Chapter 1:

The boot process with systemd

Learning objectives

Upon completing this chapter, the learner should be able to:

- Understand and manage systemd
- Understand and manage units' types
- Understand and manage units' targets
- Start and stop services safely
- Configure services to start automatically at boot
- Manage default target in systemd
- Understand the boot procedure phases

Key terms

Systemd

Units files

System units

User units

Runtime units

Services units

Targets units

halt poweroff runlevel0 shutdown

rescue runlevel

multi−user runlevel2|3|4

graphical runlevel5

reboot runlevel6

default target

emergency hibernate suspend hybrid−sleep

Units dependencies

Units conflicts

Targets isolation

systemctl status start stop enable disable

Boot process

POST UEFI BIOS

MBR Boot Loader GRUB2

kernel initramfs

Table of content

Before starting this course, it's important to know that CentOS is a community-developed and supported alternative to RHEL. It is similar to Red Hat Enterprise Linux but lacks the enterprise-level support. CentOS is more or less a free replacement for RHEL with few minor configuration differences. So when Centos7 or RHEL7 is mentioned through this course it means the same.

1. Working with systemd

Systemd is the new service in Red Hat Enterprise Linux7 that is responsible for starting all kinds of things. Systemd goes way beyond starting services, other items are started from systemd as well, where the major benefit of working with systemd, as compared to previous methods RedHat, CentOS and most other common distributions used for managing services, is that it provides a uniform interface to start units, this interface is defined in the unit file.

Systemd is a suite of system management daemons, libraries, and utilities designed as a central management and configuration platform for the GNU/Linux computer operating system. It provides a system and service manager that runs as PID1 (Process ID) and starts the rest of the system as an alternative to the previous traditional systems.

It provides aggressive parallelization capabilities, uses socket and D–Bus activation for starting services, offers on–demand starting of daemons, the listing below showing how systemd is controlling all running processes in a Linux system:

```
[root@server1 ~]# pstree -pu
systemd(1)─┬─agetty(832)
               ├─apache2(4879)─┬─apache2(29837,www-data)

...
               │           └─apache2(29844,www-data)
                   ├─cron(566)
                   ├─dbus-daemon(565,messagebus)
                   ├─mysqld(7142,mysql)─┬─{mysqld}(7143)
...
               │           └─{mysqld}(30092)
                   ├─php-fpm7.3─┬─(22788)
                         php-fpm7.3(22790,www-data)
                   ├─php-fpm7.3(22791,www-data)
                   ├─rsyslogd(568)─┬─{rsyslogd}(571)
               │           └─{rsyslogd}(572)
               │           └─{rsyslogd}(573)
                   ├─systemd(11075,root)───(sd-pam)(11076)
                   ├─systemd-journal(330)
                   ├─systemd-logind(567)
                   ├─systemd-timesyn(535,systemd-timesync)
                                   ├─systemd-timesyn}(564)
```

To describe it in a generic way, the systemd system and service manager is used to start stuff which is referred to as units which in turn can be many things, one of the most important unit types is the service, therefore to understand the systemd it's important to understand the units concept first and how they are built and how they work.

In this chapter, you learn how systemd is organized and what items are started from systemd.

1.1. Understanding units

Systemd units are objects that are used for organizing boot and maintenance tasks, such as hardware initialization, socket creation, file system mounts, and service startups. Unit configuration is stored in their respective configuration files, which are auto – generated from other configurations, created dynamically from the system state, produced at runtime, or user–developed.

Unit files contain common and specific configuration elements:

- Common elements fall under the [Unit] and [Install] sections, and comprise description, documentation location, dependency information, conflict information, and other options that are independent of the type of unit.
- The unit specific configuration data is located under the unit type section: [Service] for the service unit type, [Socket] for the socket unit type, and so forth.

Apart from services, other unit types exist, such as sockets, targets, devices, mounts, and others. Units have a name and a type, and they are encoded in files with names in the form unitname. type. Some examples tmp.mount, sshd.service, syslog.socket, and umount.target. These files are a direct replacement of the initialization scripts found in the /etc/rc.d/init.d directory in the older RHEL releases and could be divided into three main types of unit configuration files:

- system unit files that are distributed with installed packages and located in the /user/lib/systemd/system directory.
- user unit files that are generated by users and stored in the /etc/systemd/user directory.
- runtime unit files that are created at runtime and destroyed when they are no longer needed, they are located in the /run/systemd/system directory and take precedence over the system unit files, and the user unit files take priority over the runtime files.

