum.repos.d directory, as described earlier. As well as during the exam, yum is an excellent tool for administering Red Hat systems.

Start with a simple command: **yum list**. It'll return a list of all packages, whether they're installed or available, along with their version numbers and repositories. **yum list | grep *packagename*** tells you what version of a package you will get with a yum install. If you want to show all the configured repositories, you can do this with **yum repolist all**. More information about a specific package can be obtained via the **yum info** command. For example, the following command is functionally equivalent to **rpm -qi samba**:

```
# yum info samba
```

The **rpm -qi** command works if the queried package is already installed. The **yum info** command is not subject to that limitation.

Installation Mode

There are two basic installation commands. If you haven't installed a package before, or you want to update it to the latest stable version, run the **yum install *packagename*** command. You don't need to specify the version or release number. Only the package name is required. For example, if you're checking for the latest version of the Samba RPM, the following command will update it or add it if it isn't already installed on the target system:

```
# yum install samba
```

If you just want to keep the packages on a system up to date, run the **yum update *packagename*** command. For example, if you already have the Samba RPM installed, the following command makes sure it's updated to the latest version:

```
# yum update samba
```

If you haven't installed Samba, this command doesn't add it to your installed packages. In that way, the **yum update** command is analogous to the **rpm -F** command.

Of course, the **yum** command is not complete without options that can remove a package. The first one is straightforward because it uninstalls the Samba package along with any dependencies:

```
# yum remove samba
```

The **yum update** command by itself is powerful; if you want to make sure that all installed packages are updated to the latest stable versions, run the following command:

```
# yum update
```

The **yum update** command may take some time as it communicates with the Red Hat Portal or other repositories. It may need to download the current database of packages with all dependencies. It then finds all packages with available updates and adds them to the list of packages to be updated. It finds all dependent packages if they're not already included in the list of updates.

What if you just want a list of available updates? The **yum list updates** command can help there. It's functionally equivalent to the **yum check-update** command.

But what if you aren't quite sure what to install? For example, if you want to install the Evince document reader and think the operational command includes the term "evince," then the **yum whatprovides "**evince**"** command can help.

Alternatively, to search for all instances of files with the .repo extension, run the following command:

```
# yum whatprovides "*.repo"
```

It lists all instances of the packages with files that end with the .repo extension, with the associated RPM package. The wildcard is required because the **whatprovides** option requires the full path to the file. It accepts partial filenames; for example, the **yum whatprovides "/etc/systemd/*"** command returns the RPM associated with files in the /etc/systemd directory. Once the needed package is known, you can proceed with the **yum install *packagename*** command.

In many cases, problems with yum can be solved with the yum clean all **command. If there are recent updates to Red Hat packages (or third-party repositories), this command flushes the current cache of headers, allowing you to synchronize headers with configured repositories, without having to wait the default six hours before the cache is automatically flushed.**

Security and yum

GPG digital signatures can verify the integrity and authenticity of yum updates. It's the same system described earlier in this chapter for RPM packages. As an example, look at the output the first time a new package is installed over a network on RHEL 7:

```
# yum install samba
```

After packages are downloaded, you'll see something similar to the following messages:

```
Importing GPG key 0xFD431D51:
 Userid     : "Red Hat, Inc. (release key 2) <security@redhat.com>"
 Fingerprint: 567e 347a d004 4ade 55ba 8a5f 199e 2f91 fd43 1d51
 Package    : redhat-release-server-7.0-1.el7.x86_64 (@anaconda/7.0)
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
Is this ok [y/N]: y
Importing GPG key 0x2FA658E0:
 Userid     : "Red Hat, Inc. (auxiliary key) <security@redhat.com>"
 Fingerprint: 43a6 e49c 4a38 f4be 9abf 2a53 4568 9c88 2fa6 58e0
 Package    : redhat-release-server-7.0-1.el7.x86_64 (@anaconda/7.0)
```

```
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
Is this ok [y/N]: y
```

If you're simultaneously downloading packages from other repositories, additional GPG keys may be presented for approval. As suggested by the last line, **N** is the default response; you actually have to type in **y** to proceed with the download and installation—not only of the GPG key, but also of the package in question.

You may notice that the GPG key used is from the same directory of keys associated with the **rpm** command earlier in this chapter.

Updates and Security Fixes

Red Hat maintains a public list of errata, classified by RHEL release, at https://rhn.redhat.com/errata. If you have a RHEL subscription, affected packages are normally made available through the Red Hat Portal; for RHSM-connected machines, all you need to do is run the **yum update** command periodically. This list is useful for those third parties who use RHEL source code, such as CentOS, Scientific Linux, and even Oracle Linux; typically RHEL rebuilds provide similar errata shortly after Red Hat.

