Maintaining System Startup and Services

- ✓ Objective 1.4: Given a scenario, configure and use the appropriate processes and services
- ✓ Objective 4.5: Given a scenario, use systemd to diagnose and resolve common problems with a Linux system

After your Linux system has traversed the boot process, it enters final system initialization, where it needs to start various services. A service, or daemon, is a program that performs a particular duty. Several services were covered in Chapter 2, "Introduction to Services."

The initialization daemon (`init`) determines which services are started and in what order. This daemon also allows you to stop and manage the various system services.

The SysV init (SysV) was based on the Unix System V initialization daemon. While it's not used by many major Linux distributions anymore, you still may find it lurking around that older Linux server at your company.

The systemd initialization method is the new kid on the block. Started around 2010, it is now the most popular system service initialization and management mechanism. This daemon reduces initialization time by starting services in a parallel manner.

Beyond initialization, these daemons are also responsible for managing these services well past boot time. We'll explore these concepts in this chapter.

Looking at *init*

Before we start examining the various service management methods, it's a good idea to take a look at the `init` program itself. Classically, service startups are handled by the `init` program. This program can be located in the `/etc/`, the `/bin/`, or the `/sbin/` directory. Also, it typically has a process ID (PID) of 1.

The `init` program or systemd is the parent process for every service on a Linux system. If your system has the `pstree` program installed, you can see a diagram depicting this relationship by typing in **`pstree -p 1`** at the command line.

This information will assist you in determining which system initialization method your current Linux distribution is using, systemd or SysV init. First find the `init` program's location, using the `which` command. An example is shown in Listing 6.1.

Listing 6.1 Finding the `init` program file location

```
# which init
/sbin/init
#
```

Now that you know the `init` program's location, using super user privileges, you can use the `readlink -f` command to see if the program is linked to another program, as shown in Listing 6.2.

Listing 6.2 Checking the `init` program for links

```
# readlink -f /sbin/init
/usr/lib/systemd/systemd
#
```

You can see in Listing 6.2 that this system is actually using systemd. You can verify this by taking a look at PID 1, as shown in Listing 6.3.

Listing 6.3 Checking PID 1

```
# ps -p 1
  PID TTY          TIME CMD
    1 ?        00:00:06 systemd
#
```

In Listing 6.3, the `ps` utility is used. This utility allows you to view processes. A process is a running program. The `ps` command shows you what program is running for a particular process in the `CMD` column. In this case, the systemd program is running. Thus, this Linux system is using systemd.

Keep in mind that these discovery methods are not foolproof. Also, there are other system initialization methods, such as the now-defunct Upstart. The following brief list shows a few Linux distribution versions that used Upstart:

- Fedora v9–v14
- openSUSE v11.3–v12.2
- RHEL v6
- Ubuntu v6.10–v15.04

If you are using the distribution versions recommended in Chapter 1, "Preparing Your Environment," be aware that those distributions are all systemd systems.

Managing systemd Systems

The systemd approach introduced a major paradigm shift in how Linux systems manage services. Services can now be started when the system boots, when a particular hardware component is attached to the system, when certain other services are started, and so on. Some services can be started based on a timer.

In the following sections, we'll focus on starting, stopping, and controlling systemd managed services. We'll walk you through the systemd technique's structures, commands, and configuration files.

Exploring Unit Files

The easiest way to start exploring systemd is through the *systemd units*. A unit defines a service, a group of services, or an action. Each unit consists of a name, a type, and a configuration file. There are currently 12 different systemd unit types, as follows:

- automount
- device
- mount
- path
- scope
- service
- slice
- snapshot
- socket
- swap
- target
- timer

The `systemctl` utility is the main gateway to managing systemd and system services. Its basic syntax is as follows:

```
systemctl [OPTIONS...] COMMAND [NAME...]
```

You can use the `systemctl` utility to provide a list of the various units currently loaded in your Linux system. A snipped example is shown in Listing 6.4.

Listing 6.4 Looking at systemd units

```
$ systemctl list-units
UNIT                       LOAD   ACTIVE SUB       DESCRIPTION
[...]
smartd.service             loaded active running   Self Monitor[...]
sshd.service               loaded active running   OpenSSH serv[...]
sysstat.service            loaded active exited    Resets Syste[...]
[...]
graphical.target           loaded active active    Graphical I[...]
[...]
$
```

In Listing 6.4 you can see various units as well as additional information. Units are identified by their name and type using the format *`name.type`*. System services (daemons) have unit files with the `.service` extension. Thus, the secure shell (SSH) daemon, `sshd`, has a unit filename of `sshd.service`.

TIP

Many displays from the `systemctl` utility use the `less` pager by default. Thus, to exit the display, you must press the Q key. If you want to turn off the `systemctl` utility's use of the `less` pager, tack the `--no-pager` option on the command.

Groups of services are started via target unit files. At system startup, the `default.target` unit is responsible for ensuring that all required and desired services are launched at system initialization. It is set up as a symbolic link to another target unit file, as shown in Listing 6.5 on an Ubuntu distribution.

Listing 6.5 Looking at the default.target link

```
$ find / -name default.target 2>/dev/null
/usr/lib/systemd/system/default.target
[...]
$ readlink -f /usr/lib/systemd/system/default.target
/usr/lib/systemd/system/graphical.target
$ systemctl get-default
graphical.target
$
```

First, in Listing 6.5, the `default.target` unit's full filename is located via the `find` utility. The `readlink` command is then employed to find the actual target file, which determines what services to start at system boot. In this case, the target unit file is `graphical.target`.

