RHCSA - EX200

Trainer

Ali Aydemir
CISCO, CCIE #47287 SP/RS, CCSI#35413
AVAYA, ACE-Fx #169
HUAWEI, HCDP

RHCSA Timetable

Day	AM	Lunch	PM
1	-Installing RHEL Server -Using Essential Tools	--	-Essential File Management -ToolsWorking with Text Files -Connecting to a RHEL Server
2	-User and Group Management -Permissions Management	--	-Configuring Networking -Process Management -Working with Virtual Machines
3	-Installing Software Packages -Scheduling Tasks	--	-Configuring Logging -Managing Partitions -Managing LVM Logical Volumes
4	-Basic Kernel Management -Configuring a Basic Apache Server	--	-Managing and Understanding the Boot Procedure -Essential Boot Procedure Troubleshooting
5	-Managing SELinux -Configuring a Firewall	--	-Configuring Remote Mounts and FTP -Configuring Time Services

Chapter 16:

Basic Kernel Management

Chapter 16 Objectives

- The following topics are covered in this chapter:
  - Understanding the Role of the Linux Kernel
  - Working with Kernel Modules
  - Updating the Linux Kernel

- The following RHCSA exam objectives are covered in this chapter:
  - Update the kernel package appropriately to ensure a bootable system

Basic Kernel Management

- The Linux kernel is the heart of the Linux operating system. It takes care of many things, including hardware management. In this chapter, you learn all you need to know about the Linux kernel from an RHCSA perspective. In fact, you even learn a bit more.

- This chapter includes information about topics that are not on the current list of RHCSA objectives. I think it is good to know about these topics anyway. Any serious Linux administrator should be able to deal with issues related to the topics discussed in this chapter.

Understanding the Role of the Linux Kernel

The Linux kernel is the heart of the operating system. It is the layer between the user who works with Linux from a shell environment and the hardware that is available in the computer on which the user is working. The kernel is doing so by managing the I/O instructions it receives from the software and translating those to processing instructions that are to be executed by the central processing unit and other hardware in the computer. The kernel also takes care of handling essential operating system tasks. One example of such a task is the scheduler that makes sure that processes that are started on the operating system are handled by the CPU.

Understanding the use of Kernel Threads and Drivers

The operating system tasks that are performed by the kernel are implemented by different kernel threads. Kernel threads are easily recognized with a command like **ps aux**. The kernel thread names are listed between square brackets (see Listing 16.1).

Understanding the use of Kernel Threads and Drivers

Listing 16.1 Listing Kernel Threads with **ps aux**

```
[root@server1 ~]# ps aux | head -n 20
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root     1   1.8  0.6  52980  6812 ?        Ss   11:44   0:02 /usr/lib/
systemd/systemd --switched-root --system --deserialize 23
root     2   0.0  0.0      0     0 ?    S     11:44   0:00 [kthreadd]
root     3   0.0  0.0      0     0 ?    S     11:44   0:00 [ksoftirqd/0]
root     4   0.0  0.0      0     0 ?    S     11:44   0:00 [kworker/0:0]
root     5   0.0  0.0      0     0 ?    S<    11:44   0:00 [kworker/0:0H]
root     6   0.0  0.0      0     0 ?    S     11:44   0:00 [kworker/u128:0]
root     7   0.1  0.0      0     0 ?    S     11:44   0:00 [migration/0]
root     8   0.0  0.0      0     0 ?    S     11:44   0:00 [rcu_bh]
root     9   0.0  0.0      0     0 ?    S     11:44   0:00 [rcuob/0]
root     10   0.0  0.0     0     0 ?    S     11:44   0:00 [rcuob/1]
root     11   0.0  0.0     0     0 ?    S     11:44   0:00 [rcuob/2]
root     12   0.0  0.0     0     0 ?    S     11:44   0:00 [rcuob/3]
```

Analyzing What the Kernel Is Doing

To help analyze what the kernel is doing, some tools are provided by the Linux operating systems:

- The **dmesg** utility
- The /proc file system
- The uname utility

The first utility to consider whether detailed information about the kernel activity is required is dmesg. This utility shows the contents of the kernel ring buffer, an area of memory where the Linux kernel keeps its recent log messages. An alternative method to get access to the same information in the kernel ring buffer is by using the **journalctl --dmesg** command, which is equivalent to **journalctl -k**. In Listing 16.2, you can see a part of the result of the **dmesg** command.

Analyzing What the Kernel Is Doing

Listing 16.2 Analyzing Kernel Activity Using dmesg

```
[    8.153928] sd 0:0:0:0: Attached scsi generic sg0 type 0
[    8.154289] sd 0:0:1:0: Attached scsi generic sg1 type 0
[    8.154330] sd 0:0:2:0: Attached scsi generic sg2 type 0
[    8.154360] sd 0:0:3:0: Attached scsi generic sg3 type 0
[    8.154421] sr 4:0:0:0: Attached scsi generic sg4 type 5
[    8.729016] ip_tables: (C) 2000-2006 Netfilter Core Team
[    8.850599] nf_conntrack version 0.5.0 (7897 buckets, 31588 max)
[    8.939613] ip6_tables: (C) 2000-2006 Netfilter Core Team
[    9.160092] Ebtables v2.0 registered
[    9.203710] Bridge firewalling registered
[    9.586603] IPv6: ADDRCONF(NETDEV_UP): eno16777736: link is not
ready
[    9.587520] e1000: eno16777736 NIC Link is Up 1000 Mbps Full Duplex,
Flow Control: None
[    9.589066] IPv6: ADDRCONF(NETDEV_CHANGE): eno16777736: link becomes
ready
[   10.689365] Rounding down aligned max_sectors from 4294967295 to
4294967288
[ 5158.470480] Adjusting tsc more than 11% (6940512 vs 6913395)
[21766.132181] e1000: eno16777736 NIC Link is Down
[21770.391597] e1000: eno16777736 NIC Link is Up 1000 Mbps Full Duplex,
Flow Control: None
[21780.434547] e1000: eno16777736 NIC Link is Down
```

Analyzing What the Kernel Is Doing

In the dmesg output, all kernel-related messages are shown. Each message starts with a time indicator that shows at which specific second the event was logged. This time indicator is relative to the start of the kernel, which allows you to see exactly how many seconds have passed between the start of the kernel and a particular event. (Notice that the **journalctl -k / --dmesg** commands show clock time, instead of time that is relative to the start of the kernel.) This time indicator gives a clear indication of what has been happening and at which time it has happened.

A last useful source of information that should be mentioned here is the **uname** command. This command gives different kinds of information about your operating system. Type, for instance, **uname -a** for an overview of all relevant parameters of **uname -r** to see which kernel version currently is used. This information also shows when using the **hostnamectl status** command.

Analyzing What the Kernel Is Doing

TIP On some occasions, you might need to know specific information about the RHEL version you are using. To get that information, display the contents of the **/etc/redhat-release** command; it will tell you which Red Hat version you are using and which update level is applied. In Listing 16.3, you can see the results of the **uname -r** command and the contents of the redhat-release file.

Listing 16.3 Getting More Information About the System

```
[root@server1 ~]# uname -r
3.10.0-123.el7.x86_64
[root@server1 ~]# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.0 (Maipo)
```

Working with Kernel Modules

- In the old days of Linux, kernels had to be compiled to include all drivers that were required to support computer hardware. Other specific functionality needed to be compiled into the kernel as well.

- Since the release of Linux kernel 2.0 in the mid 1990s, kernels are no longer compiled but modular. A modular kernel consists of a relatively small core kernel and provides driver support through modules that are loaded when required. Modular kernels are very efficient, as only those modules that really are needed are included.

Understanding Hardware Initialization

The loading of drivers is an automated process that roughly goes like this:

1. During boot, the kernel probes available hardware.
2. Upon detection of a hardware component, the **systemd-udevd** process takes care of loading the appropriate driver and making the hardware device available.
3. To decide how the devices are initialized, systemd-udevd reads rules files in /usr/lib/udev/rules.d. These are system provided udev rules files that should not be modified.
4. After processing the system provided udev rules files, systemd-udevd goes to the /etc/udev/rules.d directory to read any custom rules if these are available.
5. As a result, required kernel modules are loaded automatically and status about the kernel modules and associated hardware is written to the sysfs file system which is mounted on the /sys directory.

Understanding Hardware Initialization

The systemd-udevd process is not a one-time only process; it continuously monitors plugging and unplugging of new hardware devices. To get an impression of how this works, as root you can type the command **udevadm monitor**. This is all events that are processed while activating new hardware devices. Use Ctrl+C to close the ude-vadm monitor output.

Managing Kernel Modules

Linux kernel modules normally are loaded automatically for the devices that need them, but you will sometimes have to load the appropriate kernel modules manually. A few commands are used for manual management of kernel modules. Table 16.2 provides an overview.

An alternative method of loading kernel modules is by doing this through the /etc/modules-load.d directory. In this directory, you can create files to load modules automatically that are not loaded by the udev method already.

Managing Kernel Modules

Table 16.2 Linux Kernel Module Management Overview

Command	Use
lsmod	Lists currently loaded kernel modules
modinfo	Displays information about kernel modules
modprobe	Loads kernel modules, including all of their dependencies
modprobe -r	Unloads kernel modules, considering kernel module dependencies

The first command to use when working with kernel modules is **lsmod**. This command lists all kernel modules that currently are used, including the modules by which this specific module is used. Listing 16.5 shows the output of the first 10 lines of the **lsmod** command.

Managing Kernel Modules

Listing 16.5 Listing Loaded Modules with **lsmod**

```
[root@server1 udev]# lsmod  | head
Module                  Size  Used by
ipt_MASQUERADE         12880  3
xt_CHECKSUM            12549  1
ip6t_rpfilter          12546  1
target_core_pscsi      18810  0
target_core_file       18030  0
target_core_iblock     18177  0
iscsi_target_mod      278732  1
target_core_mod       299412  5 target_core_iblock,target_core_
pscsi,iscsi_target_mod,target_core_file
ip6t_REJECT            12939  2
```

Managing Kernel Modules

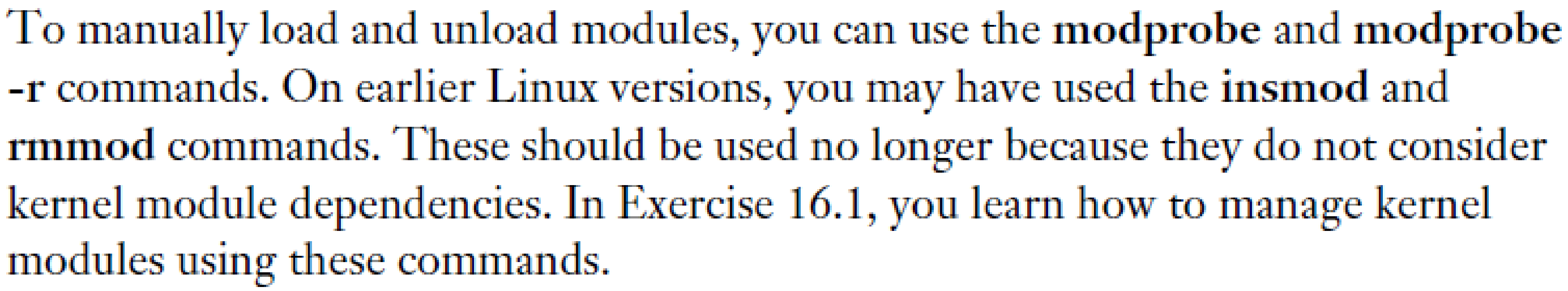
If you want to have more information about a specific kernel module, you can use the **modinfo** command. This gives complete information about the specific kernel modules, including two interesting sections: the alias and the parms. A module alias is another name that can also be used to address the module. The parms lines refer to parameters that can be set while loading the module. (In the section "Managing Kernel Module Parameters" later in this chapter, you learn how to work with kernel module parameters.) Listing 16.6 shows partial output of the **modinfo e1000** command.

Managing Kernel Modules

Listing 16.6 Showing Module Information with **modinfo**

```
[root@server1 udev]# modinfo e1000
filename:       /lib/modules/3.10.0-123.el7.x86_64/kernel/drivers/net/
ethernet/intel/e1000/e1000.ko
version:        7.3.21-k8-NAPI
license:        GPL
description:    Intel(R) PRO/1000 Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
srcversion:     BB8DA267AB1A33D60457C03
alias:          pci:v00008086d00002E6Esv*sd*bc*sc*i*
...
depends:
intree:    Y
vermagic: 3.10.0-123.el17.x86_64 SMP mod_unload modversions
signer:  CentOS Linux kernel signing key
sig_key: BC:83:D0:FE:70:C6:2F:AB:1C:58:B4:EB:AA:95:E3:93:61:28:FC:F4
sig_hashalgo:   sha256
parm:    TxDescriptors:Number of transmit descriptors (array of int)
parm:    RxDescriptors:Number of receive descriptors (array of int)
```

Managing Kernel Modules



To manually load and unload modules, you can use the **modprobe** and **modprobe -r** commands. On earlier Linux versions, you may have used the **insmod** and **rmmod** commands. These should be used no longer because they do not consider kernel module dependencies. In Exercise 16.1, you learn how to manage kernel modules using these commands.

Exercise 16.1 Managing Kernel Modules from the Command Line

In this exercise, you work with the basic commands that are used for managing Linux kernel modules from the command line.

1. Open a root shell and type **lsmod | head**. This shows all kernel modules currently loaded.
2. Type **modprobe ext4** to load the ext4 kernel module. Verify that it is loaded, using the **lsmod** command again.
3. Type **modinfo ext4** to get information about the ext4 kernel module. Notice that it does not have any parameters.
4. Type **modprobe -r ext4** to unload the ext4 kernel module again.
5. Type **modprobe -r xfs** to try to unload the xfs kernel module. Notice that you get an error message as the kernel module currently is in use.

Checking Driver Availability for Hardware Devices

On modern Linux servers, many hardware devices are supported. On occasion, you might find that some devices are not supported properly, though. The best way to find out whether this is the case for your hardware is by using the **lspci** command. If used without arguments, it shows all hardware devices that have been detected on the PCI bus. A very useful argument is **-k**, which lists all kernel modules that are used for the PCI devices that were detected. Listing 16.7 shows sample output of the **lspci -k** command.

Checking Driver Availability for Hardware Devices

Listing 16.7 Checking Kernel Module Availability

```
[root@server1 ~]# lspci -k
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host
bridge (rev 01)
          Subsystem: VMware Virtual Machine Chipset
          Kernel driver in use: agpgart-intel
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP
bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
          Subsystem: VMware Virtual Machine Chipset
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev
01)
          Subsystem: VMware Virtual Machine Chipset
          Kernel driver in use: ata_piix
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
          Subsystem: VMware Virtual Machine Chipset
00:07.7 System peripheral: VMware Virtual Machine Communication
Interface (rev 10)
```

Managing Kernel Module Parameters

You might sometimes want to load kernel modules with specific parameters. If this is the case, you first need to find out which parameter you want to use. If you have found the parameter you want to use, you can load it manually, specifying the name of the parameter followed by the value that you want to assign. To make this an automated procedure, you can create a file in the /etc/modprobe.d directory, where the module is loaded including the parameter you want to be loaded. In Exercise 16.2 you see how to do this using the cdrom kernel module.

Exercise 16.2 Loading Kernel Modules with Parameters

In this exercise, you learn how to work with kernel module parameters.

1. Type **lsmod | grep cdrom**. If you have used the optical drive in your computer, this module should be loaded, and it should also indicate that it is used by the sr_mod module.
2. Type **modprobe -r cdrom**. This will not work because the module is in use by the sr_mod module.
3. Type **modprobe -r sr_mod; modprobe -r cdrom**. This will unload both modules.
4. Type **modinfo cdrom**. This will show information about the cdrom module including the parameters that it supports. One of these is the debug parameter, that supports a Boolean as its value.
5. Now use the command **modprobe cdrom debug=1**. This will load the cdrom module with the debug parameter set to on.

Exercise 16.2 Loading Kernel Modules with Parameters

6. Type **dmesg**. For some kernel module, load information is written to the kernel ring buffer which can be displayed using the **dmesg** command. Unfortunately this is not the case for the cdrom kernel module.
7. Create a file with the name /etc/modprobe.d/cdrom and give it the following contents:

   ```
   options cdrom debug=1
   ```

This will enable the parameter every time the cdrom kernel module will be loaded.

Upgrading the Linux Kernel

From time to time, you need to upgrade the Linux kernel. When you upgrade the Linux kernel, a new version of the kernel is installed and used as the default kernel. The old version of the kernel file will still be available, though. This ensures that your computer can still boot if in the new kernel nonsupported functionality is included. To install a new version of the kernel, you can use the command **yum upgrade kernel**. The **yum install kernel** command also works. Both commands install the new kernel besides the old kernel.

The kernel files for the last four kernels that you have installed on your server will be kept in the /boot directory. The GRUB 2 boot loader automatically picks up all kernels that it finds in this directory.

Summary

In this chapter, you learned how to work with the Linux kernel. You learned that the Linux kernel is modular, and how working with kernel modules is important. You also learned how to manage kernel modules, and how kernel modules are managed automatically while working with new hardware.

Define Key Terms

- Define the following key terms:
  - kernel,
  - module,
  - dmesg,
  - udev,
  - sysfs,
  - proc

Lab 16.1

1. Find out whether a new version of the kernel is available. If so, install it and reboot your computer so that it is used.
2. Use the appropriate command to show recent events that have been logged by the kernel.
3. Locate the kernel module that is used by your network card. Find out whether it has options. Try loading one of these kernel module options manually; if that succeeds, take the required measures to load this option persistently.

Chapter 17:

Configuring a Basic Apache Server

Chapter 17 Objectives

- The following topics are covered in this chapter:
  - Configuring a Basic Apache Server
  - Understanding Apache Configuration files
  - Creating Apache Virtual Hosts

- The following RHCSA exam objectives are covered in this chapter:
  - No RHCSA exam objectives relate directly to Apache

Configuring a Basic Apache Server

Configuring a basic Apache server is not hard to do. It consists of a few easy steps:

1. Install the required software.
2. Identify the main configuration file.
3. Create some web server content.

Installing the Required Software

The Apache server is provided through some different software packages. The basic package is httpd; this package contains everything that is needed for an operational but basic web server. There are some additional packages, as well. For a complete overview, you can use the **yum search http** command and use **yum install httpd** to install the base package.

Instead of using the individual software packages, you can also use yum groups. The **yum groups list** command gives an overview of all yum groups that are available, and the Basic Web Server yum group provides all you need to install the Apache web server and its core requirements. Use **yum groups install "Basic Web Server"** to install it.

Identifying the Main Configuration File

The configuration of the Apache web server goes through different configuration files. The section "Understanding Apache Configuration Files" later in this chapter provides an overview of the way these files are organized. The main Apache configuration file is /etc/httpd/conf/httpd.conf. In this section, many parameters are specified. The most important parameter to understand for setting up a basic web server is the **DocumentRoot** parameter. This parameter specifies the default location where the Apache web server looks for its contents.