Package Groups and yum

The **yum** command can do more. It can install and remove packages in groups. These are the groups defined in the *-comps-Server.x86_64.xml file described in Chapter 2. One location for that file is on the RHEL 7 DVD in the /repodata subdirectory. At the start of most of those stanzas, you'll see the **<id>** and **<name>** XML directives, which list two identifiers for each of those groups.

But that's a lot of work to find a package group. The **yum** command makes it simpler. With the following command, you can identify available package groups from configured repositories:

```
# yum group list
```

Note how the groups are divided into installed and available groups. Some of the groups listed may be of particular interest, such as "Basic Web Server," which you'll use in Chapter 14. To find out more about this group, run the following command. The output is shown in Figure 7-10.

```
# yum group info "Basic Web Server"
```

There are two types of groups in yum: regular groups, which include standard RPM packages, and environment groups, which are made of other groups. "Basic Web Server" in Figure 7-10 is identified as an "environment group" and is in fact a collection of regular groups. Environment groups and regular groups are associated with an environment ID and group ID, respectively, which is shown by the **yum group info** command. These IDs are

FIGURE 7-10

Packages in the Basic Web Server group

```
[root@server1 ~]# yum group info "Basic Web Server"
Loaded plugins: langpacks, product-id

Environment Group: Basic Web Server
 Environment-Id: web-server-environment
 Description: Server for serving static and dynamic internet content.
 Mandatory Groups:
    base
    core
    web-server
 Optional Groups:
   +backup-client
   +directory-client
    guest-agents
   +hardware-monitoring
   +java-platform
   +large-systems
   +load-balancer
   +mariadb-client
   +network-file-system-client
   +performance
   +perl-web
   +php
   +postgresql-client
   +python-web
   +remote-system-management
   +web-servlet
[root@server1 ~]#
```

alternative names for the groups, without spaces or uppercase characters, and they are often used in Kickstart configuration files.

To list all groups, you can type

```
# yum group list hidden
```

Let's get some information about one of the regular groups:

```
# yum group info "Remote Desktop Clients"
```

Note how the packages are all listed as "Optional Packages." In other words, they're not normally installed with the package group. Thus, suppose you were to run the following command:

```
# yum group install "Remote Desktop Clients"
```

Nothing would be installed. Desired packages from this package group have to be specifically named to be installed with commands like the following:

```
# yum install tigervnc
```

FIGURE 7-11

Packages in the Print Server group

```
[root@server1 ~]# yum group info "Print Server"
Loaded plugins: langpacks, product-id

Group: Print Server
 Group-Id: print-server
 Description: Allows the system to act as a print server.
 Mandatory Packages:
   +cups
   +ghostscript-cups
 Default Packages:
   +foomatic
   +foomatic-filters
   +gutenprint
   +gutenprint-cups
   +hpijs
   +paps
[root@server1 ~]#
```

But optional packages are not the only category. The following command lists all packages in the "Print Server" package group. The output is shown in Figure 7-11.

```
# yum group info "Print Server"
```

Packages in the Print Server group are classified in two other categories. Mandatory packages are always installed with the package group. Default packages are normally installed with the package group; however, specific packages from this group can be excluded with the **-x** switch. For example, the following command installs the two mandatory and six default packages:

```
# yum group install "Print Server"
```

In contrast, the following command excludes the paps and the gutenprint-cups packages from the list of those to be installed:

```
# yum group install "Print Server" -x paps -x gutenprint-cups
```

After running this command, show again a list of the packages in the Print Server package group:

```
# yum group info "Print Server"
```

The output is shown in Figure 7-12. Compare this with Figure 7-11. You will notice that some of the packages have an equal sign (=) marker in front. This means that the corresponding package was installed using the **yum group install** command. Conversely, the minus marker (-) indicates that a package was excluded from installation and will not be installed if we upgrade or install the group. Similarly, the plus marker (+) indicates that a package is not installed, but it

FIGURE 7-12

Packages after the Print Server group is installed

```
[root@server1 ~]# yum group info "Print Server"
Loaded plugins: langpacks, product-id

Group: Print Server
 Group-Id: print-server
 Description: Allows the system to act as a print server.
 Mandatory Packages:
   =cups
   =ghostscript-cups
 Default Packages:
   =foomatic
   =foomatic-filters
   =gutenprint
   -gutenprint-cups
   =hpijs
   -paps
[root@server1 ~]#
```

will be added to the system if we install or upgrade the group. If no marker is present, then the package was installed, but not as part of a **yum group install** command.

The options to the **yum** command are not complete unless there's a command that can reverse the process. As suggested by the name, the **group remove** command uninstalls all packages from the noted package group:

```
# yum group remove "Print Server"
```

Exclusions are not possible with the **yum group remove** command. If you don't want to remove all packages listed in the output to the command, it may be best to remove target packages individually.

More yum Commands

A number of additional **yum**-related commands are available. Two of them may be of particular interest to those studying for the Red Hat exams: **yum-config-manager** and **yumdownloader**, which can display all current settings for each repository as well as download individual RPM packages. One more related command is **createrepo**, which can help you set up a local repository.