Also notice in Listing 6.5 that the `systemctl` command is much easier to use than the other two commands. It is simply `systemctl get-default`, and it displays the actual target file. Due to the `default.target` file being located in different directories on the different distros, it is always best to use the `systemctl` utility. Table 6.1 shows the more commonly used system boot target unit files.

TABLE 6.1 Commonly used system boot target unit files

Name	Description
`graphical.target`	Provides multiple users access to the system via local terminals and/or through the network. Graphical user interface (GUI) access is offered.
`multi-user.target`	Provides multiple users access to the system via local terminals and/or through the network. No GUI access is offered.
`network-online.target`	Provides a target that runs after the system has established a connection to the network. This is useful for starting applications that require the network to be present.
`runlevel`*n*`.target`	Provides backward compatibility to SysV init systems, where *n* is set to 1–5 for the desired SysV runlevel equivalence.

In Table 6.1, you'll notice that systemd provides backward compatibility to the classic SysV init systems. The SysV runlevels will be covered later in this chapter.

TIP

The master systemd configuration file is the `/etc/systemd/system.conf` file. In this file you will find all the default configuration settings commented out via a hash mark (#). Viewing this file is a quick way to see the current systemd configuration. If you need to modify the configuration, just edit the file. However, it would be wise to peruse the file's man page first by typing **`man systemd-system.conf`** at the command line.

Focusing on Service Unit Files

Service unit files contain information, such as which environment file to use, when a service must be started, what targets want this service started, and so on. These configuration files are located in different directories.

Keep in mind that a unit configuration file's directory location is critical, because if a file is found in two different directory locations, one will have precedence over the other. The following list shows the directory locations in ascending priority order:

1. `/etc/systemd/system/`
2. `/run/systemd/system/`
3. `/usr/lib/systemd/system/`

To see the various service unit files available, you can again employ the `systemctl` utility. However, a slightly different command is needed than when viewing units, as shown in Listing 6.6.

Listing 6.6 Looking at systemd unit files

```
$ systemctl list-unit-files
UNIT FILE                                    STATE
[...]
dev-hugepages.mount                          static
dev-mqueue.mount                             static
proc-fs-nfsd.mount                           static
[...]
nfs.service                                  disabled
nfslock.service                              static
ntpd.service                                 disabled
ntpdate.service                              disabled
[...]
ctrl-alt-del.target                          disabled
default.target                               static
emergency.target                             static
[...]
$
```

Besides the unit file's base name, you can also see a unit file's state in Listing 6.6. Their states are called enablement states and refer to when the service is started. There are at least 12 different enablement states, but you'll commonly see these three:

- `enabled`: Service starts at system boot.
- `disabled`: Service does not start at system boot.
- `static`: Service starts if another unit depends on it. Can also be manually started.

To see what directory or directories store a particular systemd unit file(s), use the `systemctl` utility. An example on an Ubuntu distribution is shown in Listing 6.7.