This information can be vetted with the pkg-config command:

```
[root@server1      ~]#      pkg-config      systemd      --
variable=systemdsystemunitdir
/usr/lib/systemd/system

[root@server1      ~]#      pkg-config      systemd      --
variable=systemduserconfdir
/etc/systemd/user
```

Units are in one of several operational states, including active, inactive, in the process of being activated or deactivated, and failed. Units can be enabled or disabled, an enabled unit can be started to an active state, a disabled unit cannot be started.

Units can have dependency relationships amongst themselves based on a sequence or a requirement. A sequence outlines one or more actions that need to be taken before or after the activation of a unit (options Before and After), and a requirement specifies what must already be running (option Requires) or not running (option Conflicts) in order for the successful launch of a unit. See Managing dependencies and conflicts for more explanations.

When working with systemd unit files, you risk getting overwhelmed with options and 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 listing below shows all systemd options that can be configured in the sshd.service unit, including their current default values:

```
[root@server1 ~]# systemctl show sshd.service
Id=sshd.service
Names=sshd.service
Requires=basic.target
Wants=sshd-keygen.service system.slice
WantedBy=multi-user.target
ConsistsOf=sshd-keygen.service
Conflicts=shutdown.target
ConflictedBy=sshd.socket
Before=shutdown.target multi-user.target
After=network.target     sshd-keygen.service     systemd-
journald.socket
basic.target system.slice
Description=OpenSSH server daemon
LoadState=loaded
ActiveState=active
SubState=running
```

Exercise: Displaying Units with systemctl

1. Type systemctl and display all active units.
2. Notice the columns UNIT, LOAD, ACTIVE and others in the output.
3. Try to display the inactive units using the option --all.
4. Try to see all available units types using the switch -t.
5. Try out the show option to see the different options available for unit files sections.

1.1.1. Service units

Typically, services are processes that provide specific functionality and allow connections from external clients coming in, so it's necessary to show and explain how the systemd works with services units, here is an example of the vsftpd service unit file:

```
[root@server1 ~]# cat /usr/lib/systemd/system/vsftpd.service
...
[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 any systemd service unit file consists of three sections. (You’ll find different sections in other types of unit files):

- **[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.

1.1.2. Mount units

Another type of a unit file is the mount units, here is an example of the tmp.mount unit which looks like below snippet:

```
[root@server1 ~]# cat /usr/lib/systemd/system/tmp.mount
...
[Unit]
Description=Temporary Directory
Documentation=man:hier(7)
Documentation=http://www.freedesktop.org/wiki/Software/sy
stemd/
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
```

The tmp.mount unit file shows some interesting additional information:

- In the **[Unit]** section, you can see the Conflicts statement, this is used to list units that cannot be used together with this unit. Use this for mutually exclusive units.
- Next, there is the **[Mount]** section, which is specific for this unit type and defines where exactly the mount has to be performed. You'll recognize the arguments that are typically used in any mount command.
- Last, there is the WantedBy in the **[install]** section, which defines where the unit has to be started. See the Understanding wants.

1.1.3 Target units

Targets are a special systemd unit type with the (.target) file extension and are needed to load other units in the right order and at the right moment, so simply they are logical collections of units. Some targets inherit all services from other targets and add their own to them.

Systemd includes several predefined targets that are equivalent to SysVinit run levels, however, they are named rather than numbered and are used to execute a series of units. This is typically true for booting the system to a specific operational level (similar to a numbered run level) with all the required services up and running at that level.

It's always possible to know the current default running target by typing systemctl get - default, and to change it use the systemctl set-default to set the desired default target. To change to a different target unit in the current session, we can use the systemctl isolate name.target. Remember that some targets might need extra pages to be installed in your system like the graphical target that requires the GNOME and GUI stuff installed correctly.

Here is a list of these targets with the comparable run level associated with each one of them:

Target	SysVinit Run Level
halt, poweroff or runlevel0, shutdown	0
rescue or runlevel1	1, s, or single
Multi–user or runlevel2|3|4	3
graphical or runlevel5	5
reboot or runlevel6	6
default	Typically set to 3 or 5
emergency, hibernate, suspend, hybrid–sleep	N/A

Some targets, such as the multi–user.target and the graphical.target, define a specific state that the system needs to enter, but other targets just bundle a group of units together, such as the nfs.target and the printer.target and they are included from other targets, like the multi–user or graphical targets.

On the other hand, targets by themselves can have dependencies to other targets, for example see the multi-user.target which defines the normal operational state of a RHEL server:

```
[root@server1   ~]#   cat   /usr/lib/systemd/system/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
```

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. Also, it defines the target ability to be isolated or not in the statement AllowIsolate, this is used to determine whether this unit is affected when others call the system to start any isolating units. See Isolating targets for more information.

- 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 Understanding wants section.

Isolating targets

Isolating targets in simple words is to run the requested unit with all its dependencies and making sure nothing else is running, but what if other targets should be running and mustn't be stopped? in this case those targets should set the flag IgnoreOnIsolate to yes in their configuration files.

Best case example to use isolation are those like poweroff, rescue, multi-user, graphical or reboot targets as mentioned above they must set AllowIsolate=yes. That means that you can switch the current state of your computer to either one of these targets using the systemctl isolate command, it's important to know that running this command on a target puts the system in that state for the current session only and the system will return to its default the next reboot.

1.1.4. Understanding wants

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 named by the target name followed by. wants word, this directory contains wants as symbolic links to specific services that are to be started. This means that services know for themselves in which targets they need to be started and a want is created automatically in that target.

The snippet below shows the default target as a symbolic link to the multi−user target and the associated wants subdirectory with related services targets in it.

```
[root@server1 ~]# ls -l /etc/systemd/system
...
lrwxrwxrwx.  ...  default.target  ->  /lib/systemd/system/multi-
user.target
drwxr-xr-x. ... default.target.wants
...
[root@server1 ~]# ls -l /etc/systemd/system/default.target.wants/
...
lrwxrwxrwx. ... systemd-readahead-collect.service ->
/usr/lib/systemd/system/systemd-readahead-collect.service
lrwxrwxrwx. ... systemd-readahead-replay.service ->
/usr/lib/systemd/system/systemd-readahead-replay.service
```

1.2. Managing units through systemd

After understanding the systemd concepts and its units' elements, this section presents how to manage and control these units using mainly the systemctl command, which comes with the default installation of any Linux distribution that uses the systemd service.

Basically, Linux administrators will often need to get a current overview of the current status of systemd units, to do so a set of systemctl options and sub-commands can help to get this insight, some of which are shown in the following table:

Command	Description
systemctl	Shows all active and loaded units.
systemctl --all	Shows all loaded units whether they are active or not.
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 list-unit-files	Shows every available unit file within the systemd paths, including those that systemd has not attempted to load.
systemctl cat your.service	Shows the unit file that systemd has loaded into its system.
systemctl status -l your.service	Shows detailed status information about services.

1.2.1. Overviewing units

Administrators need to know different kinds of information about current status of a systemd unit, the table below shows the different kinds of information that could be requested about unit files when using the systemctl status command:

Systemd 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 cannot be enabled but may be started by another unit automatically.

To display the current situation of a unit the systemctl status command is used. For example, the following snippet displays the ftp service status where some lines were ellipsized and could be listed by using the –l switch:

```
[root@server1 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...
Sep 28 08:42:59 server202.example.com systemd[1]: Started Vsftpd
ftp
daemon.
```

1.2.2. Controlling units

As an administrator, you need to manage systemd units. It starts by starting and stopping units where the systemctl command is used to do that and also to make sure that it restarts automatically upon reboot where you do this by enabling or disabling the unit.

In order to achieve this systemctl command followed by start, stop, enable or disable commands is used, of course the unit name should be followed as a parameter, here is a list of available commands:

Command	Description
systemctl start unit	Starting unit immediately.
systemctl stop unit	Stop a unit immediately.
systemctl restart unit	Restart a unit.
systemctl reload unit	Ask a unit to reload its configuration.
systemctl is-enabled unit	Check whether a unit is already enabled or not.
systemctl enable unit	Enable a unit to be started on bootup, use --now option to start it immediately.
systemctl disable unit	Disable a unit to not start during bootup.
systemctl mask unit	Mask a unit to make it impossible to start it (both manually and as a dependency, which makes masking dangerous).
systemctl unmask unit	Unmask a unit.

The following procedure is a simple example that you through the steps of enabling the ftp service on our system:

1. Display the service status as below to see If it has not yet been enabled, the Loaded line will show that it currently is disabled:

```
[root@server1 ~] # 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
```

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, in the background a symbolic link is created.

Exercise: 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. 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.

1.2.3. Managing dependencies and conflicts

Systemd units in many cases have dependencies, as an example of dependencies and combination of units in systemd, the graphical.target unit file tells us that the system must already be operating in the multi-user mode and must not be running in rescue mode in order for it to boot successfully into the graphical mode.

Some units will be started as a dependency of other units, and an event where one specific unit is requested may trigger the start of another unit. As an administrator, you can request a list of unit dependencies by calling the systemctl list dependencies command followed by a unit name to find out which dependencies it has, and add the --reverse option to find out which units are dependent of this unit.