View All Directives with yum-config-manager

To some extent, the directives listed in the yum.conf and related configuration files provide only a small snapshot of available directives. To review the full list of directives, run the **yum-config-manager** command. Pipe it to the **less** command as a pager. It includes more than 300 lines. The excerpt from the [**main**] section shown in Figure 7-13 includes settings that apply to all the configured repositories.

FIGURE 7-13

A partial list of yum directives

```
fssnap_automatic_keep = 1
fssnap_automatic_post = False
fssnap_automatic_pre = False
fssnap_devices = !*/swap,
   !*/lv_swap
fssnap_percentage = 100
gaftonmode = False
gpgcheck = True
group_command = objects
group_package_types = mandatory,
   default
groupremove_leaf_only = False
history_list_view = single-user-commands
history_record = True
history_record_packages = yum,
   rpm
http_caching = all
installonly_limit = 3
installonlypkgs = kernel,
   kernel-bigmem,
   installonlypkg(kernel-module),
   installonlypkg(vm),
   kernel-enterprise,
   kernel-smp,
   kernel-debug,
   kernel-unsupported,
   kernel-source,
   kernel-devel,
   kernel-PAE,
   kernel-PAE-debug
```

Some of the directives associated with **yum** are not filled in, such as **exactarchilist**; some don't really matter, such as the color directives. Some of the other significant directives are shown in Table 7-5. It is not a comprehensive list. If you're interested in a directive not shown, it's defined in the man page for the yum.conf file.

yum-config-manager can also manage repositories. For example, if you know the URL of a repository, you can automatically generate a configuration file using a command similar to the following:

```
# yum-config-manager --add-repo="http://192.168.122.1/inst"
```

Package Downloads with yumdownloader

As suggested by the name, the **yumdownloader** command can be used to download packages from yum-based repositories. It's a fairly simple command. For example, the following command reviews the contents of configured repositories for a package named cups:

```
# yumdownloader cups
```

TABLE 7-5 Configuration Parameters from yum-config-manager

Configuration Directive in yum	Description
alwaysprompt	Prompts for confirmation on package installation or removal.
assumeyes	Set to no by default; if set to 1, yum proceeds automatically with package installation and removal.
cachedir	Set to the directory for database and downloaded package files.
distroverpkg	Lists the RPM packages that yum checks to find the version of the Linux distribution installed on the current machine.
enablegroups	Supports **yum group*** commands.
installonlypkgs	Lists packages that should never be updated; normally includes Linux kernel packages.
logfile	Specifies the name of file with log information, normally /var/log/yum.log.
pluginconfpath	Notes the directory with plugins, normally /etc/yum/pluginconf.d.
reposdir	Specifies the directory with repository configuration files.
ssl*	Supports the use of the Secure Sockets Layer (SSL) for secure updates.
tolerant	Determines whether yum stops if an error is encountered with one of the packages.

Either the RPM package is downloaded to the local directory or the command returns the following error messages:

```
No Match for argument cups
Nothing to download
```

Sometimes, more specifics are required. If there are multiple versions of a package stored on a repository, the default is to download the latest version of that package. That may not always be what you want. For example, if you want to use the originally released RHEL 7 kernel, use the following command:

```
# yumdownloader kernel-3.10.0-123.el7
```

Create Your Own Repository with createrepo

An earlier version of the RHCE objectives for RHEL 6 suggested that you should know how to "create a private yum repository." Although that objective has since been removed, it's a necessary job skill for a Red Hat system engineer.

Custom repositories can provide additional control. Enterprises that want to control the packages installed on their Linux systems can create their own customized repository. Although this can be based on the standard repositories developed for a distribution, it can include additional packages such as custom software unique to an organization. Just as easily, it can omit packages that may violate organizational policies such as games. Limits on the choices for certain functions such as browsers can minimize related support requirements.

To create a customized repository, you need to collect desired packages in a specific directory. The **createrepo** command can process all packages in that directory. The database is created in XML files in a repodata/ subdirectory. An example of this package database already exists in the repodata/ subdirectory of the RHEL 7 DVD.

The Red Hat Portal enables support of customized repositories with related products, such as Red Hat Satellite Server. For more information on repository management, see *Linux Patch Management* written by Michael Jang and published by Prentice Hall.

CERTIFICATION OBJECTIVE 7.04

More Package Management Tools

Whether a Red Hat system is connected to Red Hat Customer Portal or repositories provided by a distribution such as CentOS or Scientific Linux, it will use the same basic package management tools. Regardless of source, the **rpm** command is used to process RPM packages. Higher-level tools such as the **yum** command are used to satisfy dependencies and install groups of packages. This makes sense because the rebuild distributions are based on the same source code as RHEL 7 and all are distributed via RPM.