Listing 6.7 Finding and displaying a systemd unit file

```
$ systemctl cat cron.service
# /lib/systemd/system/cron.service
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)
After=remote-fs.target nss-user-lookup.target

[Service]
EnvironmentFile=-/etc/default/cron
ExecStart=/usr/sbin/cron -f $EXTRA_OPTS
IgnoreSIGPIPE=false
KillMode=process
Restart=on-failure

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

Notice in Listing 6.7 that the first displayed line shows the `cron.service` unit file's base name and directory location. The next several lines are the unit configuration file's contents.

For service unit files, there are three primary configuration sections:

- `[Unit]`
- `[Service]`
- `[Install]`

Within the service unit configuration file's `[Unit]` section, there are basic *directives*. A directive is a setting that modifies a configuration, such as the `After` setting shown in Listing 6.7. Commonly used `[Unit]` section directives are described in Table 6.2.

TABLE 6.2 Commonly used service unit file `[Unit]` section directives

Directive	Description
`After`	Sets this unit to start after the designated units.
`Before`	Sets this unit to start before the designated units.
`Description`	Describes the unit.
`Documentation`	Sets a list of Uniform Resource Identifiers (URIs) that point to documentation sources. The URIs can be web locations, particular system files, info pages, and man pages.
`Conflicts`	Sets this unit to *not* start with the designated units. If any of the designated units start, this unit is not started. (Opposite of `Requires`.)
`Requires`	Sets this unit to start together with the designated units. If any of the designated units do not start, this unit is *not* started. (Opposite of `Conflicts`.)
`Wants`	Sets this unit to start together with the designated units. If any of the designated units do not start, this unit is *still* started.

TIP

There is a great deal of useful information in the man pages for systemd and unit configuration files. Just type **`man -k systemd`** to find several items you can explore. For example, explore the service type unit file directives and more with the `man systemd.service` command. You can find information on all the various directives by typing **`man systemd.directives`** at the command line.

The `[Service]` directives within a unit file set configuration items, which are specific to that service. Commonly used `[Service]` section directives are described in Table 6.3.

TABLE 6.3 Commonly used service unit file `[Service]` section directives

Directive	Description
`ExecReload`	Indicates scripts or commands (and options) to run when unit is reloaded.
`ExecStart`	Indicates scripts or commands (and options) to run when unit is started.
`ExecStop`	Indicates scripts or commands (and options) to run when unit is stopped.
`Environment`	Sets environment variable substitutes, separated by a space.

Directive	Description
`Environment File`	Indicates a file that contains environment variable substitutes.
`RemainAfterExit`	Set to either no (default) or `yes`. If set to `yes`, the service is left active even when the process started by `ExecStart` terminates. If set to `no`, then `ExecStop` is called when the process started by `ExecStart` terminates.
`Restart`	Service is restarted when the process started by `ExecStart` terminates. Ignored if a `systemctl restart` or `systemctl stop` command is issued. Set to no (default), `on-success`, `on-failure`, `on-abnormal`, `on-watchdog`, `on-abort`, or `always`.
`Type`	Sets the startup type.

The `[Service] Type` directive needs a little more explanation than what is given in Table 6.3. This directive can be set to at least six different specifications, of which the most typical are listed here:

- `forking`: `ExecStart` starts a parent process. The parent process creates the service's main process as a child process and exits.
- `simple`: (Default) `ExecStart` starts the service's main process.
- `oneshot`: `ExecStart` starts the service's main process, which is typically a configuration setting or a quick command, and the process exits.
- `idle`: `ExecStart` starts the service's main process, but it waits until all other start jobs are finished.

NOTE You will only find a unit file `[Service]` section in a service unit file. This middle section is different for each unit type. For example, in auto mount unit files, you would find an `[Automount]` section as the middle unit file section.

Another `[Service]` configuration setting that needs additional explanation is the `Environment` directive. Linux systems use a feature called *environment variables* to store information about the shell session and working environment (thus the name *environment variable*). If you want to ensure that a particular environment variable is set properly for your service, you will want to employ the `Environment` directive. A snipped example on a Rocky Linux distribution is shown in Listing 6.8.

Listing 6.8 Viewing a service unit file's `Environment` directive

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:[...]
$
```

```
$ systemctl --no-pager cat anaconda.service
# /usr/lib/systemd/system/anaconda.service
[Unit]
Description=Anaconda
[...]
[Service]
Type=forking
Environment=HOME=/root MALLOC_CHECK_=2 MALLOC_PERTURB_=204 PATH=/usr/bin:/
bin:/sbin:/usr/sbin:/mnt/sysimage/bin: [...]
LANG=en_US.UTF-8 [...]
[...]
$
```

In Listing 6.8, you can see that the `PATH` environment variable's contents are displayed. This environment variable is a colon-separated list of directories where the process looks for commands. The `anaconda.service` unit file uses the `Environment` directive to set particular environment variables to its own desired environment parameters. These parameters are separated by a space. You can see in Listing 6.8 that one of those parameters set by the `Environment` directive is `PATH`.

NOTE Some service type unit files use the `EnvironmentFile` directive, instead of the `Environment` directive. This directive points to a file containing environment parameters. The `cron.service` unit file shown in Listing 6.7 does just that.

The `[Install]` directives within a unit file determine what happens to a particular service if it is enabled or disabled. An *enabled service* is one that starts at system boot. A *disabled service* is one that does *not* start at system boot. Commonly used `[Install]` section directives are described in Table 6.4.

TABLE 6.4 Commonly used service unit file `[Install]` section directives

Directive	Description
`Alias`	Sets additional names that can be used to denote the service in `systemctl` commands.
`Also`	Sets additional units that must be enabled or disabled for this service. Often the additional units are socket type units.
`RequiredBy`	Designates other units that require this service.
`WantedBy`	Designates which target unit manages this service.

Focusing on Target Unit Files

For systemd, you need to understand the service unit files as well as the target unit files. The primary purpose of target unit files is to group together various services to start at system boot time. The default target unit file, `default.target`, is symbolically linked to the target unit file used at system boot. In Listing 6.9, the default target unit file is located and displayed using the `systemctl` command.

Listing 6.9 Finding and displaying the systemd target unit file

```
$ systemctl get-default
graphical.target
$
$ systemctl cat graphical.target
# /usr/lib/systemd/system/graphical.target
[...]
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
$
```

Notice in Listing 6.9 that the `graphical.target` unit file has many of the same directives as a service unit file. These directives were described back in Table 6.2. Of course, these directives apply to a target type unit file instead of a service type unit file. For example, the `After` directive in the `graphical.target` unit file sets this target unit to start after the designated units, such as `multi-user.target`. Target units, similar to service units, have various target dependency chains as well as conflicts.

In Listing 6.9, there is one directive we have not covered yet. The `AllowIsolate` directive, if set to `yes`, permits this target file to be used with the `systemctl isolate` command. This command is covered later in this chapter.

Real World Scenario

Modifying Systems Configuration Files

Occasionally you may need to change a particular unit configuration file for your Linux system's requirements or add additional components. However, be careful when doing this task. You should not modify any unit files in the `/lib/systemd/system/` or `/usr/lib/systemd/system/` directory.

To modify a unit configuration file, copy the file to the `/etc/systemd/system/` directory and modify it there. This modified file will take precedence over the original unit file left in the original directory. Also, it will protect the modified unit file from software updates.