```
root@server1 ~]# systemctl list-dependencies vsftpd
vsftpd.service
├─system.slice
└─basic.target
├─alsa-restore.service
├─alsa-state.service
...
├─rhel-configure.service
├─rhel-dmesg.service
├─rhel-loadmodules.service
├─paths.target
├─slices.target
│ ├─slice
│ └─system.slice
├─sockets.target
│ ├─avahi-daemon.socket
│ ├─cups.socket
```

Apart from dependencies, some units have conflicts with other units. If units have conflicts with other units, this is described in the unit file. As an administrator, you can also make sure that conflicting units will never be loaded at the same time on the same system, examples of these include the following:

- Mount and umount units that cannot be loaded together
- The network and NetworkManager service
- The iptables and the firewalld service
- The cronyd and ntpd service

To prevent conflicts, the systemctl mask command is used, which basically makes a unit no longer a candidate for being started. Apply the following procedure to find out how it works.

Exercise: Masking units

1. Open a root shell and type systemctl status firewalld. Next type systemctl status iptables. If one of the services is active, do not load it again in the next step.
2. Type systemctl start firewalld and systemctl start iptables to load both services.
3. Type cat /usr/lib/systemd/system/firewalld. service. Notice the conflicts setting. Type cat /usr/lib/systemd/system/iptables. service. Notice that this unit does not have a conflicts line.
4. Unload both services by using systemctl stop firewalld followed by systemctl stop iptables. Notice that it is not really necessary to stop the iptables service because it has failed to load, but we really need to make sure that it is not loaded at all before continuing.

5. Type systemctl mask iptables and look at what is happening: A symbolic link to /dev/null is created for /etc/systemd/system/iptables. service (as you can see in the output of the following command example). Because the unit files in /etc/systemd have precedence over the files in /usr/lib/systemd, this makes it impossible to start the iptables service by accident.

6. Type systemctl start iptables. You’ll see an error message indicating that this service is masked and for that reason cannot be started.

7. Type systemctl enable iptables. Notice that no error message is shown and it looks as if it is working all right. Restart your server using systemctl reboot (or just reboot).

8. After restart, type systemctl status –l iptables. You’ll see that it is inactive and that the loaded status is indicated as masked.

2. Linux boot process

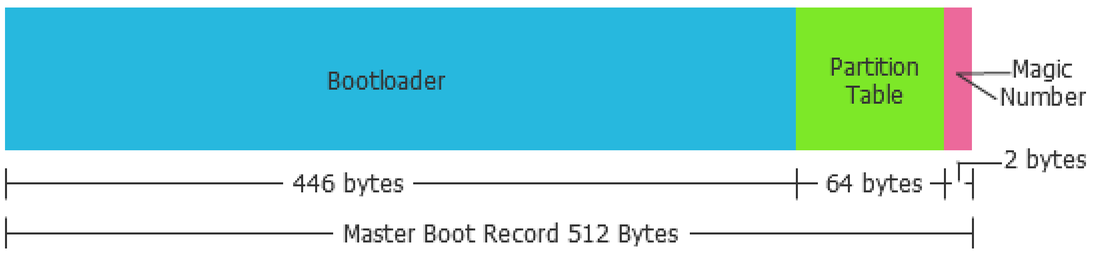
RHEL goes through the boot process when the system is powered up or reset, with the boot process lasting until all enabled services are started and a login prompt appears on the screen. The following steps summarize how the boot procedure happens on Linux.

2.1. 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.

2.2. Selecting the bootable device

Either from the UEFI boot firmware or from the Master Boot Record (MBR), a bootable device is located which is the first 512 bytes of the boot drive that is read into memory by the BIOS. The next 64 bytes contain the partition table for the disk. The last two bytes are the "Magic Number" which is used for error detection. MBR discovers the bootable device and loads the GRUB2 boot loader into memory and transfers control over to it.



2.3. Loading the boot loader

From the bootable device, a boot loader is located which is usually GRUB2 on RHEL 7, it's the first thing that needs to be working well to boot a Linux server, it is installed in the boot sector of your server's hard drive and is configured to load the Linux kernel and initramfs.

One of the most important differences between GRUB2 and its previous version is the availability of GRUB2 modules, where a large number of modules are available and they determine what you can and what you cannot do from the GRUB2 boot loader.

2.4. Loading the kernel

The kernel is the heart of the operating system, allowing users to interact with the hardware that is installed in the server. 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 which 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 RHEL7, the initramfs contains a complete operational system (which may be used for trouble-shooting purposes).