These similarities extend to GUI-based package management tools. While the identity of these tools has changed between RHEL 6 and RHEL 7, they are still front ends to the **rpm** and **yum** commands. They take advantage of the package groups configured in the .xml file described in Chapter 2. Since Red Hat uses GNOME as the default GUI desktop environment, the associated software management tools are based on that interface.

In RHEL 7, GUI-based package management tools rely on PackageKit, a common abstraction layer that provides a unified interface to all Linux software management applications. However, it's quite possible that PackageKit won't be available on a server, or perhaps even a system configured for a Red Hat exam. If you absolutely need PackageKit, install the required RPMs with the **yum install gnome-packagekit** command. Of course, if you're already comfortable with the **yum** command, you may not need PackageKit.

exam watch

Although the RHN is listed as part of the RHCSA objectives, it's listed in context as a choice. Whether you're installing or updating software packages from RHN, "a remote repository, or a local filesystem," you can use the rpm **and** yum **commands. Of course, it's simplest if you have an official Red Hat subscription.**

The GNOME Software Update Tool

The Software Update tool can be started from a GUI terminal with the **gpk-update-viewer** command. Alternatively, from the GNOME Desktop Environment, click Applications | System Tools | Software Update. The tool, as shown in Figure 7-14, lists packages that are available for update.

FIGURE 7-14 The GNOME Software Update tool

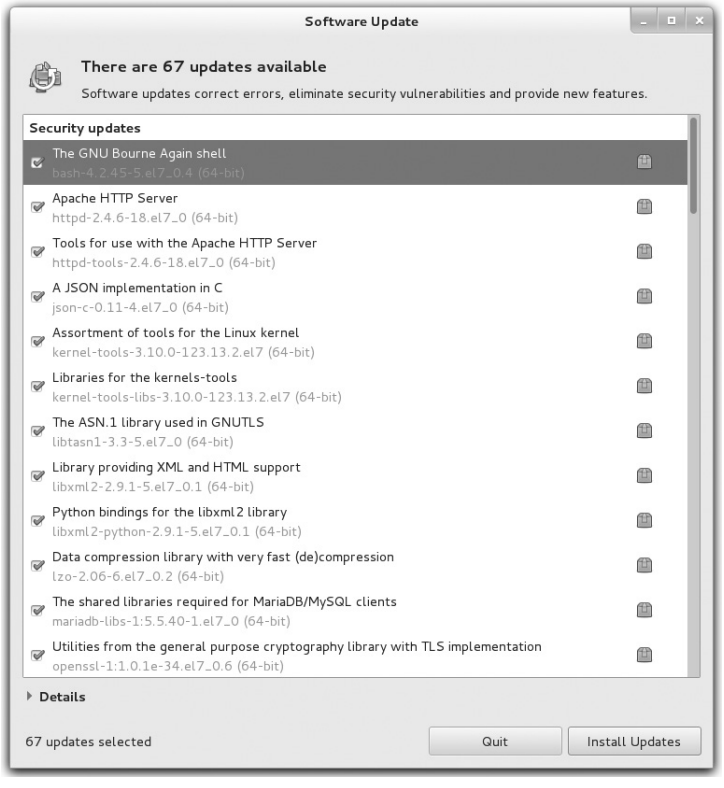


FIGURE 7-15 The GNOME Software Update Preferences tool

It's a pretty straightforward interface. It's effectively a front end to the **yum update** command. Note the additional information, with a description of changes.

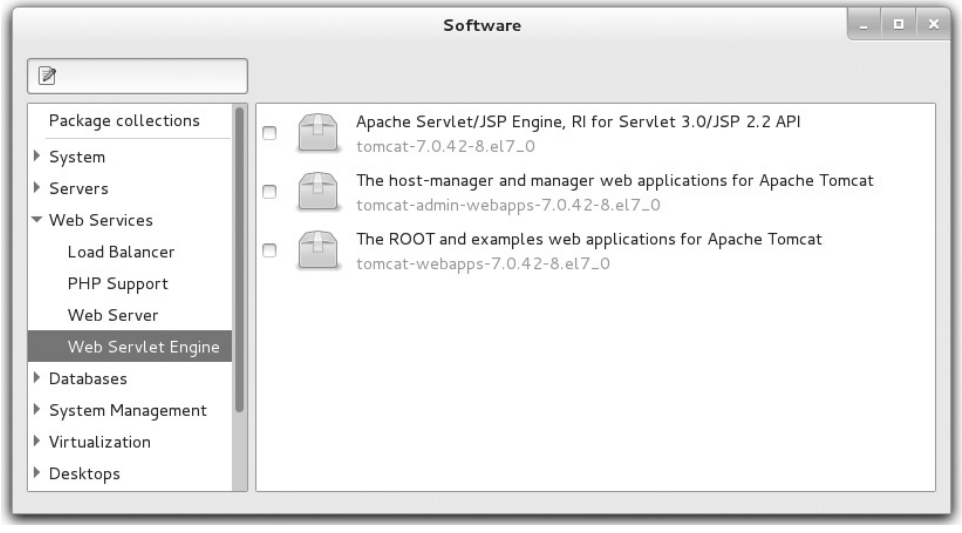
Automated Updates

It is important to install the latest security updates as quickly as possible. To do that, open the Software Update Preferences tool shown in Figure 7-15. You can open it from a GUI command line with the **gpk-prefs** command. You can configure the system to check for updates on an hourly, daily, or a weekly basis, or not at all. When updates are found, you can configure automatic installation of all available updates, of only security updates, or of nothing at all.