If you just have a few additional components, you can extend the configuration. Using super user privileges, create a new subdirectory in the `/etc/systemd/system/` directory named `service.`*`service-name`*`.d`, where *`service-name`* is the service's name. For example, for the OpenSSH daemon, you would create the `/etc/systemd/system/service.sshd.d` directory. This newly created directory is called a drop-in file directory, because you can drop in additional configuration files. Create any configuration files with names like *`description`*`.conf`, where *`description`* describes the configuration file's purpose, such as `local` or `script`. Add your modified directives to this configuration file.

After making these modifications, there are a few more needed steps. Find and compare any unit file that overrides another unit file by issuing the `systemd-delta` command. It will display any unit files that are duplicated, extended, redirected, and so on. Review this list. It will help you avoid any unintended consequences from modifying or extending a service unit file.

To have your changes take effect, issue the `systemctl daemon-reload` command for the service whose unit file you modified or extended. After you accomplish that task, issue the `systemctl restart` command to start or restart the service. These commands are explained in the next section.

Looking at *systemctl*

You may have noticed that while there are various commands to manage systemd and system services, it is easier and faster to employ the `systemctl` utility.

There are several basic `systemctl` commands available for you to manage system services. One that is often used is the `status` command. It provides a wealth of information. A couple of snipped examples on an Ubuntu distro are shown in Listing 6.10.

Listing 6.10 Viewing a service unit's status via `systemctl`

```
$ systemctl status cron
 cron.service - Regular background program processing daemon
     Loaded: loaded (/lib/systemd/system/cron.service; disabled; vendor
preset: >
     Active: inactive (dead) since Sat 2021-11-20 14:28:12 EST; 5s ago
       Docs: man:cron(8)
    Process: 593 ExecStart=/usr/sbin/cron -f $EXTRA_OPTS (code=killed,
signal=T>
   Main PID: 593 (code=killed, signal=TERM)
```

```
Nov 20 14:13:44 ubuntu20 systemd[1]: Started Regular background program
process>
Nov 20 14:13:44 ubuntu20 cron[593]: (CRON) INFO (pidfile fd = 3)
Nov 20 14:13:44 ubuntu20 cron[593]: (CRON) INFO (Running @reboot jobs)
Nov 20 14:17:01 ubuntu20 CRON[2329]: pam_unix(cron:session): session opened
for>
Nov 20 14:17:01 ubuntu20 CRON[2330]: (root) CMD (   cd /  & & run-parts
--report >
Nov 20 14:17:01 ubuntu20 CRON[2329]: pam_unix(cron:session): session closed
for>
Nov 20 14:28:12 ubuntu20 systemd[1]: Stopping Regular background program
proces>
Nov 20 14:28:12 ubuntu20 systemd[1]: cron.service: Succeeded.
Nov 20 14:28:12 ubuntu20 systemd[1]: Stopped Regular background program
process>
$ systemctl status sshd
 ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: e>
     Active: active (running) since Sat 2021-11-20 14:13:45 EST; 12min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 686 (sshd)
      Tasks: 1 (limit: 9469)
     Memory: 2.3M
     CGroup: /system.slice/ssh.service
             └686 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

Nov 20 14:13:44 ubuntu20 systemd[1]: Starting OpenBSD Secure Shell server...
Nov 20 14:13:45 ubuntu20 sshd[686]: Server listening on 0.0.0.0 port 22.
Nov 20 14:13:45 ubuntu20 sshd[686]: Server listening on :: port 22.
Nov 20 14:13:45 ubuntu20 systemd[1]: Started OpenBSD Secure Shell server.
$
```

In Listing 6.10, the first `systemctl` command shows the status of the `cron` service. Notice the third line in the utility's output. It states that the service is `disabled`. The fourth line states that the service is `inactive`. In essence, this means that the `cron` service is not running (`inactive`) and is not configured to start at system boot time (`disabled`). Another item to look at within the `cron` service's status is the `Loaded` line. Notice that the unit file's complete filename and directory location are shown.

The status of the `sshd` service is also displayed, showing that `sshd` is running (`active`) and configured to start at system boot time (`enabled`).

There are several simple commands you can use with the `systemctl` utility to manage systemd services and view information regarding them. Common commands are listed in Table 6.5. These `systemctl` commands generally use the following syntax:

```
systemctl COMMAND UNIT-NAME...
```

TABLE 6.5 Commonly used `systemctl` service management commands

Command	Description
`daemon-reload`	Load the unit configuration file of the running designated unit(s) to make unit file configuration changes without stopping the service. Note that this is different from the `reload` command.
`disable`	Mark the designated unit(s) to *not* be started automatically at system boot time.
`enable`	Mark the designated unit(s) to be started automatically at system boot time.
`mask`	Prevent the designated unit(s) from starting. The service cannot be started using the `start` command or at system boot. Use the `--now` option to immediately stop any running instances as well. Use the `--running` option to mask the service only until the next reboot or `unmask` is used.
`restart`	Stop and immediately restart the designated unit(s). If a designated unit is not already started, this will simply start it.
`start`	Start the designated unit(s).
`status`	Display the designated unit's current status.
`stop`	Stop the designated unit(s).
`reload`	Load the service configuration file of the running designated unit(s) to make service configuration changes without stopping the service. Note that this is different from the `daemon-reload` command.
`unmask`	Undo the effects of the `mask` command on the designated unit(s).

Notice the difference in Table 6.5 between the `daemon-reload` and the `reload` command. This is an important difference. Use the `daemon-reload` command if you need to load systemd unit file configuration changes for a running service. Use the `reload` command to load a service's modified configuration file. For example, if you modified the `ntpd` service's configuration file, `/etc/ntp.conf`, and wanted the new configuration to take immediate effect, you would issue the command **`systemctl reload ntpd`** at the command line.