Another important configuration parameter is the **ServerRoot**. This defines the default directory where Apache will look for its configuration files. By default, the /etc/httpd directory is used for this purpose, but alternative directories can be used as well. You notice that in the httpd.cond many other configuration files are referred to. The use of additional configuration files makes it easy for applications to install snap-in files that will be included by the Apache server from RPM packages. The names of these configuration files are all relative to the **ServerRoot** /etc/httpd.

Identifying the Main Configuration File

Listing 17.1 Partial Contents of the /etc/httpd/conf/httpd.conf Configuration File

```
[root@server1 ~]# cat /etc/httpd/conf/httpd.conf | grep -v '#'

ServerRoot "/etc/httpd"
Listen 80

Include conf.modules.d/*.conf

User apache
Group apache

ServerAdmin root@localhost

<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

Exercise 17.1 Setting Up a Basic Web Server

1. Type **yum groups install "Basic Web Server"**. This installs the httpd package, and some of the most commonly used additional packages as well.
2. Open the main Apache configuration file with an editor, and look up the line that starts with DocumentRoot. This identifies the location where the Apache server will look for the contents it will service. Confirm that it is set to /var/www/html.
3. In the directory /var/www/html, create a file with the name index.html. In this file, type **"Welcome to my web server"**.
4. To start and enable the web server, type **systemctl start httpd; systemctl enable httpd**. This starts the web server and makes sure that it starts automatically after restarting the server. Use **systemctl status httpd** to check that the web server is up and running. In Listing 17.2 you can see what the result of this command should look like.
5. Type **yum install elinks** to install the elinks text-based browser. Type **elinks http://localhost** to connect to the web server and verify it is working.

Exercise 17.1 Setting Up a Basic Web Server

Listing 17.2 Verifying the Availability of the Apache Web Server with **systemctl status**

```
[root@server2 ~]# systemctl status httpd
httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled)
   Active: active (running) since Sat 2015-05-16 04:27:49 PDT; 1s ago
 Main PID: 42997 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─42997 /usr/sbin/httpd -DFOREGROUND
           ├─42998 /usr/sbin/httpd -DFOREGROUND
           ├─42999 /usr/sbin/httpd -DFOREGROUND
           ├─43000 /usr/sbin/httpd -DFOREGROUND
           ├─43001 /usr/sbin/httpd -DFOREGROUND
           └─43002 /usr/sbin/httpd -DFOREGROUND

May 16 04:27:49 server2.example.com systemd[1]: Started The Apache HTTP
Server.
```

Understanding Apache Configuration Files

A default installation of the Apache web server creates a relatively complex configuration tree in the /etc/httpd directory. Listing 17.3 shows the default contents of this directory. Notice that the contents of this directory may differ on your server if additional software has been installed. Apache is modular, and upon installation of additional Apache modules, different configuration files might be installed here.

Listing 17.3 Default Contents of the /etc/httpd Directory

```
[root@server1 httpd]# \ls -l
total 8
drwxr-xr-x. 2 root root   35 Feb 23 03:12 conf
drwxr-xr-x. 2 root root 4096 Feb 25 12:41 conf.d
drwxr-xr-x. 2 root root 4096 Feb 25 12:41 conf.modules.d
lrwxrwxrwx. 1 root root   19 Feb 17 13:26 logs -> ../../var/log/httpd
lrwxrwxrwx. 1 root root   29 Feb 17 13:26 modules -> ../../usr/lib64/
httpd/modules
lrwxrwxrwx. 1 root root   10 Feb 17 13:26 run -> /run/httpd
```

Understanding Apache Configuration Files

The first thing you notice is the presence of three symbolic links to logs, modules, and a run directory. These are created to allow Apache to be started in a chroot environment.

A chroot environment provides a fake root directory. This is a directory in the file system that is presented as the root directory for the process that is running in the chroot environment. This is done for security reasons: Processes that are running in a chroot environment can access files in that chroot environment only, which decreases the risk of security incidents to happen when intruders manage to get a login shell using the web server identity and try walking through the file system to do unauthorized things.

Creating Apache Virtual Hosts

Many companies host more than one website. Fortunately, it is not necessary to install a new Apache server for every website that you want to run. Apache can be configured to work with virtual hosts. A virtual host is a distinguished Apache configuration file that is created for a unique hostname. When working with virtual hosts, the procedure to access the host is roughly like the following:

Creating Apache Virtual Hosts

1. The client starts a session to a specific virtual host, normally by starting a browser and entering the URL to the website the client wants to use.
2. DNS helps resolving the IP address of the virtual host, which is the IP address of the Apache server that can host different virtual hosts.
3. The Apache process receives requests for all the virtual hosts it is hosting.
4. The Apache process reads the HTTP header to analyze which virtual host this request needs to be forwarded to.
5. Apache reads the specific virtual host configuration file to find which document root is used by this specific virtual host.
6. The request is forwarded to the appropriate contents file in that specific document root.

Exercise 17.2 Installing Apache Virtual Hosts

In this exercise, you create two virtual hosts. To help you setting up virtual hosts, you first set up name resolution, after which you create the virtual hosts configuration as well. Because SELinux has not been discussed yet, you temporarily switch off SELinux.

NOTE I later tell you that you should never switch off SELinux. For once, I make an exception to this important security rule. To focus on what needs to be done on the Apache web server, it is easier to focus just on Apache and not to configure SELinux as well.

Exercise 17.2 Installing Apache Virtual Hosts

1. On both server1 and server2, open the file /etc/hosts with an editor and add two lines that make it possible to resolve the names of the virtual host you are going to create to the IP address of the virtual machine:

```
192.168.122.210    server1.example.com        server1
192.168.122.220    server2.example.com        server2
192.168.122.210    account.example.com        account
12.168.122.210     sales.example.com          sales
```

2. On server1, open a root shell and add the following to the /etc/httpd/conf/httpd.conf file. (You can leave all other settings as they are.)

```
<Directory/www/docs>
          Required all granted
          AllowOverride None
</Directory>
```

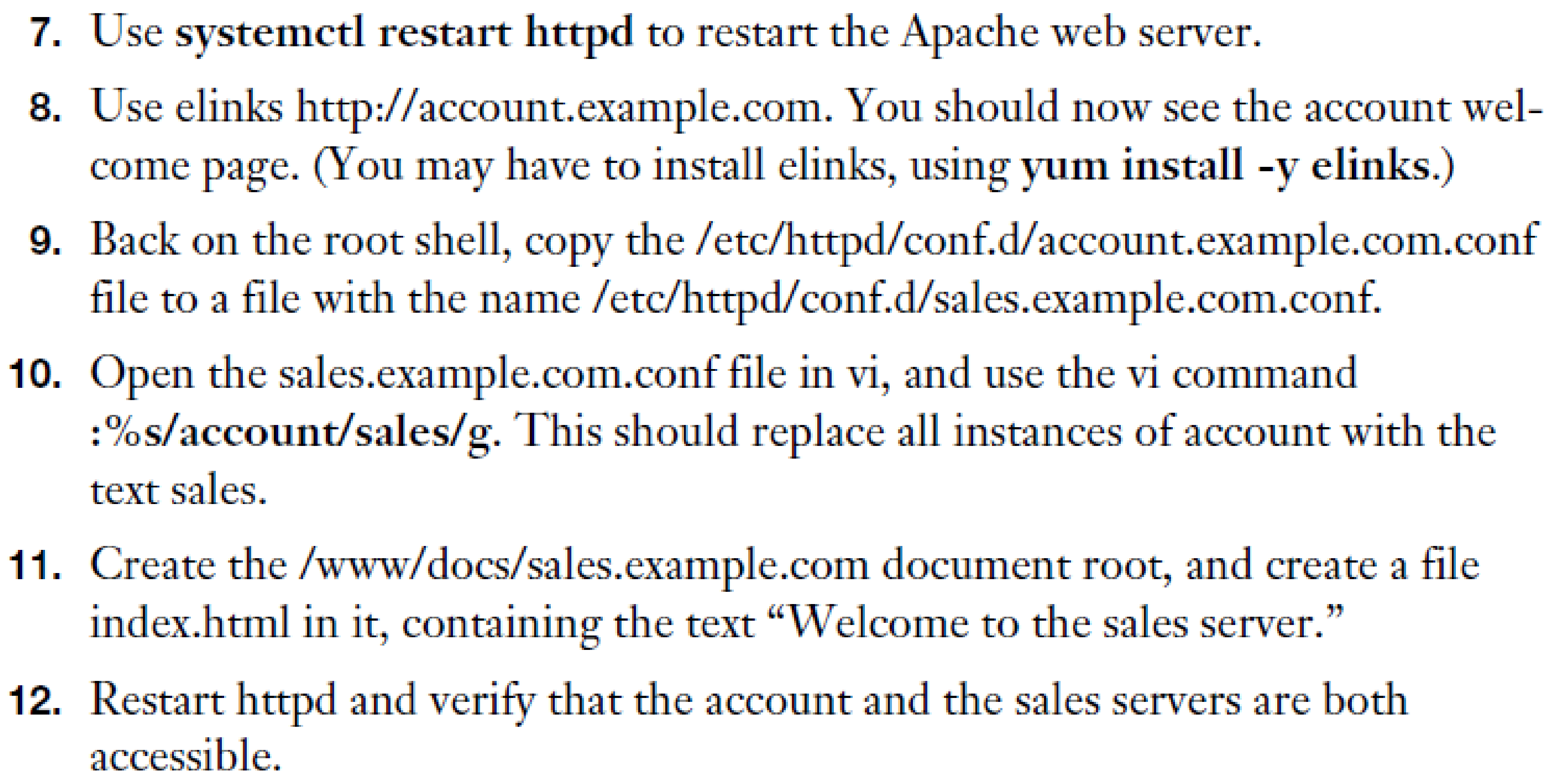
Exercise 17.2 Installing Apache Virtual Hosts

3. On server1, open a root shell and create a configuration file with the name **account.example.com.conf** in the directory /etc/httpd/conf.d. Give this file the following content:

```
<VirtualHost *:80>
        ServerAdmin webmaster@account.example.com
        DocumentRoot /www/docs/account.example.com
        ServerName account.example.com
        ErrorLog logs/account/example.com-error_log
        CustomLog logs/account.example.com-access_log common
</VirtualHost>
```

4. Close the configuration file and from the root shell use **mkdir -p /www/docs/account.example.com**.
5. Create a file with the name index.html in the account document root, and make sure its contents read "Welcome to account."
6. Temporarily switch off SELinux using **setenforce 0**.

Exercise 17.2 Installing Apache Virtual Hosts



7. Use **systemctl restart httpd** to restart the Apache web server.
8. Use elinks http://account.example.com. You should now see the account welcome page. (You may have to install elinks, using **yum install -y elinks**.)
9. Back on the root shell, copy the /etc/httpd/conf.d/account.example.com.conf file to a file with the name /etc/httpd/conf.d/sales.example.com.conf.
10. Open the sales.example.com.conf file in vi, and use the vi command **:%s/account/sales/g**. This should replace all instances of account with the text sales.
11. Create the /www/docs/sales.example.com document root, and create a file index.html in it, containing the text “Welcome to the sales server.”
12. Restart httpd and verify that the account and the sales servers are both accessible.

Summary

In this chapter, you learned about Apache basics. The information in this chapter helps you configure a basic Apache web server, which helps testing advanced topics like firewall configuration or SELinux configuration that are covered in later chapters in this book.

Define Key Terms

- Define the following key terms:
  - DocumentRoot,
  - virtual hosts,
  - chroot

Lab 17.1

1. Install the required packages that allow you to run a basic web server. Make sure that the web server process is started automatically when your server reboots. Do *not* use any virtual server.
2. Make sure the web server presents a default page showing "Welcome to my web server."
3. Use elinks to test the working of your web server.
4. Use **yum install httpd-manual** to install the Apache documentation.
5. Use a browser to test access to the /manual web page on your server.

Chapter 18:

Managing and Understanding the Boot Procedure

Chapter 18 Objectives

- The following topics are covered in this chapter:
  - Working with Systemd
  - Working with GRUB 2
- The following RHCSA exam objectives are covered in this chapter:
  - Start and stop services and configure services to start automatically at boot
  - Configure systems to boot into a specific target automatically
  - Modify the system bootloader

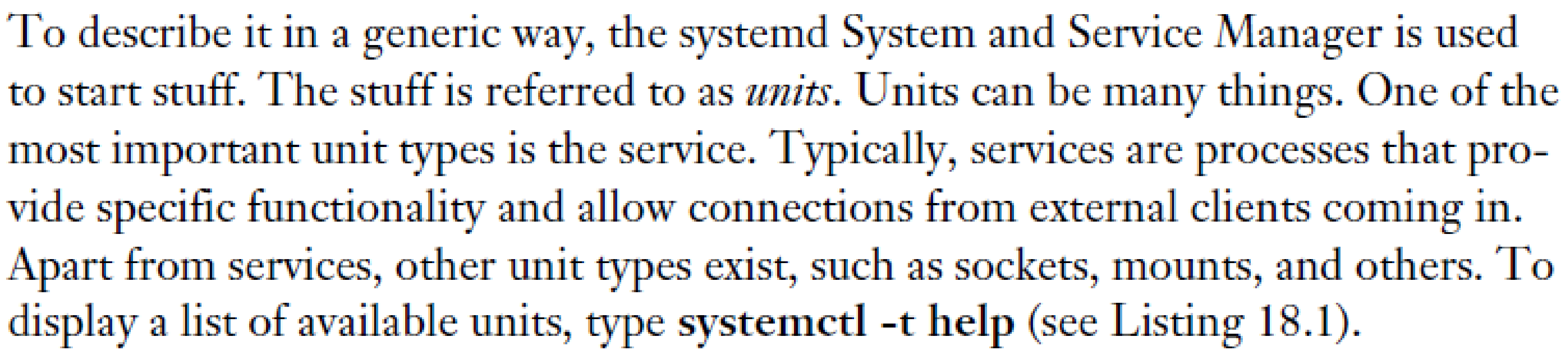
Managing and Understanding the Boot Procedure

In this chapter, you learn how the boot procedure on Red Hat Enterprise Linux is organized. We first go through a section about systemd, the overall service that takes care of starting everything on your server. In this section, you also learn how systemd targets are used to group systemd units and come to a final operational environment.

Working with *Systemd*

Systemd is the new service in Red Hat Enterprise Linux 7 that is responsible for starting all kinds of things. Systemd goes way beyond starting services; other items are started from systemd as well. In this chapter, you learn how systemd is organized and what items are started from systemd.

Understanding *Systemd*



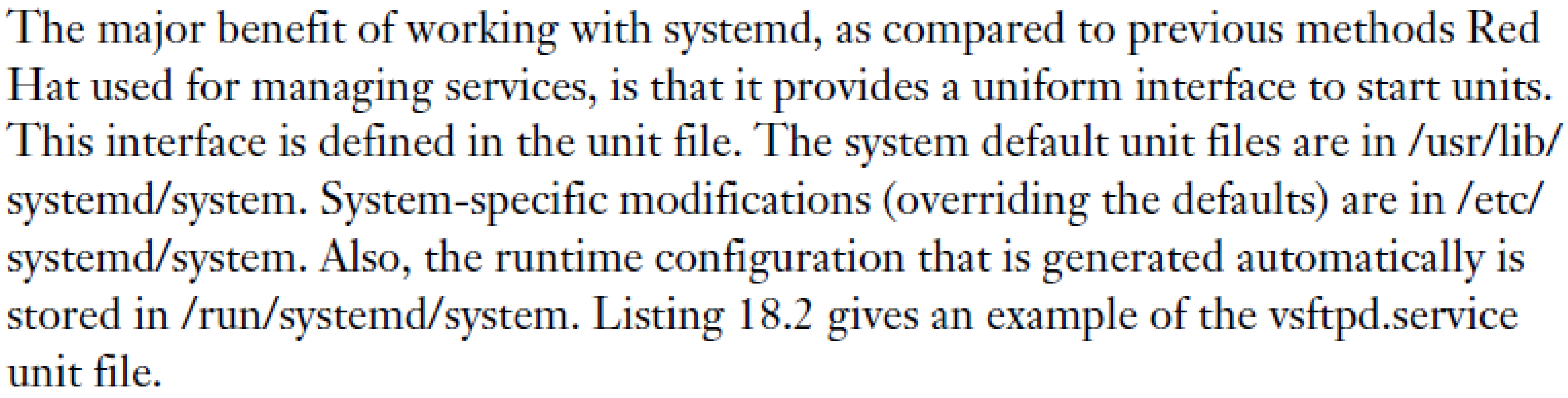
To describe it in a generic way, the systemd System and Service Manager is used to start stuff. The stuff is referred to as *units*. Units can be many things. One of the most important unit types is the service. Typically, services are processes that provide specific functionality and allow connections from external clients coming in. Apart from services, other unit types exist, such as sockets, mounts, and others. To display a list of available units, type **systemctl -t help** (see Listing 18.1).

Understanding *Systemd*

Lising 18.1 Unit Types in Systemd

```
[root@server1 ~]# systemctl -t help
Available unit types:
service
socket
target
device
mount
automount
snapshot
timer
swap
path
slice
scope
```

Understanding Service Units



The major benefit of working with systemd, as compared to previous methods Red Hat used for managing services, is that it provides a uniform interface to start units. This interface is defined in the unit file. The system default unit files are in /usr/lib/systemd/system. System-specific modifications (overriding the defaults) are in /etc/systemd/system. Also, the runtime configuration that is generated automatically is stored in /run/systemd/system. Listing 18.2 gives an example of the vsftpd.service unit file.

Understanding Service Units

Listing 18.2 Example of the Vsftpd Unit File

```
[Unit]
Description=Vsftpd ftp daemon
After=network.target

[Service]
Type=forking
ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

[Install]
WantedBy=multi-user.target
```

From this unit file example, you can see that it is relatively easy to understand. Any systemd service unit file consists of three sections. (You'll find different sections in other types of unit files.)

Understanding Service Units

- **[Unit]**, which describes the unit and defines dependencies. This section also contains the important **After** statement, and optionally the **Before** statement. These statements define dependencies between different units. The Before statement relates to another unit that is started after this unit. The after unit refers to a unit that needs to be started before this unit can be started.
- **[Service]**, in which there is a description on how to start and stop the service and request status installation. Normally, you can expect an ExecStart line, which indicates how to start the unit, or an ExecStop line, which indicates how to stop the unit.
- **[Install]**, in which the wants are taken care of. You'll read more about this in the next section, "Understanding Target Units."

Understanding Service Units

Listing 18.3 Example of a Mount Unit File

```
[Unit]
Description=Temporary Directory
Documentation=man:hier(7)
Documentation=http://www.freedesktop.org/wiki/Software/systemd/
  APIFileSystems
DefaultDependencies=no
Conflicts=umount.target
Before=local-fs.target umount.target

[Mount]
What=tmpfs
Where=/tmp
Type=tmpfs
Options=mode=1777,strictatime

# Make 'systemctl enable tmp.mount' work:
[Install]
WantedBy=local-fs.target
```

Understanding Service Units

When working with systemd unit files, you risk getting overwhelmed with options. Every unit file can be configured with different options. To figure out which options are available for a specific unit, use the **systemctl show** command. For instance, the **systemctl show sshd** command shows all systemd options that can be configured in the sshd.service unit, including their current default values. Listing 18.5 shows the output of this command.

Understanding Target Units

The unit files are used to build the functionality that is needed on your server. To make it possible to load them in the right order and at the right moment, a specific type of unit is used: the target unit. A simple definition of a target unit is “a group of units.” Some targets are used as the equivalents to the old run levels, which on earlier versions of RHEL were used to define the state a server should be started in. A run level was a collection of services that were needed for a server to be started in multi-user mode or in graphical mode. Targets go beyond that. A good starting point to understanding targets is to see them as a group of units.

Targets by themselves can have dependencies to other targets, which are defined in the target unit. Let’s take a look at Listing 18.6, where you can see the definition of the multi-user.target, which defines the normal operational state of an RHEL server.

Understanding Target Units

Listing 18.6 The Multi-user.target File

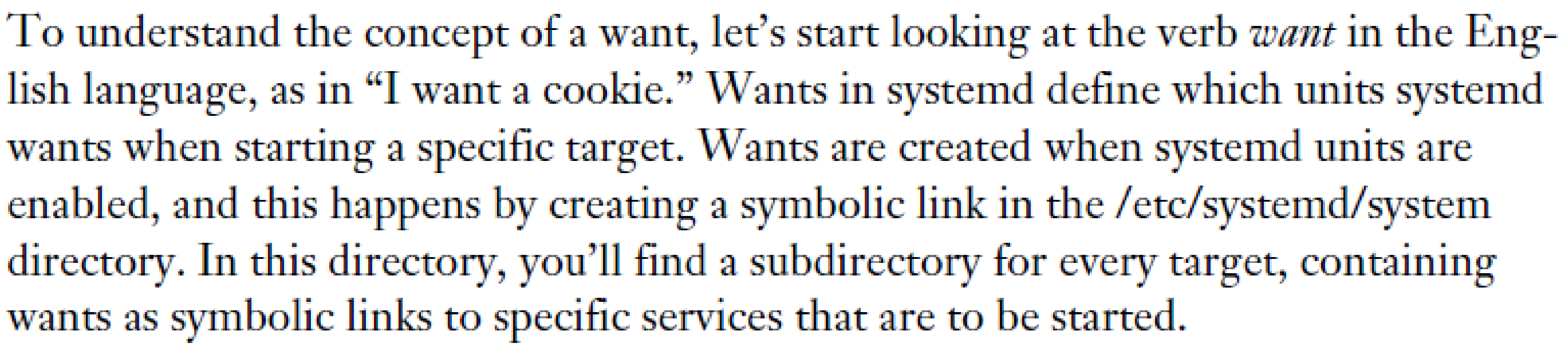
```
[root@server202 system]# cat multi-user.target
...