GNOME Software Tool

You can add, update, and remove packages with a graphical tool. To start the GNOME Software tool from a GUI command line, run the **gpk-application** command, or click Applications | System Tools | Software. This opens the tool shown in Figure 7-16. Here, you can install more than one package or package group at a time. Once packages are selected

FIGURE 7-16 The GNOME Software tool



(or deselected), the tool automatically calculates dependencies and installs (or removes) them, along with the selected packages.

You can use the GNOME Software tool to add the packages or package groups of your choice. In the upper-left part of the screen is the **Package Collections** option, which lists the same groups shown in the output to the **yum group list** command described earlier.

Software packages are further subdivided in the lower-left part of the screen. When a package or package group is selected or deselected for installation or removal, the Apply Changes button becomes clickable. Once clicked, the tool uses the **yum** command to calculate dependencies. If there are no dependencies, the installation proceeds immediately. If there are dependencies, the entire list of packages to be installed or removed is presented for your approval.

EXERCISE 7-2

Installing More with yum and the GNOME Software Tool

This exercise requires a network connection to a remote repository, or at least a RHEL 7 DVD copied or mounted as a repository, as configured earlier in this chapter. If you're using a rebuild of RHEL 7, you'll need to make sure the connection to the core repository is active,

perhaps with a **ping** command to the host of that repository, as defined in the appropriate file in the /etc/yum.repos.d directory. Given the possible variations, exact steps are not possible.

1. Run the **yum list** command. Assuming an active network connection and a responsive repository, you'll see a full list of available packages, including those already installed. Note the label in the right column; it will either show the repository where a package is available or note that the package is already installed.
2. Enter the **gpk-application** command in a GUI command line. This should open the GNOME Software tool.
3. In a second command-line console, type in the **yum group list** command. In the GNOME Software tool, select Package Collections. Compare the list of package groups in each output.
4. Review available package groups in the GNOME Software tool. For example, click the arrow next to Servers. Under the options that appear, click FTP Server. There are only two official packages in the RHEL 7 configuration of this group. Select the packages, which will be installed when you click Apply Changes.
5. Locate the text box in the upper-left corner of the GNOME Software tool. Type in a common search term such as *gnome* and watch as a long list of packages are shown. Compare the result to the output of the **yum search gnome** command.
6. Use a less common search term such as *iptables.* Highlight the iptables package and review it in the lower-right part of the screen. Compare the result to the output of the **yum info iptables** command.
7. Select again the iptables packages and click the Files and Dependent Packages buttons in the lower-right part of the screen. Compare the results to the output of the commands **repoquery -l iptables** and **yum deplist iptables**.
8. Once you've selected some packages, click Apply Changes. If there are dependencies, they will be automatically installed.
9. Wait as packages are installed. When finished, close the GNOME Software tool.

Red Hat Subscription Manager

The RHCSA exam objectives require candidates to be able to "install and update software packages from Red Hat Network." However, at the time of writing, RHEL 7 systems no longer support the use of the Red Hat Network (RHN). Subscriptions to RHN repositories are available only through a local installation of an older version of Red Hat Satellite. Newer versions of Red Hat Satellite Server and stand-alone RHEL 7 systems use the Red

Hat Customer Portal Subscription Management (RHSM) to access Red Hat software repositories.

Remember, the related objective suggests that all you need to know is how to install and update packages from the RHN. And that skill was already covered with the **rpm** and **yum** commands, along with the related GUI tools discussed in most of this chapter.

Perhaps the key benefit of Red Hat Satellite (or an alternative such as Spacewalk or Katello) is the ability to manage all RHEL and rebuild distribution systems remotely over a web-based interface. Once an appropriate connection is configured from the client systems, Satellite Server can even run remote commands on any schedule. If you're administering a whole bunch of systems, Red Hat Satellite supports configuration of systems in groups. For example, if there are 10 systems configured as RHEL 7–based web servers, you can configure those systems as a single group. You can then schedule a single command that's applied to all of those systems remotely. For more information on Red Hat Satellite, see the latest version of the documentation, available from https://access.redhat.com/documentation/.