WARNING

Use caution when employing the `systemctl mask` command on a service. This links the service to the `/dev/null` (black hole) to prevent any kind of service startup. This has been described as the "third level of off." You will not be able to start the service manually. Also, the service will *not* start at boot time if you did not employ the `--running` option when you used `mask` on it. You can reenable the ability to start the service by using the `systemctl unmask` command on it.

Besides the commands in Table 6.7, there are some other handy `systemctl` commands you can use for managing system services. An example on a CentOS distro is shown in Listing 6.11.

Listing 6.11 Determining if a service is running by using `systemctl`

```
# systemctl stop sshd
#
# systemctl is-active sshd
inactive
#
# systemctl start sshd
#
# systemctl is-active sshd
active
#
```

In Listing 6.11, the OpenSSH daemon (`sshd`) is stopped using `systemctl` and its `stop` command. Instead of the `status` command, the `is-active` command is used to quickly display that the service is stopped (`inactive`). The OpenSSH service is started back up and again the `is-active` command is used, showing that the service is now running (`active`). Table 6.6 describes these useful service status checking commands.

TABLE 6.6 Convenient `systemctl` service status commands

Command	Description
`is-active`	Displays `active` for running services and `failed` for any service that has reached a failed state
`is-enabled`	Displays `enabled` for any service that is configured to start at system boot and `disabled` for any service that is *not* configured to start at system boot
`is-failed`	Displays `failed` for any service that has reached a failed state and `active` for running services

Services can fail for many reasons: for hardware issues, a missing dependency set in the unit configuration file, an incorrect permission setting, and so on. You can employ the `systemctl` utility's `is-failed` command to see if a particular service has failed. An example is shown in Listing 6.12.

Listing 6.12 Determining if a service has failed by using `systemctl`

```
$ systemctl is-failed NetworkManager-wait-online.service
failed
$
$ systemctl is-active NetworkManager-wait-online.service
failed
$
```

In Listing 6.12, you can see that this particular service has failed. Actually, it was a failure forced by disconnecting the network cable prior to boot, so you could see a service's `failed` status. If the service was not in failed state, the `is-failed` command would show an `active` status.

The `systemctl` program is a handy tool to use when troubleshooting systemd issues, such as unit name resolution problems and services not starting on time. Since the systemd startup method can control so many aspects of your Linux system, it's a good idea to have a handle on just what `systemctl` can do for you.

Examining Special systemd Commands

The `systemctl` utility has several commands that go beyond service management. Also, systemd has some special commands. You can manage what targets (groups of services) are started at system boot time, jump between various system states, and even analyze your system's boot time performance. We'll look at these various commands in this section.

One special command to explore is the `systemctl is-system-running` command. An example of this command is shown in Listing 6.13.

Listing 6.13 Determining a system's operational status

```
$ systemctl is-system-running
running
$
```

You may think the status returned here is obvious, but it means all is well with your Linux system currently. Table 6.7 shows other useful statuses.

TABLE 6.7 Operational statuses provided by `systemctl is-system-running`

Status	Description
`running`	System is fully in working order.
`degraded`	System has one or more failed units.
`maintenance`	System is in emergency or recovery mode.
`initializing`	System is starting to boot.
`starting`	System is still booting.
`stopping`	System is starting to shut down.

The `maintenance` operational status will be covered shortly in this chapter. If you receive `degraded` status, however, you should review your units to see which ones have failed and take appropriate action. Use the `systemctl --failed` command to find the failed unit(s), as shown snipped in Listing 6.14.

Listing 6.14 Finding failed units

```
$ systemctl is-system-running
degraded
$
$ systemctl --failed
  UNIT         LOAD   ACTIVE SUB    DESCRIPTION
• rngd.service loaded failed failed Hardware RNG Entropy Gatherer Daemon
[...]
$
```

Other useful `systemctl` utility commands deal with obtaining, setting, and jumping the system's target. They are as follows:

- `get-default`
- `set-default`
- `isolate`

You've already seen the `systemctl get-default` command in action within Listing 6.5. This command displays the system's default target. As you may have guessed, you can set the system's default target with super user privileges via the `systemctl set-target` command.

The `isolate` command is handy for jumping between system targets. When this command is used along with a target name for an argument, all services and processes not enabled in the listed target are stopped. Any services and processes enabled and not running in the listed target are started. A snipped example is shown in Listing 6.15.

Listing 6.15 Jumping to a different target unit

```
# systemctl get-default
graphical.target
#
# systemctl isolate multi-user.target
#
# systemctl status graphical.target
[...]
   Active: inactive (dead) since Thu 2018-09-13 16:57:00 EDT; 4min 24s ago
     Docs: man:systemd.special(7)

Sep 13 16:54:41 localhost.localdomain systemd[1]: Reached target Graphical
In...
Sep 13 16:54:41 localhost.localdomain systemd[1]: Starting Graphical
Interface.
Sep 13 16:57:00 localhost.localdomain systemd[1]: Stopped target Graphical
In[...]
Sep 13 16:57:00 localhost.localdomain systemd[1]: Stopping Graphical
Interface.
[...]
#
```

In Listing 6.15, using super user privileges, the `systemctl isolate` command caused the system to jump from the default system target to the multiuser target. Unfortunately, there is no simple command to show your system's current target in this case. However, the `systemctl status` command is useful. If you use the command and give it the previous target's name (`graphical.target` in this case), you should see that it is no longer active and thus is not the current system target. Notice that a short history of the graphical target's starts and stops is also shown in the status display.