The initramfs contains drivers that are needed to start your server as well as 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).

2.5. Starting /sbin/init

Once the kernel is loaded into memory, the first of all processes is loaded, but 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.

2.6. 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.

2.7. 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.

2.8. 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.

Quiz

Chapter review questions

1. Both BIOS and UEFI are used in newer computers?

   a. true

   b. false

2. By default, GRUB is stored in the master boot record?

   a. true

   b. false

3. Which command is used to manage system services?

   a. systemd

   b. systemctl

   c. sysVinit

4. Name the two directory paths where systemd unit files are stored.

   a. /etc/systemd/system

   b. /etc/lib/systemd/system

   c. /usr/lib/systemd/system

   d. /bin/systemd/system

5. What would the command systemctl restart rsyslog do?

   a. start the rsyslog target

   b. restart the systemctl service

   c. restart the rsyslog service

6. In which target does the X window and the graphical desktop interface become available?

   a. graphical target

   b. multi-user target

   c. rescue target

7. What is the new system initialization scheme introduced in RHEL7?

   a. systemd

   b. sysctl

   c. sysVinit

   d. systemctl

8. Which command enables you to make sure that a target is no longer eligible for automatic start on system boot?

   a. systemctl disable unit

   b. systemctl enable unit

   c. systemctl set-default unit

9. Which command should you use to show all service units that are currently loaded?

   a. systemd

   b. sysctl

   c. sysVinit

10. How do you create a want for a service?

   a. by masking this service

   b. by creating a symbolic link in the/etc/systemd/system directory

   c. by isolating this service

11. How do you switch the current operational target to the rescue target?

   a. target rescue. target start

   b. systemctl resc. target

   c. systemctl isolate rescue. target

12. Why can it happen that you get the message that a target cannot be isolated?

   a. if set AllowIsolate is not set to yes in their configuration file

   b. if systemctl command is not available

   c. if the target is already running

13. You want to shut down a systemd service, but before doing that you want to know which other units have dependencies to this service. Which command would you use?

   a. systemctl list –dependencies service

   b. systemctl dependencies service

   c. systemd list –dependencies service

14. Which command shows all service unit files on your system that are currently loaded?

a. systemctl –type=service

b. systemctl –type=service −−all

c. systemctl –list −services

d. systemctl –show −units | grep services

15. What is the best solution to avoid conflicts between incompatible units?

a. Nothing, the unit files have defined for themselves which units they are not compatible with.

b. Disable the service using systemctl disable.

c. Unmask the service using systemctl unmask.

d. Mask the service using systemctl mask.

16. Which of the following is not a valid status for systemd services?

a. Running (active)

b. Running (exited)

c. Running (waiting)

d. Running (dead)

17. To allow targets to be isolated, you need a specific statement in the target unit file. Which of the following describes that statement?

a. AllowIsolate

b. Isolate

c. SetIsolate

d. Isolated

18. Which of the following is not a valid systemd unit type?

a. service

b. udev

c. mount

d. socket

19. You want to find out which other systemd units have dependencies to this specific unit. Which command would you use?

a. systemd list − dependencies −− reverse

b. systemctl list − dependencies −− reverse

c. systemctl status my. unit – show − deps

d. systemd status my. unit – show − deps – r

20. Which of the following is not a valid command while working with units in systemctl?

a. systemctl unit start

b. systemctl status –l unit

c. systemctl mask unit

d. systemctl disable unit

Answers to chapter Review Questions:

1. a
2. a
3. b
4. a and c
5. c
6. a
7. a
8. a
9. b
10. b
11. c
12. a
13. a
14. a
15. a
16. d
17. a
18. b
19. b
20. a