If you have access to the Red Hat Customer Portal Subscription Management (RHSM), take the following steps to register and subscribe to a RHEL 7 system:

1. Run the following command to register a system with RHSM. Include a username and password of a valid Red Hat account. If the system is already registered, use the **--force** command option to re-register the machine.

   ```
   # subscription-manager --username=USERNAME --password=PASSWORD
   ```

2. Subscribe the system to a Red Hat product. In the following command, the **--auto** option finds the most appropriate subscription:

   ```
   # subscription-manager attach --auto
   ```

3. Alternatively, list all available subscriptions. Take note of the pool IDs.

   ```
   # subscription-manager list --available
   ```

   Then, use the pool ID that you have retrieved from the last command to attach a system to a subscription:

   ```
   # subscription-manager attach --pool=8a85f98146f719180146fd9593b7734c
   ```

4. Review current settings. Run the following command to list the subscriptions attached to the system:

   ```
   # subscription-manager list
   ```

5. Show all available repositories for the system:

```
# subscription-manager repos
```

6. You can enable additional repositories with the following command:

```
# subscription-manager repos --enable=REPOID
```

7. If you prefer, you can use the GUI version of Subscription Manager, which can be started with the **subscription-manager-gui** command. Alternatively, from the GNOME Desktop Environment, click Applications | System Tools | Red Hat Subscription Manager.

CERTIFICATION SUMMARY

This chapter focuses on the management of RPM packages. With different command options, you also saw how the **rpm** tool installs, removes, and upgrade packages, as well as how it works locally and remotely. When presented with a new version of a kernel, it's important to never replace the existing kernel with **rpm.** A properly configured installation of a later kernel version does not overwrite, but brings the kernels together, side by side. You'll then be able to boot into either kernel.

With the **rpm** command, you also learned how to query packages, to examine to which package a file belongs, to validate a package signature, and to find the current list of installed RPMs. You also saw the difficulties associated with dependencies that drove users to the **yum** command.

The **yum** command is, in part, a front end to the **rpm** command. When there are dependencies, it installs those packages simultaneously. You learned how to configure Red Hat and other repositories to work with the **yum** command. You should now be able to configure even the RHEL 7 DVD as its own repository. As you saw, the **yum** command also can install or remove package groups, as defined by the XML database file of packages on the RHEL 7 DVD and other repositories. The **yum** command is fully compatible with the RHSM.

Although additional package management tools are available from the GUI, they are front ends to the **yum** and **rpm** commands. With the **gpk-update-viewer** command, you started the Software Update tool to identify and install available updates. With the **gpk-prefs** command, you started the Software Update Preferences tool, which can check for and install security or all available updates on a regular schedule. With the **gpk-application** command, you opened the GNOME Software tool, which also can be used to add or remove packages and package groups. If you have a RHEL subscription, systems can also be kept up to date and registered to RHSM through the **subscription-manager** command.

TWO-MINUTE DRILL

Here are some of the key points from the certification objectives in Chapter 7.

The Red Hat Package Manager

- ❑ The RPM database tracks where each file in a package is located, its version number, and much more.
- ❑ The **rpm -i** command installs RPM packages.
- ❑ The **rpm -e** command uninstalls RPM packages.
- ❑ The **rpm** command can even install RPMs directly from remote servers.
- ❑ RPM package verification is supported by the GPG keys in the /etc/pki/rpm-gpg directory.
- ❑ Kernel RPMs should always be installed, never upgraded.
- ❑ The Upgrade mode of RPM replaces the old version of the package with the new one.

More RPM Commands

- ❑ The **rpm -q** command determines whether packages are installed on a system; with additional switches, it can list more about a package and identify the package for a specific file.
- ❑ Package signatures can be checked with the **rpm --checksig** (or **-K**) command.
- ❑ The **rpm -V** command can identify files that have changed from the original installation of the package before the RPM is installed.
- ❑ The **rpm -qa** command lists all currently installed packages.

Dependencies and the yum Command

- ❑ By including additional required packages, the **yum** command can help avoid "dependency hell."
- ❑ The behavior of the **yum** command is configured in the /etc/yum.conf file, plugins in the /etc/yum/pluginconf.d directory, and repositories configured in the /etc/yum.repos.d directory.
- ❑ Red Hat organizes packages in several different repositories for RHEL 7.
- ❑ Repositories for rebuild distributions and from third parties are accessible online.
- ❑ The **yum** command can install, erase, and update packages. It also can be used to search in different ways.

- ❑ The **yum** command uses the GPG keys developed for RPM packages.
- ❑ The **yum** command can install, remove, and list package groups.

More Package Management Tools

- ❑ RHEL 7 package management tools are based on the PackageKit built for GNOME.
- ❑ With the GNOME Software tool, you can install and remove packages and package groups.
- ❑ The PackageKit also includes tools focused on current updates. It can also set up updates of security packages or all packages on a schedule.
- ❑ The RHSM and Red Hat Satellite can help you manage subscribed systems remotely using a web-based interface.