NOTE The `systemctl isolate` command can only be used with certain targets. The target's unit file must have the `AllowIsolate=yes` directive set.

Two extra special targets are rescue and emergency. These targets, sometimes called modes, are described here:

Rescue Target When you jump your system to the rescue target, the system mounts all the local filesystems, only the root user is allowed to log into the system, networking services are turned off, and only a few other services are started. The `systemctl is-system-running` command will return the `maintenance` status. Running disk utilities to fix corrupted disks is a useful task in this particular target.

Emergency Target When your system goes into emergency mode, the system only mounts the root filesystem, and it mounts it as read-only. Similar to rescue mode, it only allows the root user to log into the system, networking services are turned off, and only a few other services are started. The `systemctl is-system-running` command will return the `maintenance` status. If your system goes into emergency mode by itself, there are serious problems. This target is used for situations where even rescue mode cannot be reached.

Be aware that if you jump into either rescue or emergency mode, you'll only be able to log into the root account. Therefore, you need to have the root account password. Also, your screen may go blank for a minute, so don't panic. An example of jumping into emergency mode is shown in Listing 6.16.

Listing 6.16 Jumping to the emergency target unit

```
# systemctl isolate emergency
Welcome to emergency mode! After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, "systemctl default" or ^D to
try again to boot into default mode.
Give root password for maintenance
(or type Control-D to continue):
#
# systemctl is-system-running
maintenance
#
# systemctl list-units --type=target
UNIT              LOAD     ACTIVE  SUB     DESCRIPTION
emergency.target  loaded   active  active  Emergency Mode
[...]
#
# systemctl default
#
```

In Listing 6.16, the `systemctl` command is used to jump into emergency mode. Notice that you do not have to add the `.target` extension on the emergency target unit's filename. This is true with all systemd targets. Once you reach emergency mode, you must enter the root password at the prompt. Once you reach the command line, you can enter commands listed in the welcome display or try some additional `systemctl` commands.

NOTE

Other targets you can jump to include `reboot`, `poweroff`, and `halt`. For example, just type **`systemctl isolate reboot`** to reboot your system.

Notice in Listing 6.16 that when the `systemctl is-system-running` command is issued, the response is `maintenance` instead of `running`. Also, when the `list-units` command is employed, it shows that the `emergency.target` is active. The `systemctl default` command will cause the system to attempt to jump into the default target.

TIP

If you are using GRUB2 as your bootloader, you can reach a different target via the bootloader menu. Just move your cursor to the menu option that typically boots your system and press the E key to edit it. Scroll down and find the line that starts with the `linux16` command. Press the End key to reach the line's end. Press the spacebar and type **`systemd.unit=`*`target-name`*`.target`**, where *`target-name`* is the name of the target you want your system to activate. This is useful for emergency situations.

A handy systemd component is the `systemd-analyze` utility. With this utility, you can investigate your system's boot performance and check for potential system initialization problems. Table 6.8 contains common commands you can use with the `systemd-analyze` utility.

TABLE 6.8 Common `systemd-analyze` commands

Command	Description
`blame`	Displays the amount of time each running unit took to initialize. Units and their times are listed starting from the slowest to the fastest.
`time`	Displays the amount of time system initialization spent for the kernel, and the initial RAM filesystem, as well as the time it took for normal system user space to initialize. (Default)
`critical-chain`	Displays time-critical units in a tree format. Can pass it a unit file argument to focus the information on that particular unit.
`dump`	Displays information concerning all the units. The display format is subject to change without notice, so it should be used only for human viewing.
`verify`	Scans unit files and displays warning messages if any errors are found. Will accept a unit file name as an argument, but follows directory location precedence.

Be aware that some of the longer `systemd-analyze` displays are piped into the `less` pager utility. You can turn that feature off by using the `--no-pager` option. In Listing 6.17, using super user privileges, a few of these `systemd-analyze` commands are shown in action.

Listing 6.17 Employing the `systemd-analyze` utility

```
# systemd-analyze verify
#
# systemd-analyze verify sshd.service
#
# systemd-analyze time
Startup finished in 665ms (kernel) +
3.285s (initrd) + 58.319s (userspace) = 1min 2.269s
#
# systemd-analyze --no-pager blame
         30.419s NetworkManager-wait-online.service
[...]
          4.848s kdump.service
          4.707s firewalld.service
          4.565s tuned.service
          4.390s libvirtd.service
          4.221s lvm2-monitor.service
[...]
           632ms NetworkManager.service
           607ms network.service
[...]
             9ms sys-kernel-config.mount
#
```

The first command used in Listing 6.17 allows you to check all your system's unit files for problems. The second one only checks the `sshd.service` unit file. If you just receive a prompt back from those two commands, it indicates there were no errors found.

The third command in Listing 6.17 provides time information concerning your system's initialization. Note that you could leave off the `time` keyword, and the `systemd-analyze` utility would still display the system initialization time because that is the default utility action.

The last command in Listing 6.17 employs the `blame` command. This display starts with those units that took the longest to initialize. At the bottom of the list are the units that initialized the fastest. It is a handy guide for troubleshooting initialization problems. Now if only you could use `systemd-analyze blame` to analyze your friends who are always late.