[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes

[Install]
Alias=default.target
```

Understanding Target Units

You can see that by itself the target unit does not contain much. It just defines what it requires and which services and targets it cannot coexist with. It also defines load ordering, by using the **After** statement in the Unit section. And you can see that in the Install section it is defined as the default.target, so this is what your server starts by default. The target file does not contain any information about the units that should be included; that is in the individual unit files and the wants (explained in the upcoming section “Understanding Wants”).

Even if a systemd target looks a bit like the old run levels, it is more than that. A target is a group of units, and there are multiple different targets. Some targets, such as the multi-user.target and the graphical.target, define a specific state that the system needs to enter. Other targets just bundle a group of units together, such as the nfs.target and the printer.target. These targets are included from other targets, like the multi-user or graphical targets.

Understanding Wants



To understand the concept of a want, let's start looking at the verb *want* in the English language, as in "I want a cookie." Wants in systemd define which units systemd wants when starting a specific target. Wants are created when systemd units are enabled, and this happens by creating a symbolic link in the /etc/systemd/system directory. In this directory, you'll find a subdirectory for every target, containing wants as symbolic links to specific services that are to be started.

Managing Units Through *Systemd*

As an administrator, you need to manage systemd units. It starts by starting and stopping units. You use the **systemctl** command to do that. In Exercise 18.1, you walk start, stop, and manage a unit. After you have configured a unit so that it can be started without problems, you need to make sure that it restarts automatically upon reboot. You do this by enabling or disabling the unit.

Exercise 18.1 Managing Units with systemctl

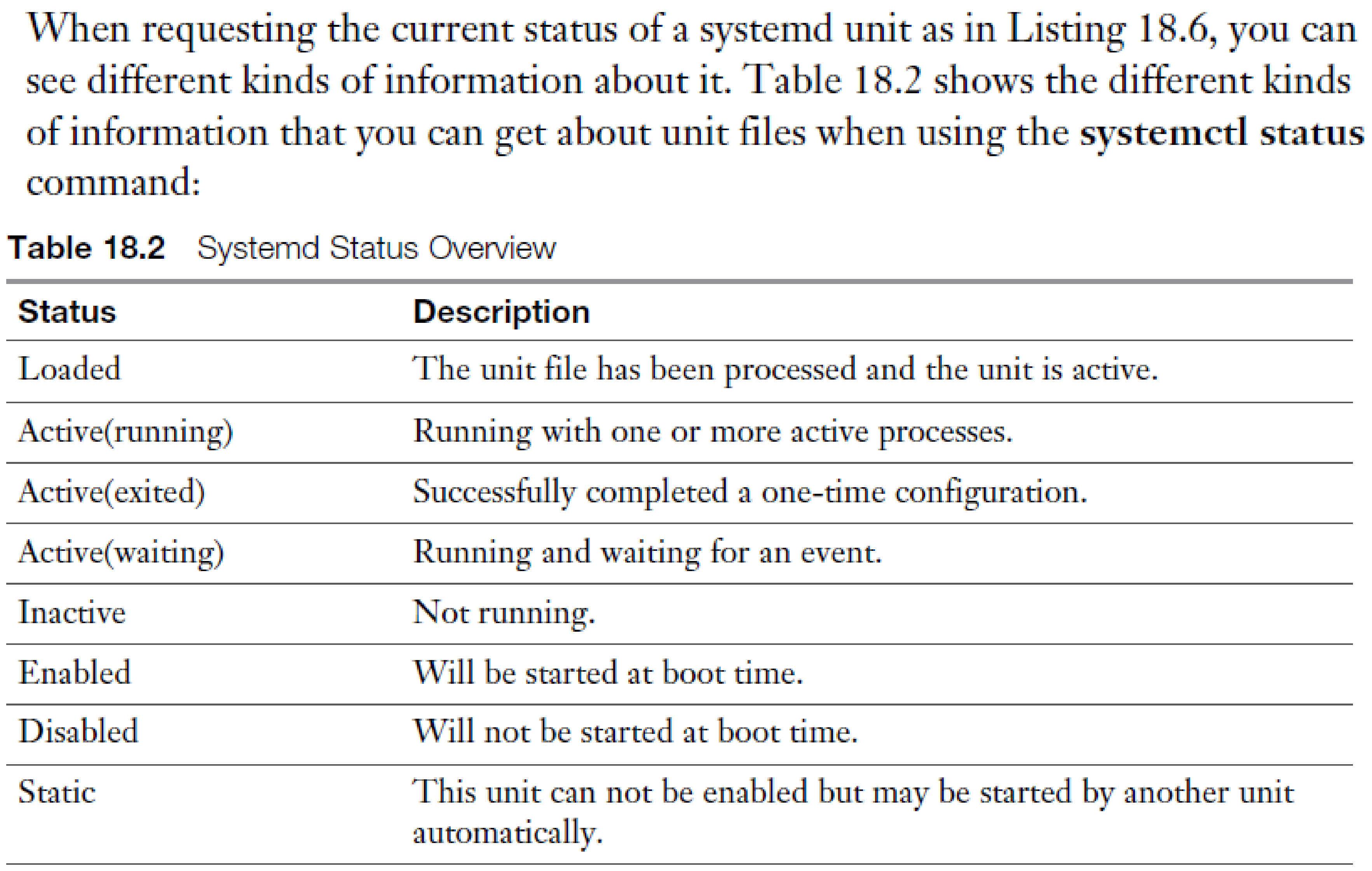
1. Type **yum -y install vsftpd** to install the Very Secure FTP service.
2. Type **systemctl start vsftpd**. This activates the FTP server on your machine.
3. Type **systemctl status vsftpd**. You'll get an output as in Listing 18.7 and see that the vsftpd service is currently operational. You can also see in the Loaded line that it is currently disabled, which means that it will not be activated on a system restart.
4. Type **systemctl enable vsftpd**. This creates a symbolic link in the wants directory for the multi-user target to ensure that the service gets back after a restart.
5. Type **systemctl status vsftpd** again. You'll now see that the unit file has changed from being disabled to enabled.

Exercise 18.1 Managing Units with systemctl

Listing 18.7 Requesting Current Unit Status with **systemctl status**

```
[root@server202 system]# systemctl status vsftpd
vsftpd.service - Vsftpd ftp daemon
   Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled)
   Active: active (running) since Sun 2014-09-28 08:42:59 EDT; 2s ago
  Process: 34468 ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf
(code=exited, status=0/SUCCESS)
 Main PID: 34469 (vsftpd)
   CGroup: /system.slice/vsftpd.service
           └─34469 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

Sep 28 08:42:59 server202.example.com systemd[1]: Starting Vsftpd ftp
daemon...
```

Managing Units with *systemctl*



When requesting the current status of a systemd unit as in Listing 18.6, you can see different kinds of information about it. Table 18.2 shows the different kinds of information that you can get about unit files when using the **systemctl status** command:

Table 18.2 Systemd Status Overview

Status	Description
Loaded	The unit file has been processed and the unit is active.
Active(running)	Running with one or more active processes.
Active(exited)	Successfully completed a one-time configuration.
Active(waiting)	Running and waiting for an event.
Inactive	Not running.
Enabled	Will be started at boot time.
Disabled	Will not be started at boot time.
Static	This unit can not be enabled but may be started by another unit automatically.

Managing Units with *systemctl*

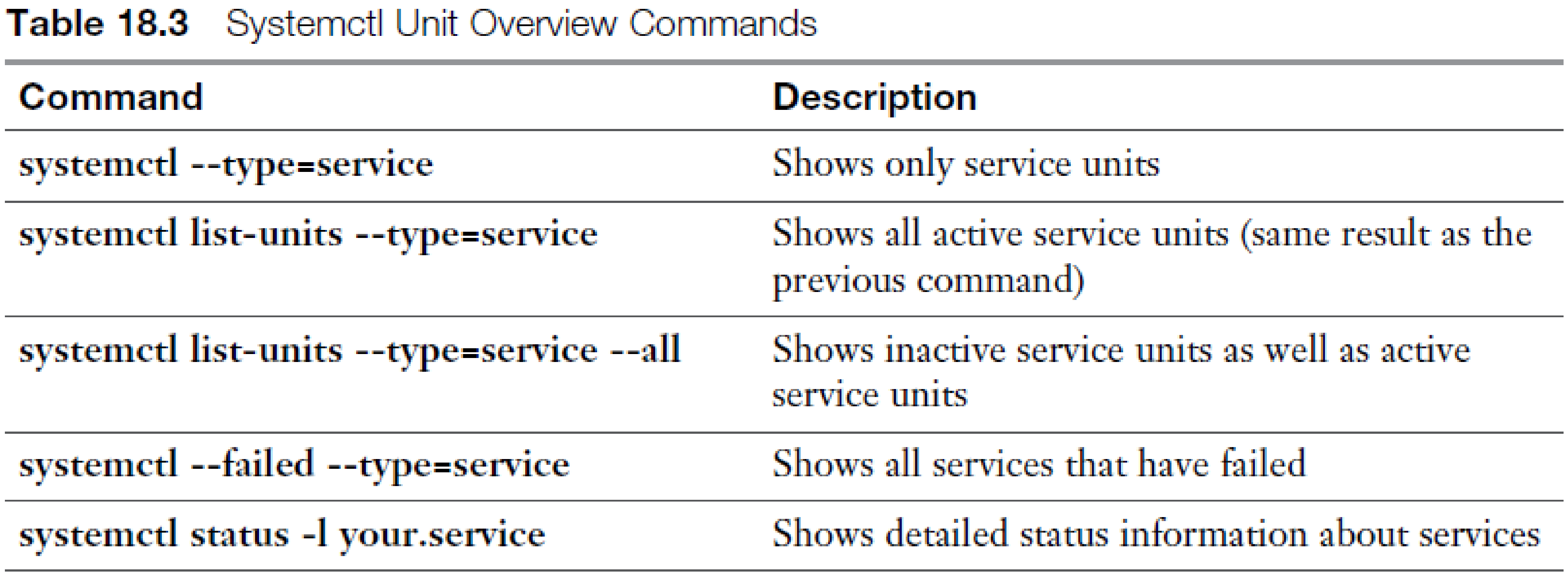
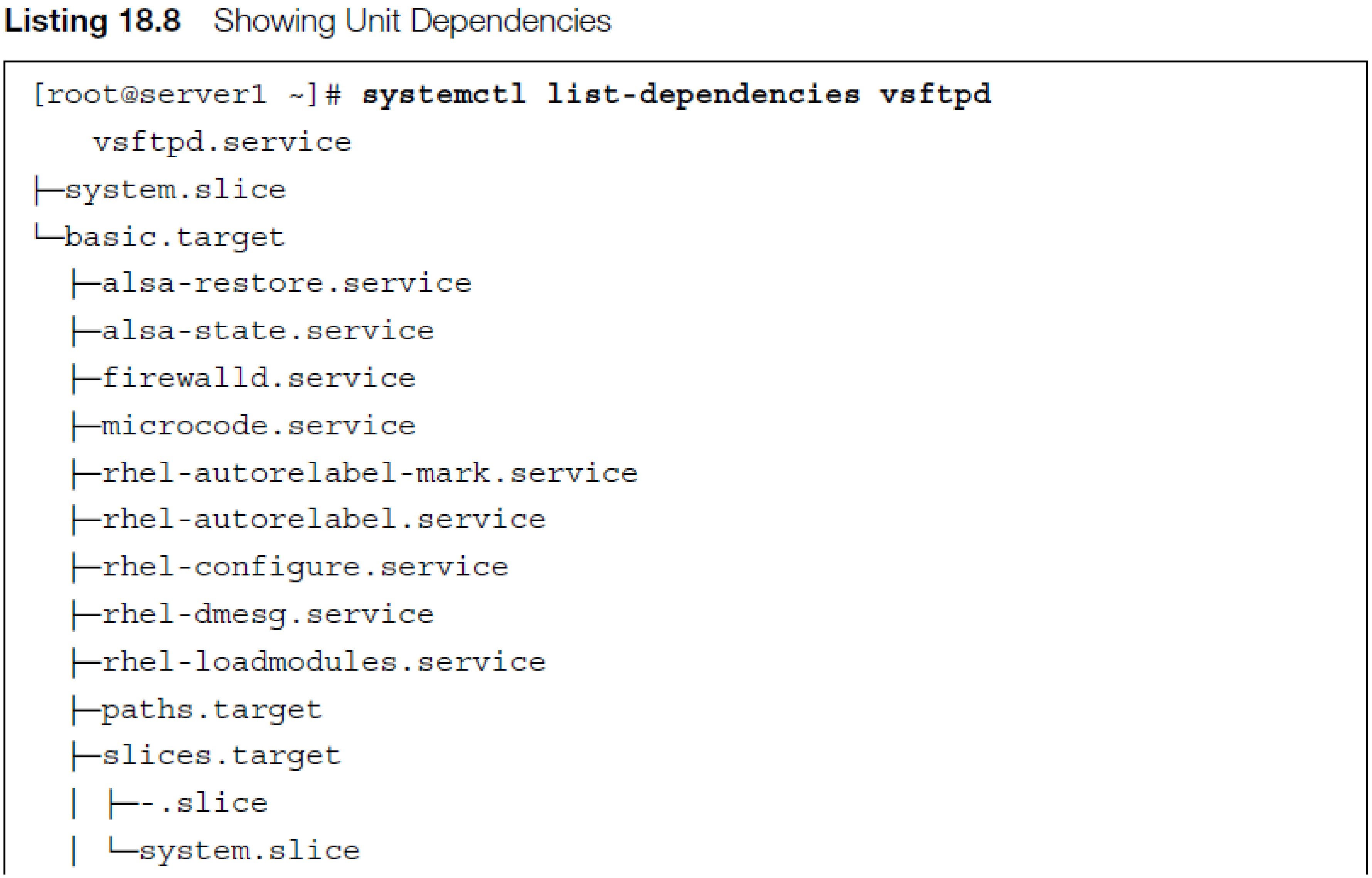


Table 18.3 Systemctl Unit Overview Commands

Command	Description
systemctl --type=service	Shows only service units
systemctl list-units --type=service	Shows all active service units (same result as the previous command)
systemctl list-units --type=service --all	Shows inactive service units as well as active service units
systemctl --failed --type=service	Shows all services that have failed
systemctl status -l your.service	Shows detailed status information about services

Managing Units with *systemctl*



Listing 18.8 Showing Unit Dependencies

```
[root@server1 ~]# systemctl list-dependencies vsftpd
   vsftpd.service
├─system.slice
└─basic.target
  ├─alsa-restore.service
  ├─alsa-state.service
  ├─firewalld.service
  ├─microcode.service
  ├─rhel-autorelabel-mark.service
  ├─rhel-autorelabel.service
  ├─rhel-configure.service
  ├─rhel-dmesg.service
  ├─rhel-loadmodules.service
  ├─paths.target
  ├─slices.target
  │ ├─-.slice
  │ └─system.slice
```

Managing Systemd Targets

As an administrator, you need to make sure that the required services are started when your server boots. To do this, use the **systemctl enable** and **systemctl disable** commands. You do not have to think about the specific target a service has to be started in. The services know for themselves in which targets they need to be started and a want is created automatically in that target. The following procedure walks you through the steps of enabling a service:

Managing Systemd Targets

1. Type **systemctl status vsftpd**. If the service has not yet been enabled, the Loaded line will show that it currently is disabled:

```
[root@server202 ~]# systemctl status vsftpd
vsftpd.service - Vsftpd ftp daemon
    Loaded: loaded (/usr/lib/systemd/system/vsftpd.service;
  disabled)
    Active: inactive (dead)
```

2. Type **ls /etc/systemd/system/multi-user.target.wants**. You'll see symbolic links that are taking care of starting the different services on your machine. You can also see that the vsftpd.service link does not exist.

3. Type **systemctl enable vsftpd**. The command shows you that it is creating a symbolic link for the file /usr/lib/systemd/system/vsftpd.service to the directory /etc/systemd/system/multi-user.target.wants. So basically, when you enable a systemd unit file, on the background a symbolic link is created.

Isolating Targets

As already discussed, on systemd machines there are a couple of targets. You also know that a target is a collection of units. Some of those targets have a special role because they can be isolated. By isolating a target, you start that target with all of its dependencies. Not all targets can be isolated, but only targets that have the isolate option enabled. We'll explore the **systemctl isolate** command in a while. Before doing that, let's take a look at the default targets on your computer.

To get a list of all targets currently loaded, type **systemctl --type=target**. You'll see a list of all the targets currently active. If your server is running a graphical environment, this will include all the dependencies required to install the graphical.target also. However, this list does not show all the targets, but only the active targets. Type **systemctl --type=target --all** for an overview of all targets that exist on your computer. You'll now see inactive targets also.

Isolating Targets

Of the targets on your system, a few have an important role because they can be started (isolated) to determine the state your server starts in. These are also the targets that can be set as the default target. These targets also roughly correspond to run levels as they were used on earlier versions of RHEL. These are the following targets:

- poweroff.target - runlevel 0
- rescue.target - runlevel 1
- multi-user.target - runlevel 3
- graphical.target - runlevel 5
- reboot.target - runlevel 6

Exercise 18.2 Isolating Targets

1. From a root shell, go to the directory /usr/lib/systemd/system. Type **grep Isolate *.target**. This shows a list of all targets that allow isolation.
2. Type **systemctl isolate rescue.target**. This switches your computer to rescue.target. You need to type the root password on the console of your server to log in.
3. Type **systemctl isolate reboot.target**. This restarts your computer.

Setting the Default Target

Setting the default target is an easy procedure that can be accomplished from the command line. Type **systemctl get-default** to see the current default target and use **systemctl set-default** to set the desired default target.

To set the graphical target as the default target, you need to make sure that the required packages are installed. If this is not the case, you can use the **yum group list** command to show a list of all RPM package groups. The "server with gui" and "GNOME Desktop" package groups both apply. Use **yum group install "server with gui"** to install all GUI packages on a server where they have not been installed yet.

Working with GRUB 2

The GRUB 2 boot loader is one of the first things that needs to be working well to boot a Linux server. As an administrator, you will sometimes need to apply modifications to the GRUB 2 boot loader configuration. This section explains how to do so. The RHEL 7 boot procedure is discussed in more detail in Chapter 19, where troubleshooting topics are covered as well.

Understanding GRUB 2

The GRUB 2 boot loader makes sure that you can boot Linux. GRUB 2 is installed in the boot sector of your server's hard drive and is configured to load a Linux kernel and the initramfs:

- The kernel is the heart of the operating system, allowing users to interact with the hardware that is installed in the server.
- The initramfs contains drivers that are needed to start your server. It contains a mini file system that is mounted during boot. In it are kernel modules that are needed during the rest of the boot process (for example, the LVM modules and SCSI modules for accessing disks that are not supported by default).

Understanding GRUB 2

Normally, GRUB 2 works just fine and does not need much maintenance. In some cases, though, you might have to change its configuration. To apply changes to the GRUB 2 configuration, the starting point is the /etc/default/grub file. In this file, you'll find options that tell GRUB what to do and how to do it. Listing 18.9 shows the contents of this file after an installation with default settings of CentOS 7.

Listing 18.9 Contents of the /etc/default/grub File

```
[root@server202 grub.d]# cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/swap vconsole.font=latarcyrheb-
  sun16 rd.lvm.lv=centos/root crashkernel=auto  vconsole.keymap=us
  rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
```

Understanding GRUB 2

Listing 18.10 Partial Contents of the /boot/grub2/grub.cfg Configuration File

```
menuentry 'CentOS Linux (3.10.0-229.1.2.el7.x86_64) 7 (Core)' --class
centos --class gnu-linux --class gnu --class os --unrestricted
$menuentry_id_option 'gnulinux-3.10.0-123.el7.x86_64-advanced-50faa2a1-
01d3-430b-8114-4a98daf5bdb9' {
      load_video
      set gfxpayload=keep
      insmod gzio
      insmod part_msdos
      insmod xfs
      set root='hd0,msdos1'
      if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1
--hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1'
057ba3d8-bfe7-4676-bb99-79e9980a1966
     else
       search --no-floppy --fs-uuid --set=root 057ba3d8-bfe7-4676-bb99-
79e9980a1966
     fi
     linux16 /vmlinuz-3.10.0-229.1.2.el7.x86_64 root=/dev/mapper/
centos-root ro rd.lvm.lv=centos/swap vconsole.font=latarcyrheb-sun16
rd.lvm.lv=centos/root crashkernel=auto  vconsole.keymap=us rhgb quiet
LANG=en_US.UTF-8
     initrd16 /initramfs-3.10.0-229.1.2.el7.x86_64.img
}
```

Modifying Default GRUB 2 Boot Options

To apply modifications to the GRUB 2 boot loader, the file /etc/default/grub is your entry point; do not change the contents of the /boot/grub2/grub.cfg configuration file directly. The most important line in this file is GRUB_CMDLINE_LINUX, which defines how the Linux kernel should be started. In this line, you can apply permanent fixes to the GRUB 2 configuration. Some likely candidates for removal are the options **rhgb** and **quiet**. These options tell the kernel to hide all output while booting. That is nice to hide confusing messages for end users, but if you are a server administrator, you probably just want to remove these options.

Another interesting parameter is GRUB_TIMEOUT. This defines the amount of time your server waits for you to access the GRUB 2 boot menu before it continues booting automatically. If your server runs on physical hardware that takes a long time to get through the BIOS checks, it may be interesting to increase this time a bit.

Modifying Default GRUB 2 Boot Options

While working with GRUB 2, you need to know a bit about kernel boot arguments. There are many of them, and most of them you'll never use, but it is good to know where you can find them. Type **man 7 bootparam** for a man page that contains an excellent description of all boot parameters that you may use while starting the kernel.

Exercise 18.3 Applying Modifications to GRUB2

In this exercise you'll apply some changes to the GRUB2 boot configuration and write them to the /boot/grub2/grub.cfg configuration file.

1. Open the file /etc/default/grub with an editor and remove the **rhgb** and **quiet** options from the GRUB_CMDLINE_LINUX line.
2. From the same file, set the GRUB_TIMEOUT parameter to 10 seconds. Save changes to the file and close the editor.
3. From the command line, type **grub2-mkconfig > /boot/grub2/grub.cfg** to write the changes to GRUB 2. (Note that instead of using the redirector > to write changes to the grub.cfg file, you could use the **-o** option. Both methods have the same result.)
4. Reboot and verify that while booting you see boot messages scrolling by.

Summary

In this chapter you learned how systemd and GRUB 2 are used to bring your server into the exact state you desire at the end of the boot procedure. You also learned how systemd is organized, and also how units can be configured for automatic start with the use of targets. You also read how to apply changes to the default GRUB 2 boot loader. In the next chapter, you learn how to troubleshoot the boot procedure and fix some common problems.

Define Key Terms

- Define the following key terms:
  - unit,
  - wants,
  - target,
  - systemd,
  - dependencies,
  - kernel,
  - boot loader,
  - GRUB

Lab 18.1

- Make sure that the firewalld service is started on boot. Also make sure that the iptables service can never be started at the same time.

- Change your GRUB 2 boot configuration so that you will see boot messages upon startup.

Chapter 19:

Essential Boot Procedure Troubleshooting

Chapter 19 Objectives

- The following topics are covered in this chapter:
  - Understanding the RHEL 7 Boot Procedure
  - Passing Kernel Boot Arguments
  - Using a Rescue Disk
  - Fixing Common Issues
  - Recovering Access to a Virtual Machine
- The following RHCSA exam objectives are covered in this chapter:
  - Boot systems into different targets manually
  - Interrupt the boot process in order to gain access to a system

Troubleshooting the Boot Procedure

In the preceding chapter, you learned how an RHEL 7 server boots and which role the boot loader GRUB 2 and systemd play in that. In this chapter, you learn what you can do when common problems occur while booting your server. This chapter teaches general approaches that help to fix some of the most common problems that may occur while booting. Make sure to master the topics discussed in this chapter well; they might save your (professional) life one day!

Understanding the RHEL 7 Boot Procedure

To fix boot issues, it is essential to have a good understanding of the boot procedure. If issues occur during boot, you need to be able to judge in which phase of the boot procedure the issue occurs so that you can select the appropriate tool to fix the issue.

The following steps summarize how the boot procedure happens on Linux.

1. **Performing POST:** The machine is powered on. From the system firmware, which can be the modern Universal Extended Firmware Interface (UEFI) or the classical Basic Input Output System (BIOS), the Power-On Self-Test (POST) is executed, and the hardware that is required to start the system is initialized.
2. **Selecting the bootable device:** Either from the UEFI boot firmware or from the Master Boot Record, a bootable device is located.

Understanding the RHEL 7 Boot Procedure

3. **Loading the boot loader:** From the bootable device, a boot loader is located. On Red Hat, this is usually GRUB 2.

4. **Loading the kernel:** The boot loader may present a boot menu to the user, or can be configured to automatically start a default operating system. To load Linux, the kernel is loaded together with the initramfs. The initramfs contains kernel modules for all hardware that is required to boot, as well as the initial scripts required to proceed to the next stage of booting. On RHEL 7, the initramfs contains a complete operational system (which may be used for troubleshooting purposes).

5. **Starting /sbin/init:** Once the kernel is loaded into memory, the first of all processes is loaded, but still from the initramfs. This is the /sbin/init process, which on Red Hat is linked to systemd. The udev daemon is loaded as well to take care of further hardware initialization. All this is still happening from the initramfs image.

Understanding the RHEL 7 Boot Procedure

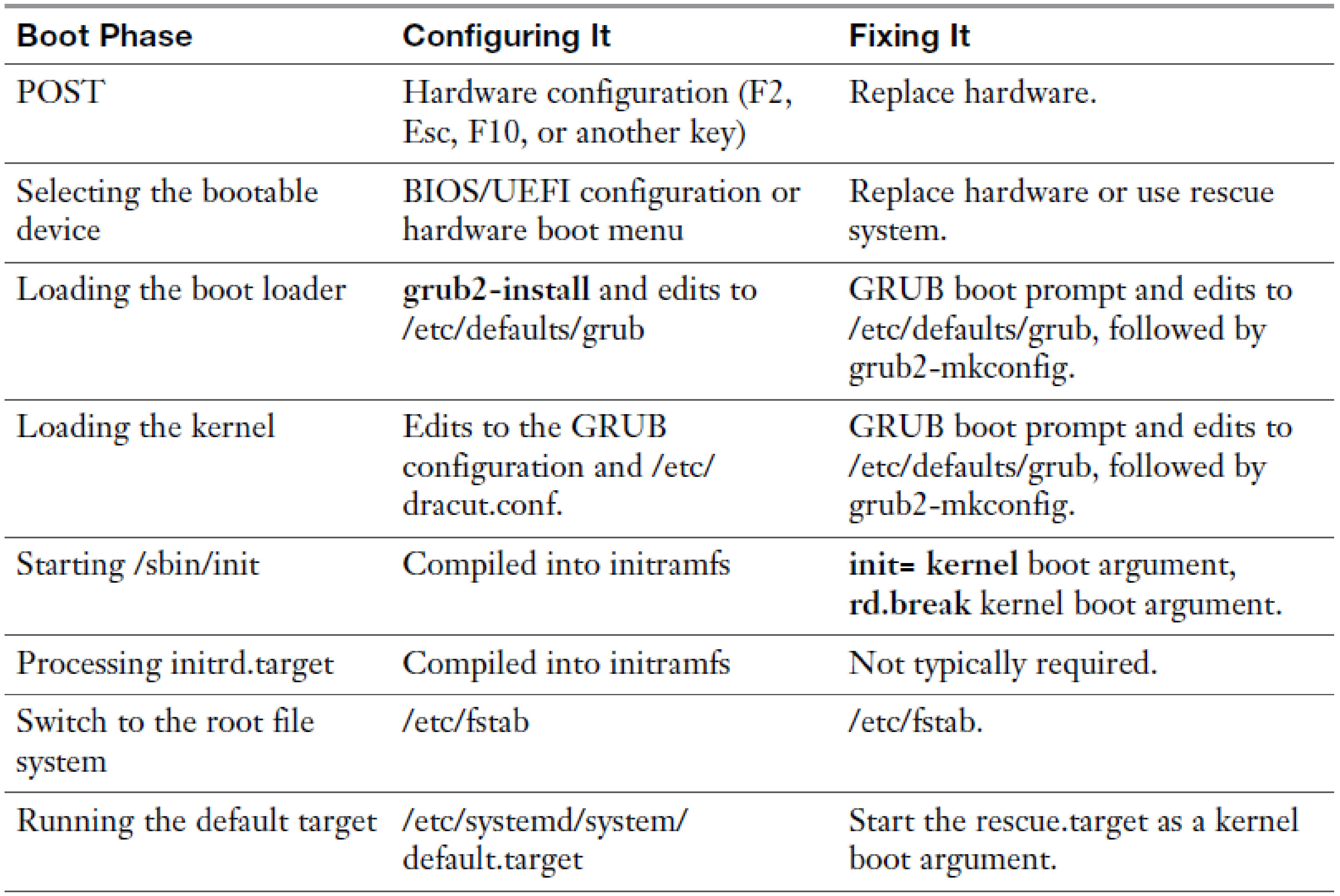
6. **Processing initrd.target:** The systemd process executes all units from the initrd.target, which prepares a minimal operating environment, where the root file system on disk is mounted on the /sysroot directory. At this point, enough is loaded to pass to the system installation that was written to the hard drive.

7. **Switching to the root file system:** The system switches to the root file system that is on disk and at this point can load the systemd process from disk as well.

8. **Running the default target:** Systemd looks for the default target to execute and runs all of its units. In this process, a login screen is presented, and the user can authenticate. Notice that the login prompt can be prompted before all systemd unit files have been loaded successfully. So, seeing a login prompt does not necessarily mean that your server is fully operational yet.

Boot Phase Configuration and Troubleshooting Overview

Table 19.2 Boot Phase Configuration and Troubleshooting Overview

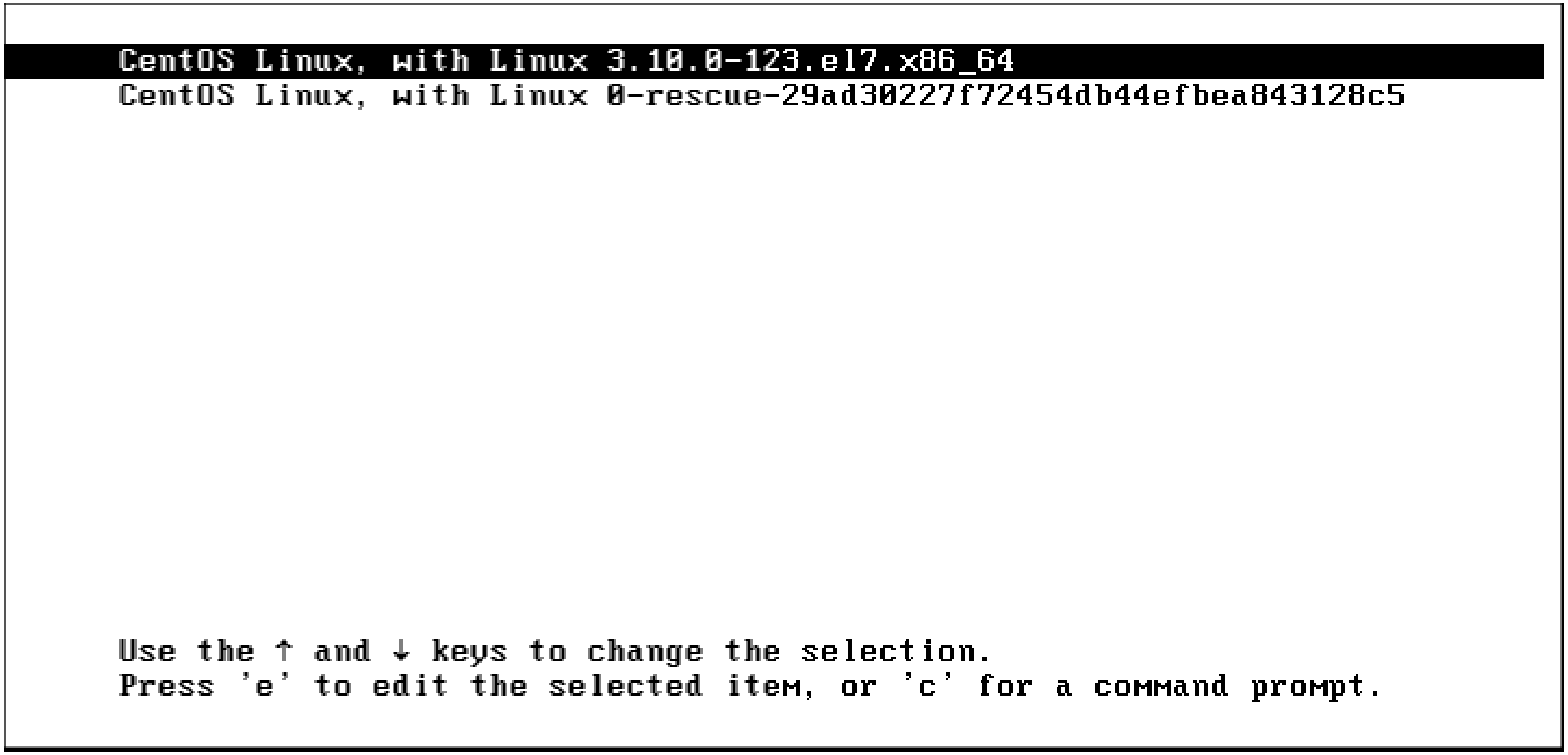


Boot Phase	**Configuring It**	**Fixing It**
POST	Hardware configuration (F2, Esc, F10, or another key)	Replace hardware.
Selecting the bootable device	BIOS/UEFI configuration or hardware boot menu	Replace hardware or use rescue system.
Loading the boot loader	**grub2-install** and edits to /etc/defaults/grub	GRUB boot prompt and edits to /etc/defaults/grub, followed by grub2-mkconfig.
Loading the kernel	Edits to the GRUB configuration and /etc/dracut.conf.	GRUB boot prompt and edits to /etc/defaults/grub, followed by grub2-mkconfig.
Starting /sbin/init	Compiled into initramfs	**init= kernel** boot argument, **rd.break** kernel boot argument.
Processing initrd.target	Compiled into initramfs	Not typically required.
Switch to the root file system	/etc/fstab	/etc/fstab.
Running the default target	/etc/systemd/system/default.target	Start the rescue.target as a kernel boot argument.

Passing Kernel Boot Arguments

If your server does not boot normally, the GRUB boot prompt offers a convenient way to stop the boot procedure and pass specific options to the kernel while booting. In this section, you learn how to access the boot prompt and how to pass specific boot arguments to the kernel while booting.

Accessing the Boot Prompt



```
      CentOS Linux, with Linux 3.10.0-123.el7.x86_64
      CentOS Linux, with Linux 0-rescue-29ad30227f72454db44efbea843128c5




      Use the ↑ and ↓ keys to change the selection.
      Press 'e' to edit the selected item, or 'c' for a command prompt.
```

Figure 19.1 Entering the GRUB boot prompt.

Accessing the Boot Prompt

After passing an **e** to the GRUB boot menu, you'll see the interface that is in Figure 19.2. From this interface, scroll down to locate the section that begins with linux16 /vmlinuz followed by a lot of arguments. This is the line that tells GRUB how to start a kernel, and by default it looks like this:

```
linux16 /vmlinuz-0-rescue-5dea58df1a3b4cb5947ddb6c78a6773f
root=UUID=432d640e-3339-45fa-a66d-89da9c869550 ro rd.lvm.lv=centos/
swap vconsole.font=latarcyrheb-sun16 rd.lvm.lv=centos/root
crashkernel=auto  vconsole.keymap=us rhgb quiet
```

To start, it is a good idea to remove the **rhgb** and **quiet** parts from this line; these arguments hide boot messages for you, and typically you do want to see what is happening while booting. In the next section you learn about some troubleshooting options that you can enter from the GRUB boot prompt.

Accessing the Boot Prompt

After entering the boot options you want to use, press **Ctrl+X** to start the kernel with these options. Notice that these options are used one time only and are not persistent. To make them persistent you must modify the contents of the /etc/default/grub configuration file and use **grub2-mkconfig -o /boot/grub2/grub.cfg** to apply the modification.

Starting a Troubleshooting Target

When you are in trouble, you have a few options that you can enter on the GRUB boot prompt:

- **rd.break** This stops the boot procedure while still in the initramfs stage. This option is useful if you do not have the root password available. The complete procedure for recovering a missing root password follows later in this chapter.
- **init=/bin/sh** or **init=/bin/bash** This specifies that a shell should be started immediately after loading the kernel and initrd. This is a useful option, but not the best option, because in some cases you'll lose console access or miss other functionality.
- **systemd.unit=emergency.target** This enters in a bare minimal mode where a minimal number of systemd units is loaded. It requires a root password. To see that only a very limited number of unit files have been loaded, you can type the **systemctl list-units** command.
- **systemd.unit=rescue.target** This starts some more systemd units to bring you in a more complete operational mode. It does require a root password. To see that only a very limited number of unit files have been loaded, you can type the **systemctl list-units** command.

Exercise 19.1 Exploring troubleshooting targets.

1. (Re)start your computer. When the GRUB menu shows, select the first line in the menu and press **e**.
2. Scroll down to the line that starts with linux16 /vmlinuz. At the end of this line, type **systemd.unit=rescue.target**. Also remove the options **rhgb quit** from this line.
3. Enter the root password when you are prompted for it.
4. Type **systemctl list-units**. This shows all unit files that are currently loaded. You can see that a basic system environment has been loaded.
5. Type **systemctl show-environment**. This shows current shell environment variables.
6. Type **systemctl reboot** to reboot your machine.
7. When the GRUB menu shows, press **e** again to enter the editor mode. At the end of the line that loads the kernel, type **systemd.unit=emergency.target**.
8. When prompted for it, enter the root password to log in.
9. After successful login, type **systemctl list-units**. Notice that the number of unit files loaded is reduced to a bare minimum.

Using a Rescue Disk

If you are lucky when you are in trouble, you'll still be able to boot from hard disk. If you are a bit less lucky, you'll just see a blinking cursor on a system that does not boot at all. If that happens, you need a rescue disk. The default rescue image for Red Hat Enterprise Linux is on the installation disk. When booting from the installation disk, you'll see a Troubleshooting menu item. Select this item to get access to the options you need to repair your machine (see Figure 19.3).

Restoring System Access Using a Rescue Disk

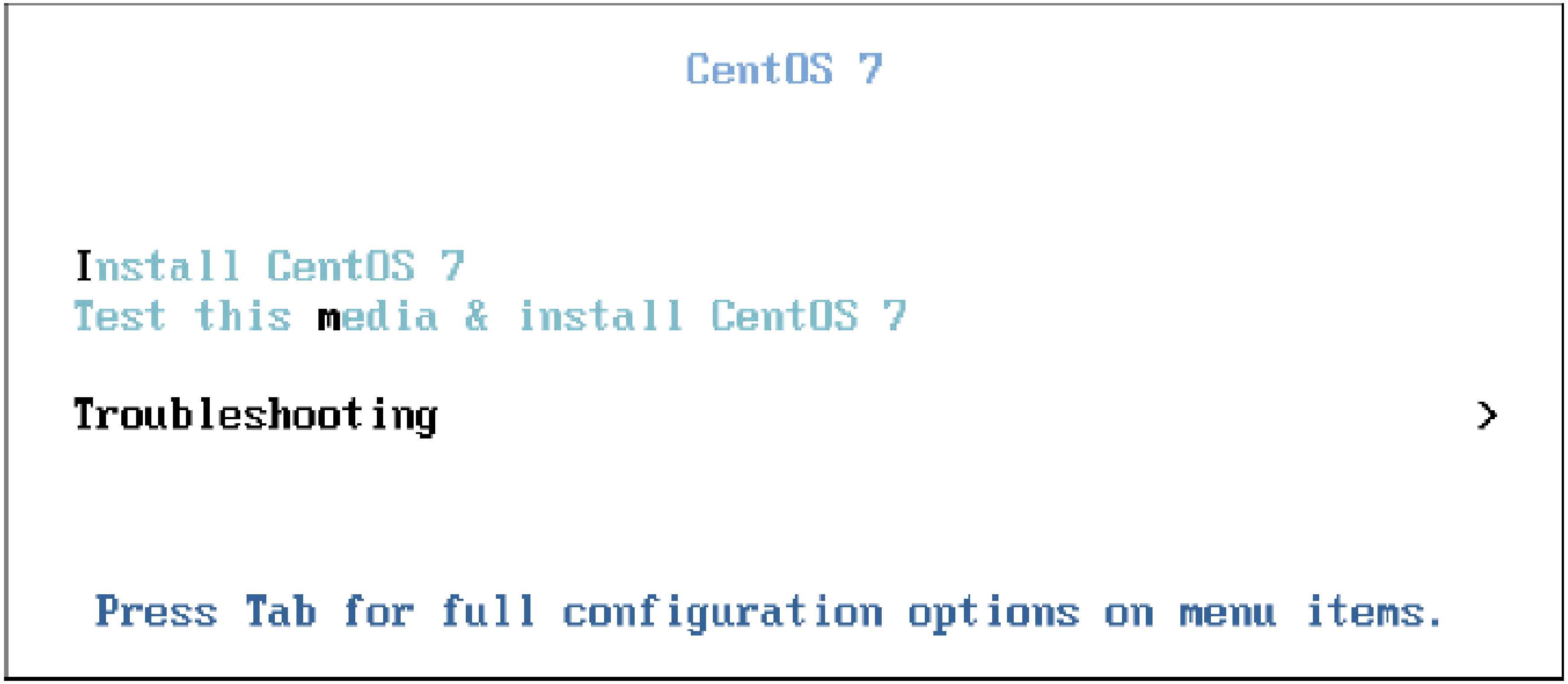

```
CentOS 7

Install CentOS 7
Test this media & install CentOS 7

Troubleshooting                                            >

Press Tab for full configuration options on menu items.
```

Figure 19.3 Starting from a rescue disk.

Restoring System Access Using a Rescue Disk

- **Install Red Hat 7 in Basic Graphics Mode:** This option reinstalls your machine. Do not use it unless you want to troubleshoot a situation where a normal installation does not work and you need a basic graphics mode.

  Normally, you should not ever need to use this option to troubleshoot a broken installation.
- **Rescue a Red Hat System:** This is the most flexible rescue system. In Exercise 19.2, you can explore it in detail. This should be the first option of choice when using a rescue disk.
- **Run a Memory Test:** Run this option if you encounter memory errors. It allows you to mark bad memory chips so that your machine can boot normally.
- **Boot from Local Drive:** If you cannot boot from GRUB on your hard disk, try this option first. It offers a boot loader that tries to install from your machine's hard drive, and as such is the least intrusive option available.

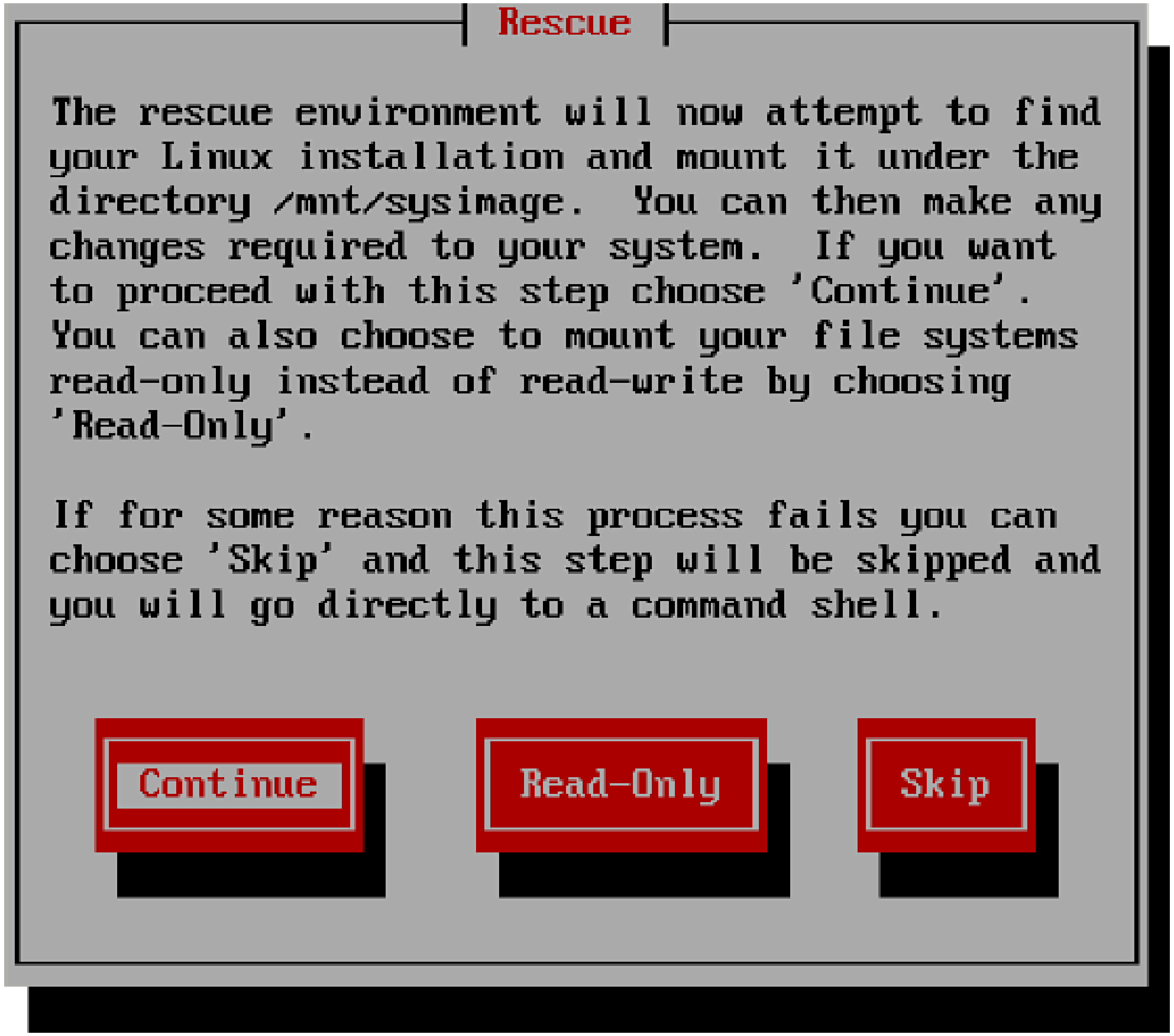
Restoring System Access Using a Rescue Disk

After starting a rescue system, you usually need to enable full access to the on-disk installation. Typically, the rescue disk detects your installation and mounts it on the /mnt/sysimage directory. To fix access to the configuration files and their default locations as they should be available on disk, use the **chroot /mnt/sysimage** command to make the contents of this directory your actual working environment. If you do not use this **chroot** command, many utilities will not work, because if they write to a configuration file that would be the version of the configuration file that exists on the rescue disk (and for that reason is read-only). Using the **chroot** command ensures that all path references to configuration files are correct.

Exercise 19.2 Using the Rescue Option

1. Restart your server from the installation disk. Select the **Troubleshooting** menu option.
2. From the Troubleshooting menu, select **Rescue a Red Hat System**. This prompts you to press **Enter** to start the installation. Do not worry: This option does not overwrite your current configuration; it just loads a rescue system.
3. The rescue system now prompts you that it will try to find an installed Linux system and mount on /mnt/sysimage. Press **Continue** to accept this option (see Figure 19.4).

Exercise 19.2 Using the Rescue Option



Rescue

The rescue environment will now attempt to find your Linux installation and mount it under the directory /mnt/sysimage. You can then make any changes required to your system. If you want to proceed with this step choose 'Continue'. You can also choose to mount your file systems read-only instead of read-write by choosing 'Read-Only'.

If for some reason this process fails you can choose 'Skip' and this step will be skipped and you will go directly to a command shell.

Exercise 19.2 Using the Rescue Option

4. If a valid Red Hat installation was found, you are prompted that your system has been mounted under /mnt/sysimage. At this point, you can press **Enter** twice to access the rescue shell.
5. Your Linux installation at this point is accessible through the /mnt/sysimage directory. Type **chroot /mnt/sysimage**. At this point, you have access to your root file system and you can access all tools that you need to repair access to your system.
6. Type **exit** and **reboot** to restart your machine in a normal mode.

Reinstalling GRUB Using a Rescue Disk

One of the common reasons you need to start a rescue disk is because the GRUB 2 boot loader is broken. If that happens, you might need to install it again. After you have restored access to your server using a rescue disk, reinstalling GRUB 2 is not hard to do and consists of two steps:

- Make sure that you have made the contents of the /mnt/sysimage directory to your current working environment.
- Use the **grub2-install** command, followed by the name of the device on which you want to reinstall GRUB 2. So on a KVM virtual machine, the command to use is **grub2-install /dev/vda**, and on a physical server or a VMware or Virtual Box virtual machine, it is **grub2-install /dev/sda**.

Re-Creating the Initramfs Using a Rescue Disk

Occasionally, the initramfs image may get damaged as well. If this happens, you cannot boot your server into normal operational mode. To repair the initramfs image after booting into the rescue environment, you can use the **dracut** command. If used with no arguments, this command creates a new initramfs for the kernel currently loaded.

Alternatively, you can use the **dracut** command with several options to make an initramfs for specific kernel environments. There is also a configuration file with the name /etc/dracut.conf that you can use to include specific options while re-creating the initramfs. The **dracut** configuration is dispersed over different locations:

- /usr/lib/dracut/dracut.conf.d/*.conf contains the system default configuration files.
- /etc/dracut.conf.d contains custom dracut configuration files.
- /etc/dracut.conf is used as the master configuration file.

Fixing Common Issues

In one small chapter such as this, it is not possible to consider all the possible problems one might encounter when working with Linux. There are some problems, though, that are more likely to occur than others. In this section you learn about some of the more common problems.

Reinstalling GRUB 2

Boot loader code does not disappear just like that, but on occasion it can happen that the GRUB 2 boot code gets damaged. In that case, you better know how to reinstall GRUB 2. The exact approach depends on whether your server is still in a bootable state. If it is, it is fairly easy to reinstall GRUB 2. Just type **grub2-install** followed by the name of the device to which you want to install it. The command has many different options to fine-tune what exactly will be installed, but you probably will not need them because, by default, the command installs everything you need to make your system bootable again.

It becomes a little bit more complicated if your machine is in a nonbootable state. If that happens, you first need to start a rescue system and restore access to your server from the rescue system. (See Exercise 19.2 for the exact procedure how to do that.) After mounting your server's file systems on /mnt/sysimage and using **chroot /mnt/sysimage** to make the mounted system image your root image, it is as easy as described previously: Just run **grub2-install** to install GRUB 2 to the desired installation device. So if you are in a KVM virtual machine, run **grub2-install /dev/vda**, and if you are on a physical disk, run **grub2-install /dev/sda**.

Fixing the Initramfs

In rare cases, it might happen that the initramfs gets damaged. If you analyze the boot procedure carefully, you will learn that you have a problem with the initramfs because you'll never see the root file system getting mounted on the root directory, nor will you see any systemd units getting started. If you suspect that you are having a problem with the initramfs, it is easy to re-create it. To re-create it using all default settings (which is fine in most cases), you can just run the **dracut --force** command. (Without **--force**, the command will refuse to overwrite your existing initramfs.)

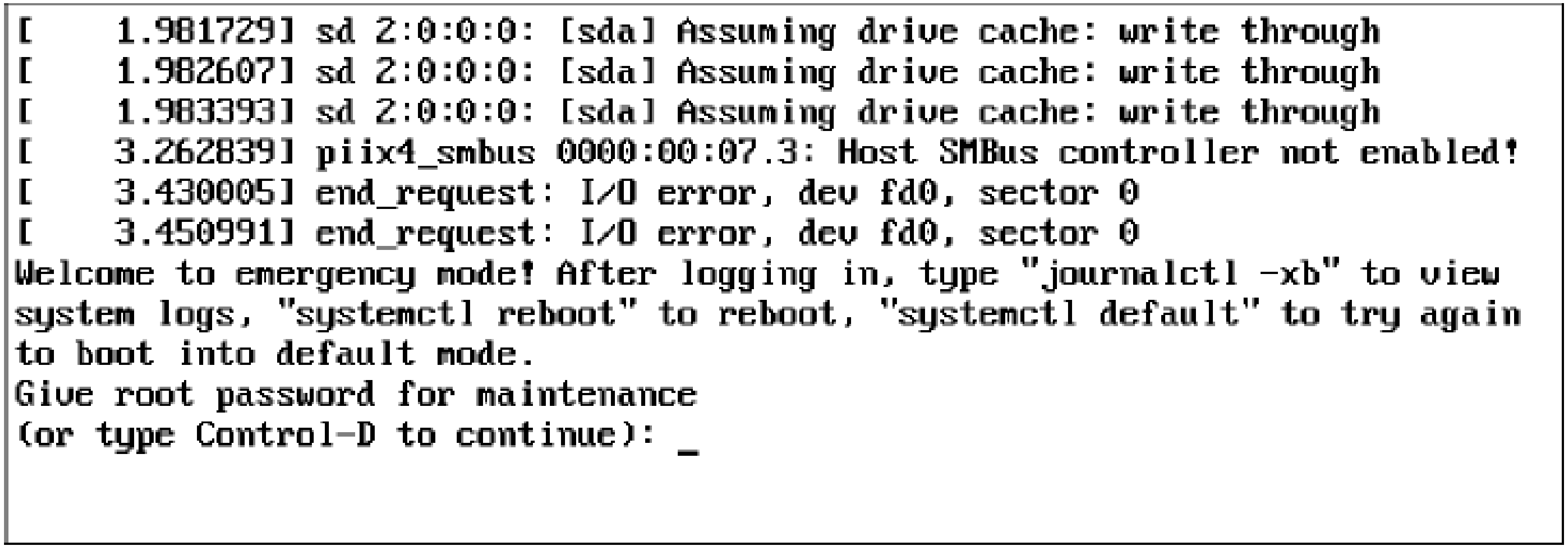
When running the **dracut** command, you can use the /etc/dracut.conf configuration file to specify what exactly is written to the initramfs. In this configuration file, you can see options like **lvmconf="no"** that can be used to switch specific features on or off. Use these options to make sure that you have all the required functionality in your initramfs.

Recovering from File System Issues

If you make a misconfiguration to your file system mounts, the boot procedure may just end with the message “Give root password for maintenance.” This message is, in particular, generated by the **fsck** command that is trying to verify the integrity of the file systems in /etc/fstab while booting. If **fsck** fails, manual intervention is required which may result in this message during boot. Make sure that you know what to do when this happens to you!

If a device is referred to that does not exist, or if there is an error in the UUID that is used to mount the device, for example, systemd waits first to see whether the device comes back by itself. If that does not happen, it gives the message “Give root password for maintenance” (see Figure 19.5). If that happens, you should by all means first enter the root password. Then you can type **journalctl -xb** as suggested to see whether relevant messages providing information about what is wrong are written to the journal. If the problem is file system oriented, type **mount -o remount,rw /** to make sure the root file system is mounted read-only and analyze what is wrong in the /etc/fstab file and fix it.

Recovering from File System Issues