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. There may be more than one answer to many of these questions.

The Red Hat Package Manager

1. What command would you use to install the penguin-3.26.x86_64.rpm package, with extra messages in case of errors? The package is on the local directory.

   _____

2. What command would you use to upgrade the penguin RPM with the penguin-3.27.x86_64.rpm package? The package is on the ftp.remotemj02.abc server.

   _____

3. If you've downloaded a later version of the Linux kernel to the local directory and the package filename is kernel-3.10.0-123.13.2.el7.x86_64.rpm, what's the best command to make it a part of your system?

   _____

4. What directory contains RPM GPG keys on an installed system?

   _____

More RPM Commands

5. What command lists all currently installed RPMs?

   _____

6. What command lists all the files in the package penguin-3.26.x86_64.rpm?

   _____

7. If you've downloaded an RPM from a third party called third.i686.rpm, how can you validate the associated package signature?

   _____

Dependencies and the yum Command

8. What is the full path to the directory where the location of **yum** repositories are normally configured?

   _____

9. What command searches **yum** repositories for the package associated with the /etc/passwd file?

   _____

More Package Management Tools

10. What command-line command lists the package groups shown in the GNOME Software tool?

    _____

11. Name two allowable time periods for automatic updates, as defined by the Software Update Preferences tool.

    _____

    _____

12. What command from the console starts the process of registration on RHSM?

    _____

LAB QUESTIONS

Red Hat presents its exams electronically. For that reason, the labs in this chapter are available from the DVD that accompanies the book in the Chapter7/ subdirectory. They're available in .doc, .html, and .txt format to reflect standard options associated with electronic delivery on a live RHEL 7 system. In case you haven't yet set up RHEL 7 on a system, refer to the first lab of Chapter 2 for installation instructions. The answers for each lab follow the Self Test answers for the fill-in-the-blank questions.

A SELF TEST ANSWERS

The Red Hat Package Manager

1. The command that installs the penguin-3.26.x86_64.rpm package, with extra messages in case of errors, from the local directory is

```
# rpm -iv penguin-3.26.x86_64.rpm
```

   Additional switches that don't change the functionality of the command, such as **-h** for hash marks, are acceptable. This applies to subsequent questions as well.

2. The command that upgrades the aforementioned penguin RPM with the penguin-3.27.x86_64 .rpm package from the ftp.remotemj02.abc server is

```
# rpm -Uv ftp://ftp.remotemj02.abc/penguin-3.26.x86_64.rpm
```

   If you use the default vsFTP server, the package may be in the pub/Packages/ subdirectory. In other words, the command would be

```
# rpm -Uv ftp://ftp.remotemj02.abc/pub/Packages/penguin-3.26.i386.rpm
```

   Yes, the question is not precise—but that's what you see in real life.

3. If you've downloaded a later version of the Linux kernel to the local directory and the package filename is kernel-3.10.0-123.13.2.el7.x86_64.rpm, the best way to make it a part of your system is to install it—and not upgrade the current kernel. Kernel upgrades overwrite existing kernels. Kernel installations allow kernels to exist side by side; if the new kernel doesn't work, you can still boot into the working kernel. As the desired package is already downloaded, you'd use a command similar to the following:

```
# rpm -iv kernel-3.10.0-123.13.2.el7.x86_64.rpm
```

   Variations of the **rpm** command, such as **rpm -i** and **rpm -ivh**, are acceptable. However, variations that upgrade, with the **-U** or **-F** switches, are incorrect.

4. The directory with RPM GPG keys on an installed system is /etc/pki/rpm-gpg. The GPG keys on the RHEL 7 CD/DVD are not "installed" on a system.

More RPM Commands

5. The command that lists all installed RPMs is

```
# rpm -qa
```

6. The command that lists all the files in the package penguin-3.26.x86_64.rpm is

```
# rpm -ql penguin-3.26.x86_64.rpm
```

7. If you've downloaded an RPM from a third party, call it third.i686.rpm, you'll first need to download and install the RPM-GPG-KEY file associated with that repository. You can then validate the associated package signature with a command like the following (note the uppercase **-K**; **--checksig** is equivalent to **-K**):

```
# rpm -K third.i386.rpm
```

Dependencies and the yum Command

8. The **yum** command repositories are normally configured in files in the /etc/yum.repos.d directory. Technically, **yum** command repositories can also be configured directly in the /etc/yum.conf file.
9. The **yum whatprovides /etc/passwd** command identifies packages associated with that file.

More Package Management Tools

10. This is a slightly tricky question because the **yum group list** command lists the package groups also shown in the GNOME Software tool.

11. Allowable time periods for updates, as defined by the Software Update Preferences tool, are hourly, daily, and weekly.
12. The **subscription-manager** command starts the process of registering a system on RHSM.