The systemd initialization approach is flexible and reliable for operating Linux systems and their services. The preceding sections provided an overview of the methods and commands for managing systemd initialized systems.

Managing SysV init Systems

Many server administrators have gone through the process of moving from a SysV init system to a systemd system. Recall that systemd is backward compatible with SysV init, so understanding SysV init is important.

First, if you want to experiment with the original SysV init commands without interference from systemd or the now defunct Upstart, find a Linux distribution that uses the SysV init initialization method. One way to find one is to visit the DistroWatch website and use their search tool at `https://distrowatch.com/search.php`. Scroll down to the Search by Distribution Criteria section, and for Init software, select SysV. Any Linux distributions still using SysV init will display in the search results.

To get clean SysV init listings for this book, we used a blast from the Linux distribution past, Fedora 7. To grab an ISO copy of this old distribution, visit `https://archives.fedoraproject.org/pub/archive/fedora/linux/releases`.

WARNING

Using any older and no-longer-supported Linux distribution can open up your system to a whole host of problems. If you do choose to take this risk, minimize your exposure by putting the Linux distribution in a virtualized environment; do not install any network interface cards (NICs) for the virtual machine, and turn off access to the host machine's filesystem.

The next section should provide you with enough of a SysV init understanding to help in the Linux server migration process to systemd.

Understanding Runlevels

At system boot time, instead of targets to determine what groups of services to start, SysV init uses runlevels. These runlevels are defined in Table 6.9 and Table 6.10. Notice that different distributions use different runlevel definitions.

TABLE 6.9 Red Hat–based distribution SysV init runlevels

Runlevel	Description
0	Shut down the system.
1, s, or S	Single-user mode used for system maintenance.
2	Multiuser mode without networking services enabled.
3	Multiuser mode with networking services enabled.

Runlevel	Description
4	Custom.
5	Multiuser mode with GUI available.
6	Reboot the system.

Note that runlevels 0 and 6 are not runlevels by definition. Instead, they denote a transition from the current system state to the desired state. For example, a running system currently operating at runlevel 5 is transitioned to a powered-off state via runlevel 0.

TABLE 6.10 Debian-based distribution SysV init runlevels

Runlevel	Description
0	Shut down the system.
1	Single-user mode used for system maintenance.
2	Multiuser mode with GUI available.
6	Reboot the system.

To determine your system's current and former runlevel, you employ the `runlevel` command. The first number or letter displayed indicates the previous runlevel (`N` indicates that the system is newly booted), and the second number indicates the current runlevel. An example is shown in Listing 6.18 of a newly booted Red Hat–based SysV init system, which is running at runlevel 5.

Listing 6.18 Employing the `runlevel` command

```
# runlevel
N 5
#
```

Instead of using a default target like systemd, SysV init systems employ a configuration file, `/etc/inittab`. This file used to start many different services, but in later years it only started terminal services and defined the default runlevel for a system. The file line defining the default runlevel is shown in Listing 6.19.

Listing 6.19 The `/etc/inittab` file line that sets the default runlevel

```
# grep :initdefault: /etc/inittab
id:5:initdefault:
#
```

Within Listing 6.19, notice the number `5` between the `id:` and the `:initdefault:` in the `/etc/inittab` file record. This indicates that the system's default runlevel is 5. The `initdefault` is what specifies the runlevel to enter after the system boots.

NOTE Look back at Table 6.1 in this chapter. You'll see that systemd provides backward compatibility to SysV init via runlevel targets, which can be used as the default target and/or in switching targets with the `systemctl isolate` command.

Setting the default runlevel is the first step in configuring certain services to start at system initialization. Next, each service must have an initialization script located typically in the `/etc/init.d/` directory. Listing 6.20 shows a snipped example of the various scripts in this directory. Note that the `-1F` options are used on the `ls` command to display the scripts in a single column and tack on a file indicator code. The `*` file indicator code denotes that these files are executable programs (Bash shell scripts in this case).

Listing 6.20 Listing script files in the `/etc/init.d/` directory

```
# ls -1F  /etc/init.d/
anacron*
atd*
[...]
crond*
cups*
[...]
ntpd*
[...]
ypbind*
yum-updatesd*
#
```

These initialization scripts are responsible for starting, stopping, restarting, reloading, and displaying the status of various system services. The program that calls these initialization scripts is the `rc` script, and it can reside in either the `/etc/init.d/` or the `/etc/rc.d/` directory. The `rc` script runs the scripts in a particular directory. The directory picked depends on the desired runlevel. Each runlevel has its own subdirectory in the `/etc/rc.d/` directory, as shown in Listing 6.21.

Listing 6.21 Runlevel subdirectories in the `/etc/rc.d/` directory

```
# ls /etc/rc.d/
init.d  rc0.d  rc2.d  rc4.d  rc6.d     rc.sysinit
rc      rc1.d  rc3.d  rc5.d  rc.local
#
```

Notice in Listing 6.21 that there are seven subdirectories named `rcn.d`, where *n* is a number from 0 to 6. The `rc` script runs the scripts in the `rcn.d` subdirectory for the desired runlevel. For example, if the desired runlevel is 3, all the scripts in the `/etc/rc.d/rc3.d/` directory are run. Listing 6.22 shows a snippet of the scripts in this directory.