```
[    1.981729] sd 2:0:0:0: [sda] Assuming drive cache: write through
[    1.982607] sd 2:0:0:0: [sda] Assuming drive cache: write through
[    1.983393] sd 2:0:0:0: [sda] Assuming drive cache: write through
[    3.262839] piix4_smbus 0000:00:07.3: Host SMBus controller not enabled!
[    3.430005] end_request: I/O error, dev fd0, sector 0
[    3.450991] end_request: I/O error, dev fd0, sector 0
Welcome to emergency mode! After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" to try again
to boot into default mode.
Give root password for maintenance
(or type Control-D to continue): _
```

Figure 19.5 If you see this, you normally have an /etc/fstab issue.

Resetting the Root Password

A common scenario for a Linux administrator is that the root password has gone missing. If that happens, you need to reset it. The only way to do that is by booting into minimal mode, which allows you to log in without entering a password. To do so, follow these steps:

Resetting the Root Password

1. On system boot, press **e** when the GRUB 2 boot menu is shown.
2. Enter **rd.break** as boot argument to the line that loads the kernel and press **Ctrl+X** to boot with this option.
3. You'll now be dropped at the end of the boot stage where initramfs is loaded, just before a mount of the root file system on the directory /.
4. Type **mount -o remount,rw /sysroot** to get read/write access to the system image.
5. At this point, make the contents of the /sysimage directory your new root directory by typing **chroot /sysroot.**
6. Now you can enter **passwd** and set the new password for the user root.
7. Because at this very early boot stage SELinux has not been activated yet, the context type on /etc/shadow will be messed up. If you reboot at this point, no one will be able to log in. So you must make sure that the context type is set correctly. To do this, at this point you should load the SELinux policy by using **load_policy -i**.

Resetting the Root Password

8. Now you can manually set the correct context type to /etc/shadow. To do this, type **chcon -t shadow_t /etc/shadow**.
9. Reboot. You can now log in with the changed password for user root.

NOTE In the preceding procedure you have read how to use the **load_policy -i** and **chcon** commands to correct the labels on the /etc/shadow file. An alternative (and easier) method is to create a file with the name /.autorelabel which will force SELinux to restore labels that are set on the entire file system.

Summary

In this chapter, you learned how to troubleshoot the Red Hat Enterprise Linux 7 boot procedure. You learned in general what happens when a server boots and at which specific points you can interfere to fix things that go wrong. You also learned what to do in some specific cases. Make sure that you know these procedures well; you are likely to encounter them on the exam.

Define Key Terms

- Define the following key terms:
  - target,
  - GRUB,
  - initramfs,
  - dracut

Lab 19.1

1. Restart your server and change the root password from the appropriate troubleshooting mode.
2. In /etc/fstab, change one of the device names so that on next reboot the file system on it cannot be mounted. Restart and fix the issue that you encounter.
3. Use a rescue disk to bring your server up in full troubleshooting mode from the rescue disk.
4. Re-create the initramfs.

Chapter 20:

Using Kickstart

Chapter 20 Objectives

- The following topics are covered in this chapter:
  - Setting Up an Installation Server
  - Setting Up a TFTP and DHCP Server for PXE Boot
  - Creating a Kickstart File

- The following RHCSA exam objectives are covered in this chapter:
  - Installing Red Hat Enterprise Linux automatically using Kickstart

Using Kickstart

In this chapter, you learn how to install RHEL 7 automatically using Kickstart. Kickstart by itself is simple to understand and configure: It is just a configuration file that needs to be used by the installer to specify how exactly the installation is to be performed (and that is also what the RHCSA objective seems to refer to).

Using Kickstart in an environment where you are still installing from an optical disk is not that useful, though, which is why in this chapter you learn how to set up a complete environment where an installation server is providing access to the repository that is used, and a PXE boot server is configured to provide access to a boot image that can be used to start a fully automated installation. This may go way beyond the RHCSA objectives, but at least it provides you with useful information that you can use to set up a fully automated installation environment.

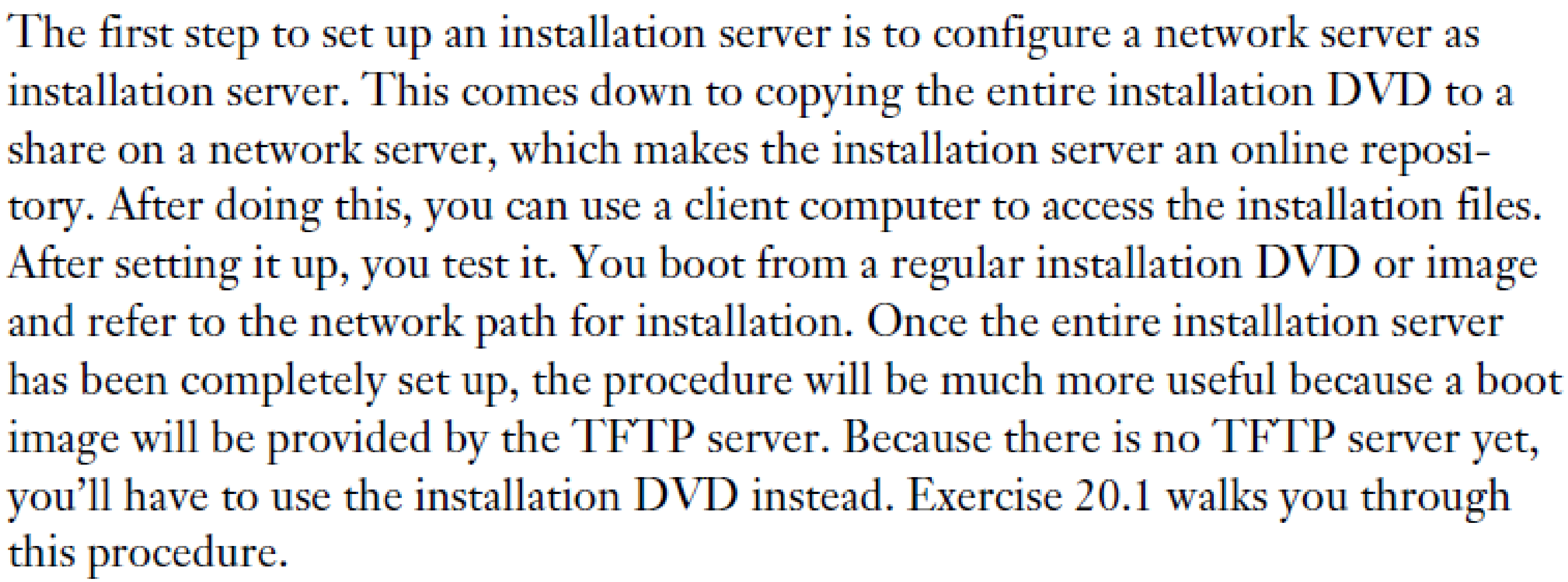
Setting Up an Installation Server

In this chapter, you learn how to set up an installation server. This is useful if you need to install many instances of Red Hat Enterprise Linux. Using an installation server means that you can avoid installing every physical server by inserting a DVD in that server. You'll install new servers over the network instead. Also, it allows you to install servers that do not have an optical drive, such as blades.

Setting up an installation server involves different steps.

- To start, you need to make the installation files available. To do this, you configure a network server. This can be an NFS, FTP, or HTTP server.
- Then you need to deliver a boot image to the clients that need to be installed. For a fully automated network installation, you need to set up PXE boot that provides a boot image to your client by working together with the DHCP server.
- As the last step to set up an installation server, you create a Kickstart file. This is an answer file that contains all settings that are needed to install your servers and the final step in setting up a completely automated installation. The Kickstart file can be provided through an installation server to provide a fully automated installation, but it can also be offered on a USB device.

Configuring a Network Server as Installation Server

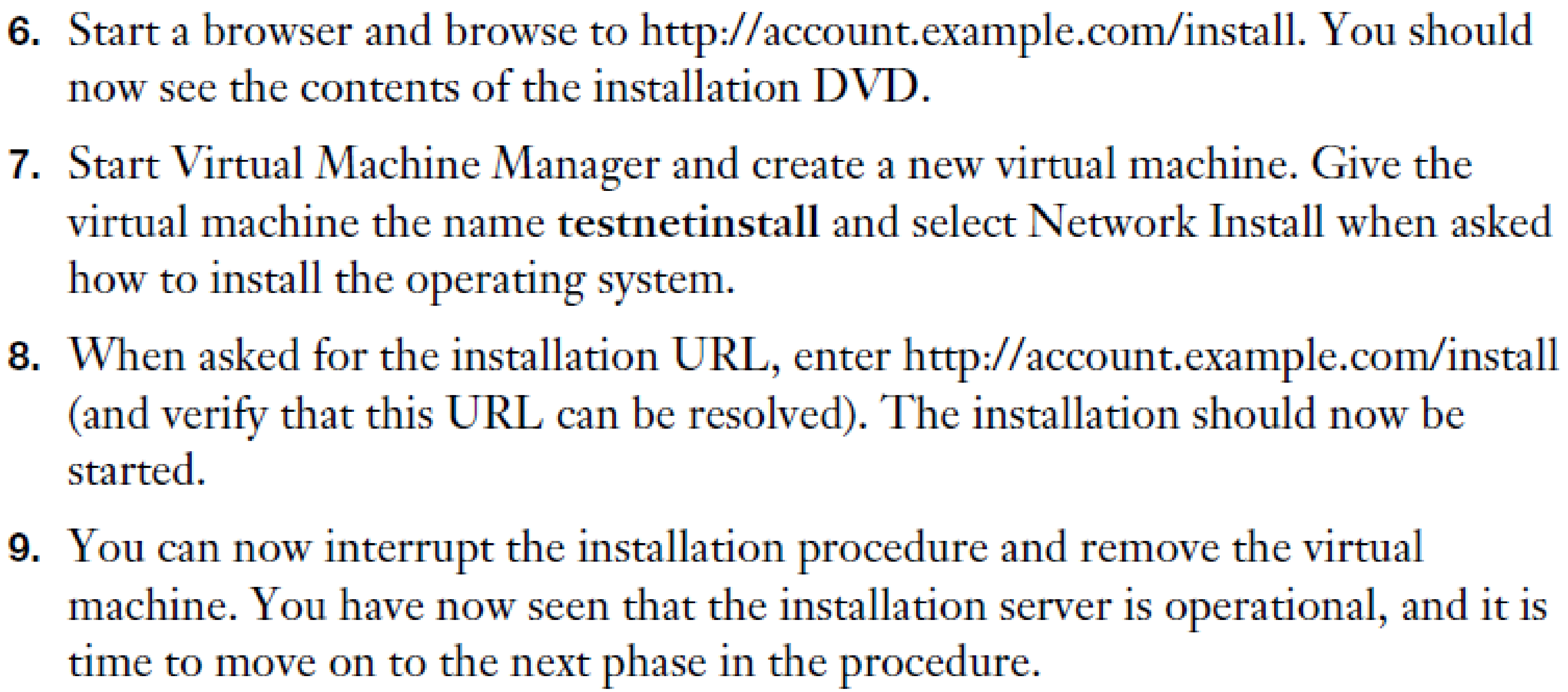


The first step to set up an installation server is to configure a network server as installation server. This comes down to copying the entire installation DVD to a share on a network server, which makes the installation server an online repository. After doing this, you can use a client computer to access the installation files. After setting it up, you test it. You boot from a regular installation DVD or image and refer to the network path for installation. Once the entire installation server has been completely set up, the procedure will be much more useful because a boot image will be provided by the TFTP server. Because there is no TFTP server yet, you'll have to use the installation DVD instead. Exercise 20.1 walks you through this procedure.

Exercise 20.1 Setting Up the Network Installation Server

1. Insert the Red Hat Enterprise Linux installation DVD in the optical drive of your server and navigate to the Packages directory on the installation disk.
2. Use **mkdir /www/docs/account.example.com/install** to create a subdirectory in the Apache document root for account.example.com.
3. Use **cp -R * /www/docs/account.example.com/install** from the directory where the Red Hat Enterprise Linux installation DVD is mounted to copy all files on the DVD to the install directory in your web server document root.
4. Modify the configuration file for the server1 virtual host in /etc/httpd/conf.d/account.example.com and make sure that it includes the line Options Indexes. Without this line, the virtual host will only show contents of a directory if it contains an index.html file.
5. Use service httpd restart to restart the Apache web server.

Exercise 20.1 Setting Up the Network Installation Server



6. Start a browser and browse to http://account.example.com/install. You should now see the contents of the installation DVD.
7. Start Virtual Machine Manager and create a new virtual machine. Give the virtual machine the name **testnetinstall** and select Network Install when asked how to install the operating system.
8. When asked for the installation URL, enter http://account.example.com/install (and verify that this URL can be resolved). The installation should now be started.
9. You can now interrupt the installation procedure and remove the virtual machine. You have now seen that the installation server is operational, and it is time to move on to the next phase in the procedure.

Setting Up a TFTP and DHCP Server for PXE Boot

Now that you have set up a network installation server, it is time to configure PXE boot. This allows you to boot a server you want to install from the network card of the server. (You normally have to change default boot order, or press a key while booting, to activate PXE boot.) The PXE server next hands out a boot image that the server you want to install uses to start the initial phase of the boot.

There are two steps involved: You need to install a TFTP server and have it provide a boot image to PXE clients, and you need to configure DHCP to talk to the TFTP server to provide the boot image to PXE clients.

Setting Up a TFTP and DHCP Server for PXE Boot

The first part of the installation is easy: You need to install the tftp-server package using **yum -y install tftp-server**. The tftpd service is managed by the xinetd service, and to tell xinetd that it should allow access to TFTP, you need to open the /etc/xinetd.d/tftp file (see Listing 20.1) and change the disabled parameter from yes to no.

NOTE The xinetd service is also know as the Internet super service. It comes from a time where memory resources were limited, and to use memory as efficient as possible, xinetd could be configured to listen on many ports, making it possible to access many different services. Using xinetd can still be helpful, particularly for services that are not accessed very often. The xinetd service will make sure that the service is started when some processes access its port, and will also shut it down after a specific period of inactivity.

Setting Up a TFTP and DHCP Server for PXE Boot

After enabling the tftp service, complete the following steps to make it available:

1. Restart the xinetd service using **systemctl start xinetd**.
2. Make sure to include xinetd in your startup procedure, using **systemctl enable xinetd**.
3. Make sure to permit tftp in your firewall configuration, using **firewall-cmd --permanent --add-service=tftp**.
4. Reload the firewall configuration to make the firewall modification effective, using **firewall-cmd --reload**.

Setting Up a TFTP and DHCP Server for PXE Boot

Listing 20.1 The xinetd File for TFTP

```
[root@server1 ~]# cat /etc/xinetd.d/tftp
# default: off
# description: The tftp server serves files using the trivial file
# transfer \
#         protocol.  The tftp protocol is often used to boot diskless \
#         workstations, download configuration files to network-aware
# printers, \
#         and to start the installation process for some operating
# systems.
service tftp
{
          socket_type                 = dgram
          protocol                    = udp
          wait                        = yes
          user                        = root
          server                      = /usr/sbin/in.tftpd
          server_args                 = -s /var/lib/tftpboot
          disable                     = no
          per_source                  = 11
          cps                         = 100 2
          flags                       = IPv4
```

Setting Up a TFTP and DHCP Server for PXE Boot

At this point the Trivial File Transfer Protocol (TFTP) server is operational. You now have to configure Dynamic Host Configuration Protocol (DHCP) to communicate to the TFTP server to hand out a boot image.

Configuring DHCP for PXE Boot

On an installation server, the TFTP server cannot exist without a DHCP server. When making a PXE boot, the DHCP server is the first to answer with all the required IP-related configuration and information about the DHCP server that is to be used. Therefore, you now have to install a DHCP server. Use **yum install -y dhcp** to install the server. You then have to create a subnet and modify the DHCP server configuration so that it can hand out a boot image to PXE clients. To do this, make sure to include the following (see Listing 20.2) in your dhcpd.conf and restart the DHCP server.

Listing 20.2 Adding PXE Boot Lines to the dhcpd.conf