LAB ANSWERS

Lab 1

When the lab is complete, run the following commands to verify the connection:

```
# yum clean all
# yum update
```

The output should look similar to the following:

```
Loaded plugins: langpacks, product-id, subscription-manager
inst                                     | 3.7 kB     00:00
inst/primary_db                          | 2.9 MB     00:00
Setting up Update Process
```

This output verifies a successful connection to the FTP server. If you see something significantly different, check the following in the /etc/yum.repos.d/file.repo file:

- Make sure the stanza in this file starts with [**inst**].
- Check the URL associated with the **baseurl** directive. It should match the URL of the FTP server defined in Chapter 1, Lab 2. You should be able to run the **lftp** and **ftp** commands with that URL from a command-line interface. If that doesn't work, either the FTP server is not running or messages to that server are blocked by a firewall.
- If there were problems, fix them. Then try the previous commands again.

Lab 2

One way to check all of the files in the /usr/sbin directory is to use the **rpm -Va | grep /usr/sbin** command.

If successful, you'll identify the /usr/sbin/vsftpd and /etc/vsftpd/vsftpd.conf files as different from their original versions, as installed from the RPM. Changes to a configuration file are not a big deal, especially if it has been customized in any way. However, changes to the binary file are a reason for suspicion.

Assuming standard Red Hat RPM packages, removal and reinstallation should preserve changes to the vsftpd.conf file in a vsftpd.conf.rpmsave file.

If you really do have a security concern, additional measures are appropriate. For example, some security professionals might compare all files on a suspect system to the files on a verified baseline system.

In that case, it may be simplest to take a copy or clone of the baseline system, reinstall the vsftpd RPM, and reconfigure it as needed. Assuming the baseline system is secure, you'd then be reasonably sure the new server would also be secure.

The changes made by the script to this lab set a new modification time for the /usr/sbin/vsftpd binary and appended a comment to the end of the vsftpd.conf configuration file. If you want to restart with fresh copies of these packages, back up your current vsftpd.conf file and run the **rpm -e vsftpd** command to uninstall the package. If the RPM package was reconfigured, you should see at least the following warning message:

```
warning: /etc/vsftpd/vsftpd.conf saved as /etc/vsftpd/vsftpd.conf.rpmsave
```

You can then reinstall the original package from either the installation DVD or a remote repository. Alternatively, you could delete (or move) the changed files and then run the following command to force the **rpm** command to provide the original copies of these files from the associated package. The version number is based on the RHEL 7.0 DVD.

```
# rpm -ivh --force vsftpd-3.0.2-9.el7.x86_64.rpm
```

Lab 3

This lab is intended to help you examine what the **yum update** command can do. It's the essential front end to GUI update tools. As you can see from the update.txt file created in this lab, the messages display how **yum** appears for all newer packages from configured repositories or the RHN, downloads their headers, and uses them to check for dependencies that also need to be downloaded and installed.

Lab 4

This lab should be straightforward because it involves the use of the Software Update Preferences tool, which you can start from a GUI command line with the **gpk-prefs** command.

Lab 5

This lab is somewhat self-explanatory and is intended to help you explore what happens when you properly install a new kernel RPM. As with other Linux distributions, when you install (and do not use upgrade mode for) a new kernel, two areas are affected.

The new kernel is added as a new option in the GRUB2 configuration menu. The existing kernel should be retained as an option in that menu. When you reboot the system, try the new kernel. Don't hesitate to reboot the system again and then try the other option, probably the older kernel.

When you review the /boot directory, all of the previously installed boot files should be there. The new kernel RPM should add matching versions of all of the same files—with different revision numbers.

To keep this all straight, it helps if you made copies of the original versions of the GRUB 2 configuration file and the file list in the /boot directory. If you choose to retain the newly installed kernel, great. Otherwise, uninstall the newly installed kernel. This is one case where revision numbers are required with the **rpm -e** command; the following is based on the removal of the kernel and kernel-firmware packages, based on version number 3.10.0-229.el7:

```
# rpm -e kernel-3.10.0-229.el7.x86_64
# rpm -e linux-firmware-20140911-0.1.git365e80c.el7
```

If the revision number of the kernel or kernel-firmware package that you installed during this lab is different, adjust the commands accordingly.

Lab 6

This lab is designed to give you practice with both the **yum** command and the Add/Remove Software tool. It should help you prepare for Chapter 9 and provide the skills required to install services for other chapters. You should realize by now that because all packages in the Remote Desktop Clients package group are optional, the **yum group install "Remote Desktop Clients"** command doesn't install anything. You'll need to install each of the optional packages by name.

To identify the names of the packages to be installed, run the **yum group info "Remote Desktop Clients"** command. Be sure to install every package from that group on both systems. The best method is with the **yum install** package1 package2 ... command, where *package1, package2,* and so on, are names of packages in the "Remote Desktop Clients" package group.