```
subnet 192.168.1.0 netmask 255.255.255.0 {
        option routers 192.168.1.1 ;
        range 192.168.1.200 192.168.1.250 ;
        next-server 192.168.1.70;
        filename "pxelinux/pxelinux.0";
}
```

Configuring DHCP for PXE Boot

In the DHCP configuration file, a subnet is specified. This is the subnet where the PXE server should offer its services. In most configurations, this would be a dedicated network, which is a good idea, because you do not want workstations that perform a PXE boot to get installed with Red Hat Enterprise Linux by accident. If you want to offer PXE-based installation services on a network where also clients are in use that rely on PXE boot, it is recommended to define a **class** statement to define which machines should be allowed to use PXE boot and which should not.

TIP On the exam, it will not be a problem making no further specification to define which machines should be using the PXE server and which should not. You will not have anything that relies on a PXE boot to get started anyway.

Configuring DHCP for PXE Boot

Within the subnet definition, the **next-server** statement gives the IP address of the server that is configured with TFTP. Even if it is on the same server that is offering DHCP, you should still specify the next-server IP address. The **filename** statement defines the file that should be offered to workstations that are making a PXE boot. Notice that this filename is relative to the TFTP server root as defined in the TFTP configuration file.

Creating the TFTP PXE Server Content

You have now prepared the DHCP server with the appropriate settings, but so far it has nothing to offer yet. This section describes how to create the TFTP PXE server content that is going to be offered through DHCP.

The role of the PXE server is to deliver an image to the client that performs a PXE boot. In fact, it replaces the task that is normally performed by GRUB 2 and the contents of the boot directory and provides a bootloader over the network. Therefore, to configure a PXE server, you need to copy everything that is needed to boot your server to the /var/lib/tftpboot/pxelinux directory. You also need to create a PXE boot file that performs the task that is normally handled by the grub.conf file. In Exercise 20.2, you copy all required content to the TFTP server root directory.

Exercise 20.2 Configuring the TFTP Server for PXE Boot

To set up a TFTP server, you configure a DHCP server and the TFTP server. Notice that the configuration of a DHCP Server on your network can cause problems. An additional complicating factor is that the KVM virtual network environment probably already runs a DHCP server. Therefore, you cannot use the DHCP server that you'll configure to serve virtual machines. To make this exercise a success, make sure your Red Hat Enterprise Linux server is disconnected from the network and connect it to one PC only that is capable of performing a PXE boot. Every modern PC should be capable of making a PXE boot. You do not have to reinstall the computer you are using for this purpose. You just have to test that it can make a PXE boot.

Exercise 20.2 Configuring the TFTP Server for PXE Boot

1. Use yum **install -y tftpserver** to install the TFTP server. Because TFTP is managed by xinetd, use **systemctl enable xinetd** to have xinetd started automatically.
2. Open the configuration file /etc/xinetd.d/tftp with an editor and change the line **disabled = yes** to **disabled = no**.
3. If not installed yet, install a DHCP server. Open the configuration file /etc/dhcp/dhcpd.conf and give it the contents of Listing 20.2. Make sure the IP address ranges you are using match the IP configuration that is used on your network.
4. Copy the syslinux<version>.rpm from the Packages directory on the RHEL installation disc to /tmp. You'll need to extract the file pxelinux.0 from it, which is an essential file for setting up the PXE boot environment. To extract the RPM file, use **cd /tmp** to go to the /tmp directory, and from there, use **rpm-2cpio syslinux<version>.rpm | cpio -idmv** to extract the file.

Exercise 20.2 Configuring the TFTP Server for PXE Boot

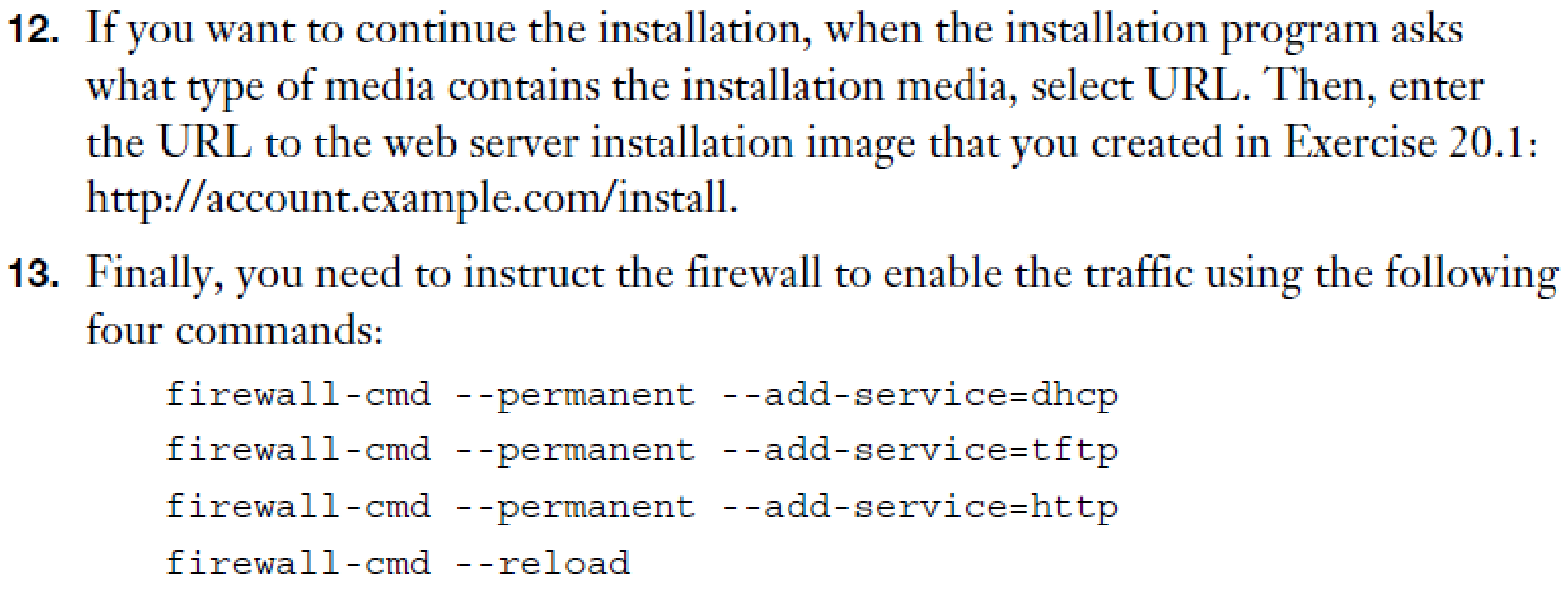
5. Copy the /usr/share/syslinx/pxelinux.0 file to /var/lib/tftpboot/pxelinux.
6. Use **mkdir /var/lib/tftpboot/pxelinux/pxelinux.cfg** to create the directory in which you'll store the pxelinux configuration file.
7. In /var/lib/tftpboot/pxelinux/pxelinux.cfg, create a file with the name default that contains the following lines:

```
default Linux
prompt 1
timeout 10
display boot.msg
label Linux
        menu label ^Install RHEL
        menu default
        kernel vmlinuz
        append initrd=initrd.img
```

Exercise 20.2 Configuring the TFTP Server for PXE Boot

8. If you want to use a splash image file while doing PXE boot, copy the /boot/grub/splash.xpm.gz file to /var/lib/tftptboot/pxelinux/.
9. On the Red Hat installation disc, in the directory images/pxeboot, you can find the files vmlinuz and initrd.img. Copy these to the directory /var/lib/tftpboot/pxelinux/.
10. Use systemctl restart dhcpd and systemctl restart xinetd to (re)start the required services.
11. On the server, use **tail -f /var/log/messages** to trace what is happening. Connect a computer directly to the server, and from that computer, choose PXE boot in the boot menu. You will see that the computer starts the PXE boot and loads the installation image that you have prepared for it.

Exercise 20.2 Configuring the TFTP Server for PXE Boot



12. If you want to continue the installation, when the installation program asks what type of media contains the installation media, select URL. Then, enter the URL to the web server installation image that you created in Exercise 20.1: http://account.example.com/install.

13. Finally, you need to instruct the firewall to enable the traffic using the following four commands:

```
firewall-cmd --permanent --add-service=dhcp
firewall-cmd --permanent --add-service=tftp
firewall-cmd --permanent --add-service=http
firewall-cmd --reload
```

Exercise 20.2 Configuring the TFTP Server for PXE Boot

If you are adding more options to the PXE menu, it also makes sense to increase the timeout to allow users to make a choice. In Listing 20.3, this is done by using the timeout 600 value. Notice, however, that this is not typically what you want if you want to use the PXE server for automated installations using a Kickstart file, as described in the following section.

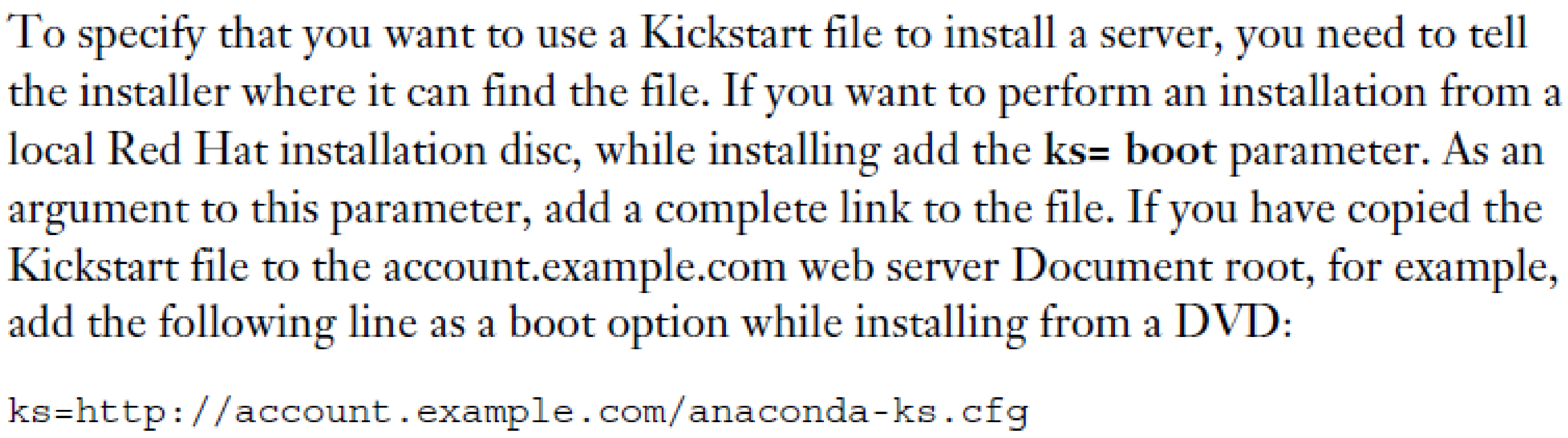
Listing 20.3 Adding More Options to the PXE Boot Menu

```
default Linux
prompt 1
timeout 600
display boot.msg
label Linux
          menu label ^Install RHEL
          menu default
          kernel vmlinuz
          append initrd=initrd.img
label Rescue
          menu label ^Rescue system
          kernel vmlinuz
          append initrd=initrd.img rescue
```

Creating a Kickstart File

You have now created an environment where everything you need to install your server is available on another server. Therefore, you do not have to work with optical disks anymore to perform an installation. Red Hat offers an excellent solution for this challenge: the Kickstart file. In this section, you learn how to use a Kickstart file to perform a completely automated installation and how you can optimize the Kickstart file to fit your needs.

Using a Kickstart File to Perform an Automated Installation



To specify that you want to use a Kickstart file to install a server, you need to tell the installer where it can find the file. If you want to perform an installation from a local Red Hat installation disc, while installing add the **ks= boot** parameter. As an argument to this parameter, add a complete link to the file. If you have copied the Kickstart file to the account.example.com web server Document root, for example, add the following line as a boot option while installing from a DVD:

```
ks=http://account.example.com/anaconda-ks.cfg
```

Using a Kickstart File to Perform an Automated Installation

To use a Kickstart file in an automated installation from a TFTP server, you need to add the Kickstart file to the section in the TFTP default file that starts the installation. The section that you need to install the server would in that case look like the following:

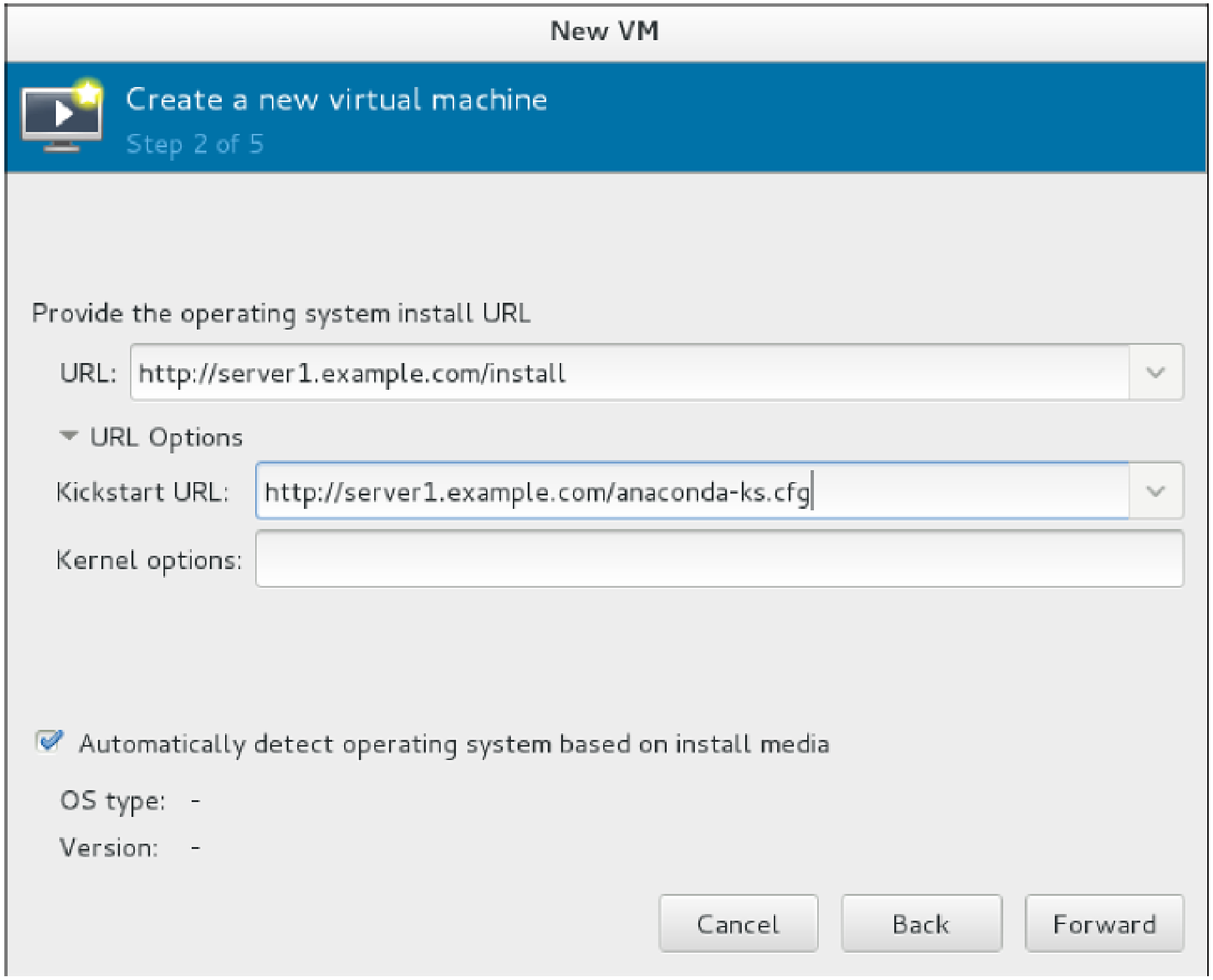
```
label Linux
          menu label ^Install RHEL
          menu default
          kernel vmlinuz
          append initrd=initrd.img \
          ks=http://account.example.com/anaconda-ks.cfg
```

In this file, make sure that the title of the menu label starts with a ^, which identifies it as a menu label title. Also make sure that the **append** line is one line only that starts with append and ends with the URL to the Kickstart file.

Exercise 20.3 Performing a Virtual Machine Network Installation Using a Kickstart File

1. On the installation server, copy the anaconda-ks.cfg file from the /root directory to /www/docs/account.example.com directory. You can just copy it straight to the root directory. After copying the file, set the permissions to mode 644; otherwise, the Apache user cannot read it.
2. Start Virtual Machine Manager and click the Create Virtual Machine button. Enter a name for the virtual machine, and select Network Install.
3. In the second screen of the Create a New Virtual Machine Wizard, enter the URL to the web server installation directory: http://account.example.com/install. Open the URL Options and enter the following Kickstart URL: http://account.example.com/anaconda-ks.cfg (see Figure 20.1).
4. Accept all default options in the remaining windows of the Create a New Virtual Machine Wizard, which will start the installation. In the beginning of the procedure, you'll see the message "Retrieving anaconda-ks.cfg." If that message disappears and you do not see an error message, the Kickstart file has loaded correctly.
5. Stop the installation after the Kickstart file has loaded. The Kickstart file was not made for virtual machines, so it will ask lots of questions anyway. (The only purpose of this exercise was to show that including a Kickstart file actually works.) After stopping the installation, remove the Kickstart file from the Virtual Machine Manager configuration.

Exercise 20.3 Performing a Virtual Machine Network Installation Using a Kickstart File



Provide the operating system install URL

URL: http://server1.example.com/install

URL Options

Kickstart URL: http://server1.example.com/anaconda-ks.cfg

Kernel options:

✓ Automatically detect operating system based on install media

OS type: -

Version: -

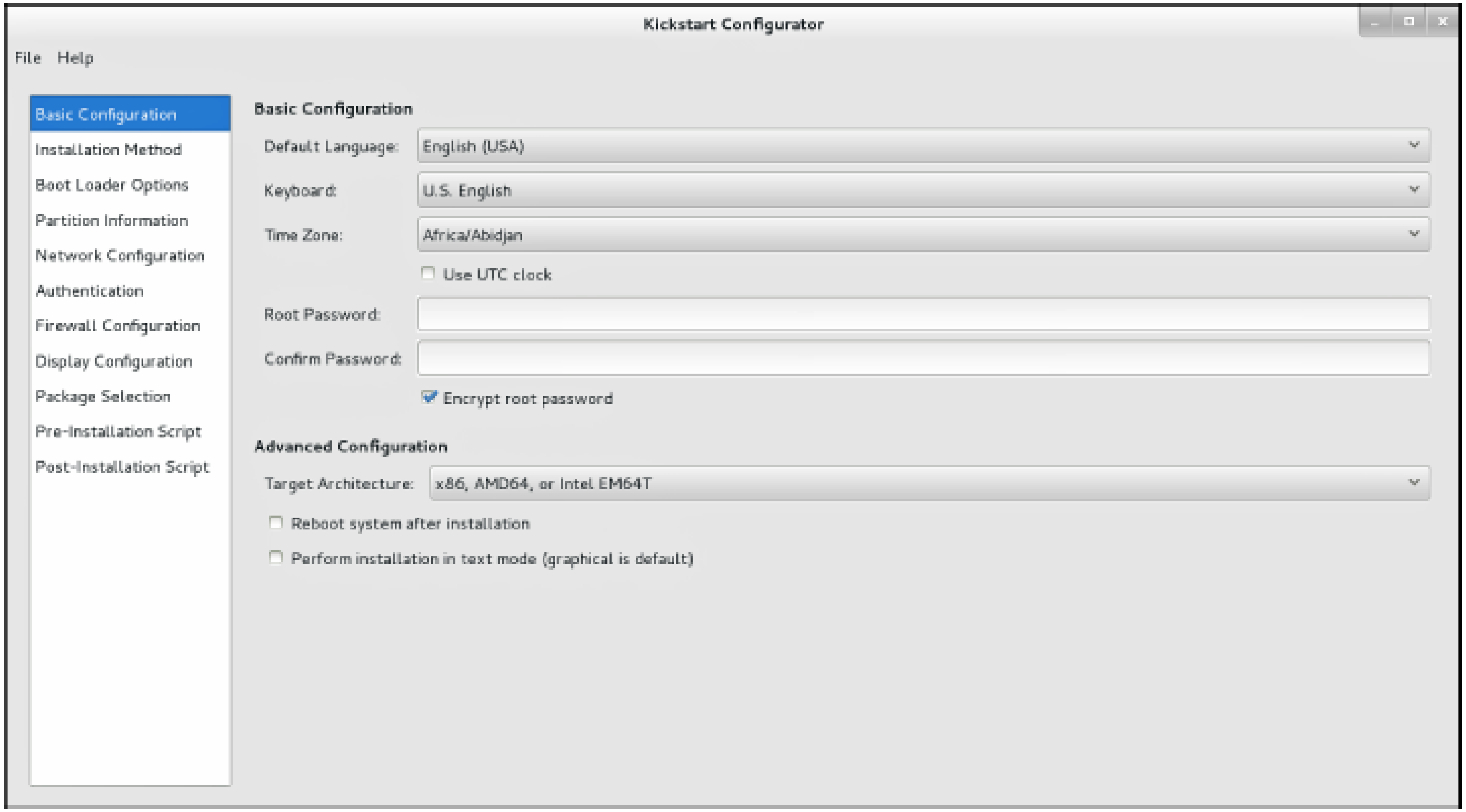
Figure 20.1 Specifying required options to use a Kickstart file.

Modifying the Kickstart File with system-config-kickstart

In the previous exercise, you started a Kickstart installation based on the Kickstart file that was created after the installation of your server was finished. You might have noticed that still many questions were asked anyway. That is because your Kickstart file did not fit the hardware of the virtual machine you were trying to install. In many cases, you need to fine-tune the Kickstart configuration file. To do this, you can use the system-config-kickstart graphical interface (see Figure 20.2).

Using system-config-kickstart, you can create new Kickstart files, but you can also read an existing Kickstart file and make all modifications you need without the risk of making syntax errors that could cause your Kickstart file to fail.

Modifying the Kickstart File with system-config-kickstart



Basic Configuration

Default Language: English (USA)

Keyboard: U.S. English

Time Zone: Africa/Abidjan

☐ Use UTC clock

Root Password:

Confirm Password:

☑ Encrypt root password

Advanced Configuration

Target Architecture: x86, AMD64, or Intel EM64T

☐ Reboot system after installation

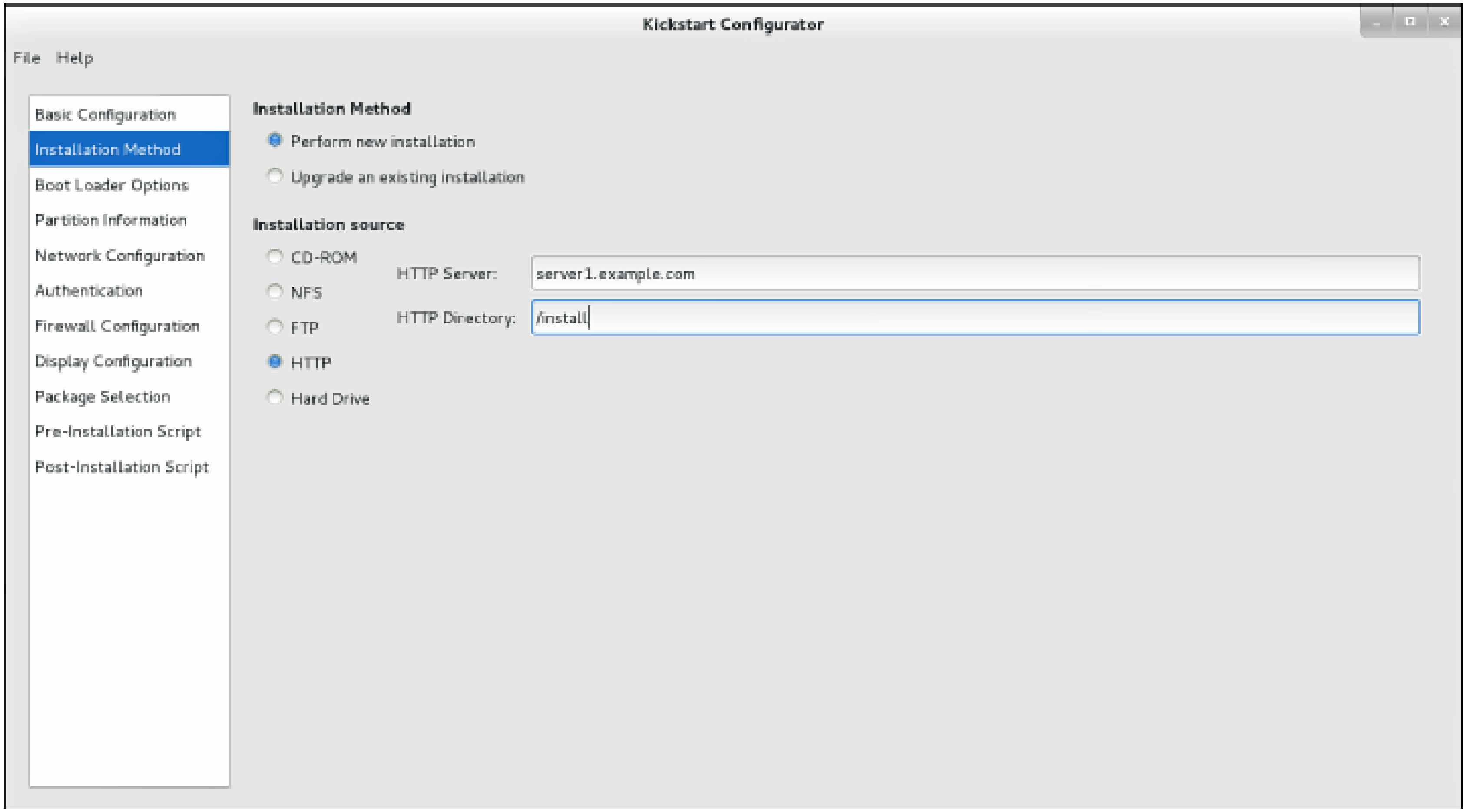
☐ Perform installation in text mode (graphical is default)

Figure 20.2 Use system-config-kickstart to create or tune Kickstart files.

Modifying the Kickstart File with system-config-kickstart

Under the Basic Configuration option, you can find basic options like the keyboard and time zone that your server will be installed in. Here, you'll also find an interface to set the root password. Under Installation Method, you'll find (among other things) the installation source. For a network installation, you need to select the type of network installation server and the directory used on that server. In Figure 20.3, you can see what this would look like for the installation server you created in Exercise 20.1.

Modifying the Kickstart File with system-config-kickstart



Installation Method

● Perform new installation
○ Upgrade an existing installation

Installation source

○ CD-ROM
○ NFS
○ FTP
● HTTP
○ Hard Drive

HTTP Server: server1.example.com

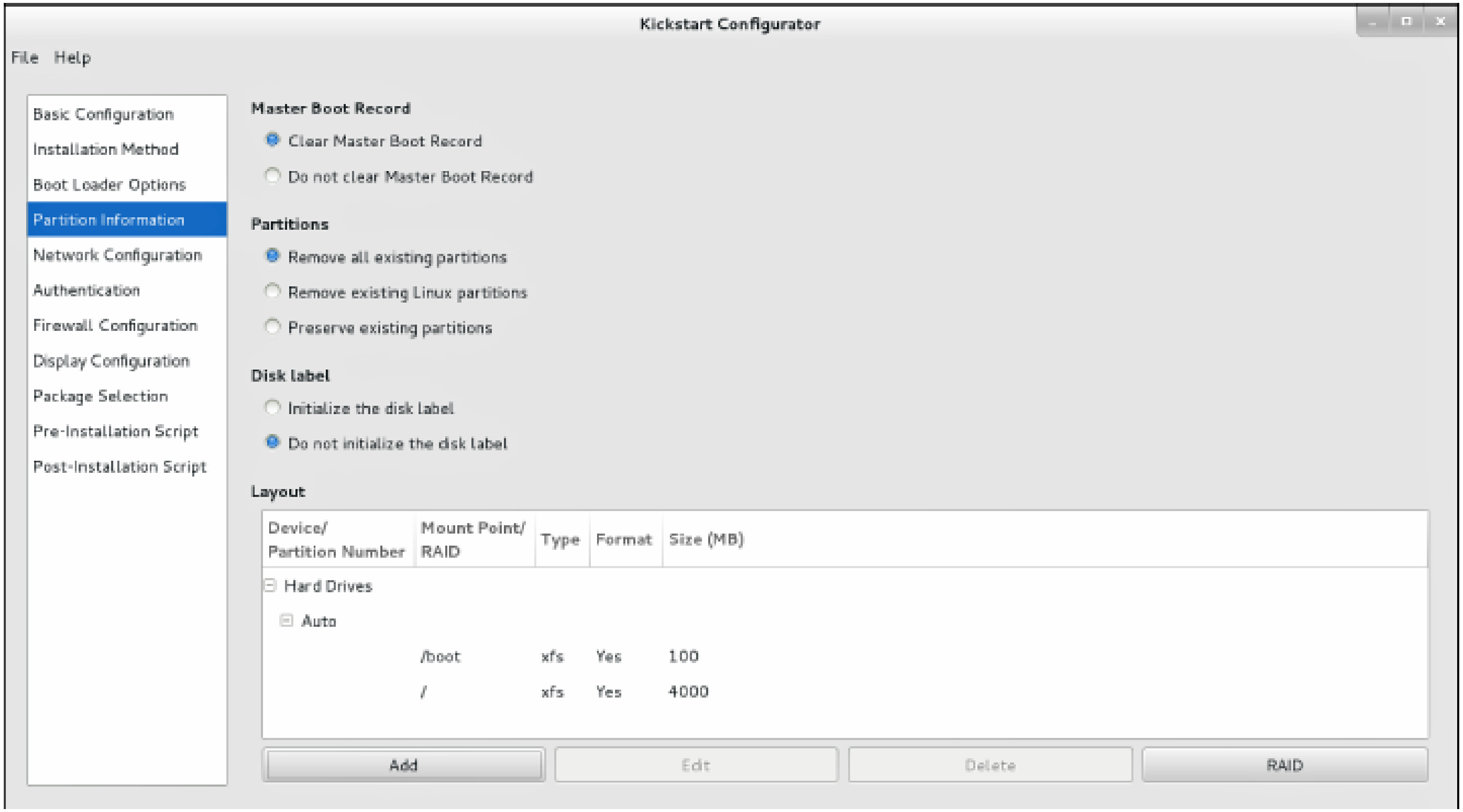
HTTP Directory: /install

Figure 20.3 Specifying the network installation source.

Modifying the Kickstart File with system-config-kickstart

Under Boot Loader Options, you can specify that you want to install a new boot-loader and where you want to install it. If specific kernel parameters are needed while booting, you can specify them here as well. An important option is Partition Information (see Figure 20.4). Here, you can tell Kickstart which partitions you want to create on the server. Unfortunately, the interface does not allow you to create logical volumes. So if you need these, you need to add them manually, which is explained in the following section.

Modifying the Kickstart File with system-config-kickstart



Master Boot Record

[✓ Clear Master Boot Record]
Do not clear Master Boot Record

Partitions

[✓ Remove all existing partitions]
Remove existing Linux partitions
Preserve existing partitions

Disk label

Initialize the disk label
[✓ Do not initialize the disk label]

Layout

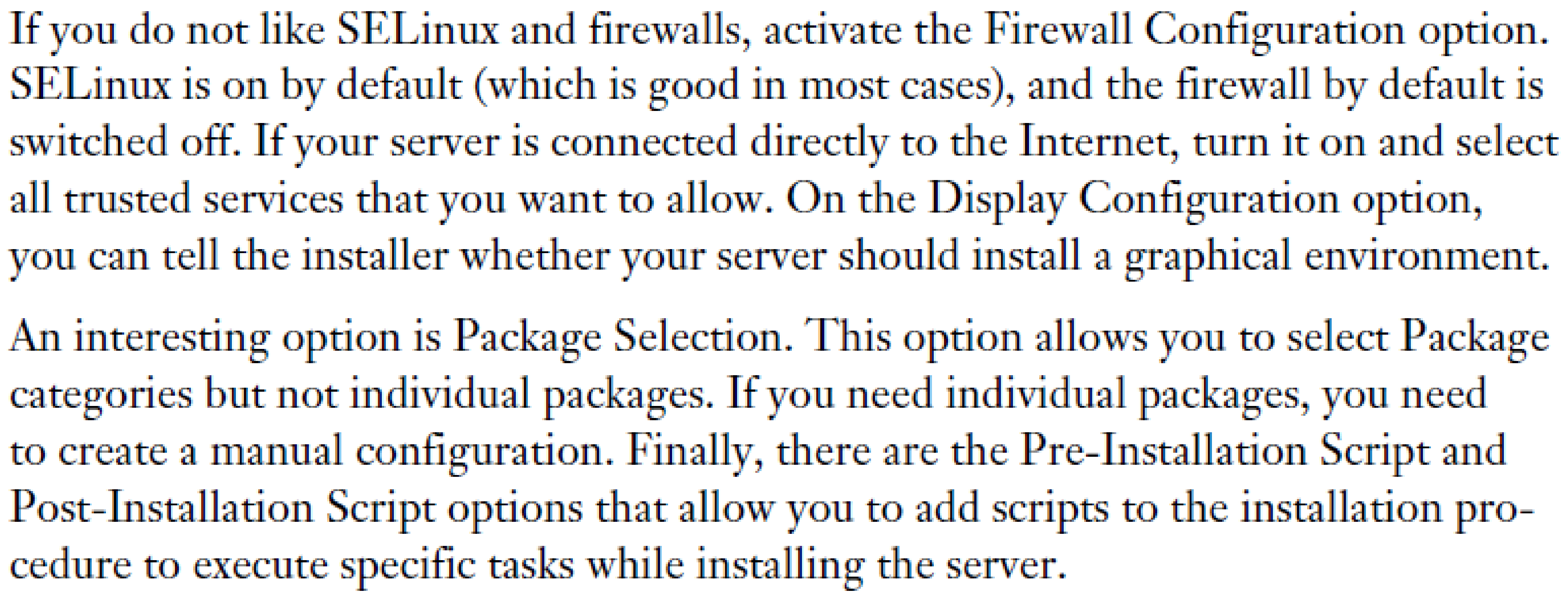
Device/ Partition Number	Mount Point/ RAID	Type	Format	Size (MB)
Hard Drives				
Auto				
	/boot	xfs	Yes	100
	/	xfs	Yes	4000

Figure 20.4 Creating partitions.

Modifying the Kickstart File with system-config-kickstart

The Network Configuration option by default is empty. If you want networking on your server, you need to use the Add Network Device option to indicate the name of the device and how you want the device to get its network configuration. The Authentication option offers tabs to specify external authentication services, such as NIS, LDAP, Kerberos, and some others. If you do not specify any of these, you just use the local authentication mechanism that goes through /etc/passwd, which is fine for many servers.

Modifying the Kickstart File with system-config-kickstart



If you do not like SELinux and firewalls, activate the Firewall Configuration option. SELinux is on by default (which is good in most cases), and the firewall by default is switched off. If your server is connected directly to the Internet, turn it on and select all trusted services that you want to allow. On the Display Configuration option, you can tell the installer whether your server should install a graphical environment.

An interesting option is Package Selection. This option allows you to select Package categories but not individual packages. If you need individual packages, you need to create a manual configuration. Finally, there are the Pre-Installation Script and Post-Installation Script options that allow you to add scripts to the installation procedure to execute specific tasks while installing the server.

Making Manual Modifications to the Kickstart File

There are some modifications that you cannot make to a Kickstart file using the graphical interface. Fortunately, a Kickstart file is an ASCII text file that can easily be edited by hand. By making manual modifications, you can configure features like LVM logical volumes, or individual packages, tasks that cannot be accomplished from the system-config-kickstart interface. In Listing 20.4, you can see the contents of the anaconda-ks.cfg file that is generated upon installation of a server. Studying this file is interesting because it shows examples of everything that cannot be done from the graphical interface.

Making Manual Modifications to the Kickstart File

Listing 20.4 Contents of the anaconda-ks.cfg File

```
[root@server1 ~]# cat anaconda-ks.cfg
#version=RHEL7
# System authorization information
auth --enableshadow --passalgo=sha512

# Use CDROM installation media
cdrom
# Run the Setup Agent on first boot
firstboot --enable
```

Making Manual Modifications to the Kickstart File

```
ignoredisk --only-use=sda
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8

# Network information
network  --bootproto=dhcp --device=eno16777736 --ipv6=auto --activate
network  --hostname=localhost.localdomain
# Root password
rootpw --iscrypted $6$w.otIj7Fjqpr4Gmy$Uwo6TCHDf8TzI98nyy3zagSi6HVrs4Ur
8JI7J90.q5PtYi236k0Zo4rQp4kDyOY8zXS1dMbJadMNvSQ9Ul9SS1
# System timezone
timezone America/New_York --isUtc
user --name=user
password=$6$CwZOJIdBeiVJmte0$cYFoDfuucuIzSOeS3FIQvhxl1T02mrfeo
BCAvybP9719/HN9qY7ZtWVJ5iKXdKTBgbFz8o3gBCAIKI4j4HrQD. --iscrypted
--gecos="user"
# X Window System configuration information
xconfig  --startxonboot
```

Making Manual Modifications to the Kickstart File

The anaconda-ks.cfg file starts with some generic settings. The first line that needs attention is the network line. As you can see, it contains the device name `--device eno16777736`. This device name is related to the specific hardware configuration of the server the file was created on, and it will probably not work on many hardware platforms. So, it is better to replace it with **--device eth0**.

The next interesting parameter is the line that contains the root password. As you can see, it contains the encrypted root password that was used while installing this server. If you want the installation process to prompt for a root password, you can just remove this line completely; everything that the installer cannot get directly from the Kickstart file will be prompted for.

Making Manual Modifications to the Kickstart File

In this example Kickstart file, you can see that the disk is partitioned automatically, using the **autopart --type=lvm** option. Then, the repository that is to be used is specified. This is also a parameter for which it is likely that it needs to be changed. The **--baseurl** parameter contains a URL that refers to the installation URL that you want to use. It can, for instance, read **--baseurl=http://account.example.com/install** to refer to an http installation server.

In the next section, the packages that are to be installed are specified. Everything that starts with a @ (such as @base) refers to an RPM package group. Individual packages can be specifically added by just mentioning the name of the packages.

TIP If you do not get any errors, but the server seems to stall or will not provide the installation files, you can switch over to other virtual terminals on the server that you are installing to check whether it shows any error messages. To do so, you can use the **Ctrl+Alt-F2** up to **Ctrl+Alt-F7** key sequences. To get back to the graphical installation screen, use **Ctrl+Alt-F1**.

Summary

In this chapter, you learned how to set up an installation server. You learned how a typical installation server consists of a few different parts. First you need to have the installation files available online, which in this chapter was done by using a web server. Then, you need a mechanism to deliver a boot image. You learned how to configure DHCP and TFTP for this purpose. Lastly, for fully automated installations, you need an answer file. This answer file is provided by Kickstart. Putting all of these together allows you to work with a fully functional installation server.

Define Key Terms

- Define the following key terms:
  - xinetd,
  - tftp,
  - installation server,
  - Kickstart,
  - anaconda

Lab 20.1

1. Set up an HTTP installation server that provides access to the installation files over the network.
2. Configure DHCP and TFTP so that the boot image can be provided to the clients that need to be installed.
3. Create a Kickstart file that provides all settings for a basic installation. Use the minimal installation pattern, but do make sure it meets at least the following requirements:
   - Put /home on a dedicated partition or logical volume.
   - Install the nmap package as well.
4. Install a virtual machine using the installation server you have just created.

Q&A