Listing 6.22 Files in the `/etc/rc.d/rc3.d` directory

```
# ls -1F /etc/rc.d/rc3.d/
K01smolt@
K02avahi-dnsconfd@
K02NetworkManager@
[...]
K99readahead_later@
S00microcode_ctl@
S04readahead_early@
[...]
S55cups@
S99local@
S99smartd@
#
```

Notice in Listing 6.22 that the script names start with either a `K` or an `S`, are followed by a number, and then have their service name. The `K` stands for kill (stop), and the `S` stands for start. The number indicates the order in which this service should be stopped or started for that runlevel. This is somewhat similar to the `After` and `Before` directives in the systemd service type unit files.

The files in the `/etc/rc.d/rcn.d/` directories are all symbolic links to the scripts in the `/etc/init.d/` directory. Listing 6.23 shows an example of this.

Listing 6.23 Displaying the `/etc/rc.d/rc3.d/S55cups` link

```
# readlink -f /etc/rc.d/rc3.d/S55cups
/etc/rc.d/init.d/cups
#
```

The `rc` script goes through and runs all the `K` scripts first, passing a `stop` argument to each script. It then runs all the `S` scripts, passing a `start` argument to each script. This not only ensures that the proper services are started for a particular runlevel, it also allows jumping between runlevels after system initialization and thus stopping and starting certain services for that new runlevel.

NOTE

If you need to enact certain commands or run any scripts as soon as system initialization is completed, there is a file for that purpose. The `/etc/rc.local` script allows you to add additional scripts and or commands. Just keep in mind that this script is not run until all the other SysV init scripts have been executed.

Scripts are central to the SysV init process. To understand SysV init scripts, be sure to read through Chapter 25, "Deploying Bash Scripts," first. That chapter will help you understand Bash shell script basics, which in turn will help you understand the SysV init script contents.

Investigating SysV init Commands

The various SysV init commands help in starting and stopping services, managing what services are deployed at various runlevels, and jumping between runlevels on an already running Linux system. We cover the various SysV init commands in this section.

Jumping between runlevels is a little different than jumping between systemd targets. It uses the `init` or the `telinit` utility to do so. These two utilities are essentially twins and can be interchanged for each other. To jump between runlevels on a SysV init system, the basic syntax is as follows:

```
init Destination-Runlevel
telinit Destination-Runlevel
```

Listing 6.24 shows an example of jumping on a SysV init system from the current runlevel 5 to the destination runlevel 3. Note that the `runlevel` command is used to show the previous and current runlevels.

Listing 6.24 Jumping from runlevel 5 to runlevel 3

```
# runlevel
N 5
#
# init 3
#
# runlevel
5 3
#
```

NOTE: Keep in mind that you can shut down a SysV init system by entering **`init 0`** or **`telinit 0`** at the command line as long as you have the proper privileges. You can also reboot a SysV init system by typing **`init 6`** or **`telinit 6`** at the command line.

To view a SysV init managed service's status and control whether or not it is currently running, use the `service` utility. This utility has the following basic syntax:

```
service SCRIPT COMMAND [OPTIONS]
```

The `SCRIPT` in the `service` utility refers to a particular service script within the `/etc/init.d/` directory. The `service` utility executes the script, passing it the designated `COMMAND`. Service scripts typically have the same name as the service. Also, you only have to provide a script's base name and not the directory location. As an example, for the NTP service script, `/etc/init.d/ntpd`, you only need to use the `ntpd` base name.

Table 6.11 describes the various commonly used items you can employ for the `COMMAND` portion of the service utility. Keep in mind that if the `COMMAND` is not handled by the script or handled differently than it's commonly handled, you'll get an unexpected result.

TABLE 6.11 Commonly used `service` utility commands

Command	Description
`restart`	Stop and immediately restart the designated service. Note that if a designated service is not already started, a `FAILED` status will be generated on the stop attempt, and then the service will be started.
`start`	Start the designated service.
`status`	Display the designated service's current status.
`stop`	Stop the designated service. Note that if a designated service is already stopped, a `FAILED` status will be generated on the stop attempt.
`reload`	Load the service configuration file of the running designated service.This allows you to make service configuration changes without stopping the service. Note that if you attempt the reload command on a stopped service, a `FAILED` status will be generated.

It helps to see examples of the `service` utility in action. Listing 6.25 provides a few for your review.

Listing 6.25 Employing the `service` utility

```
# service httpd status
httpd is stopped
#
```

```
# service httpd start
Starting httpd:                        [  OK  ]
#
# service httpd status
httpd (pid 14124 14123 [...]) is running...
#
# service httpd stop
Stopping httpd:                        [  OK  ]
#
# service httpd status
httpd is stopped
#
# service --status-all
anacron is stopped
atd (pid 2024) is running...
[...]
ypbind is stopped
yum-updatesd (pid 2057) is running...
#
```

The last `service` utility example in Listing 6.25 is worth pointing out. This command allows you to view all the services on your system along with their current status. Keep in mind that this list will scroll by quickly, so it's a good idea to redirect its STDOUT to the `less` pager utility so that you can view the display more comfortably.

TIP

While some SysV init commands have been modified to work with systemd utilities, others, such as `service --status-all`, might produce unpredictable or confusing results. As tempting as it is to hang on to past commands, those habits may cause you problems in the future. It is best to learn native systemd commands and employ them instead.

To configure various services to start at different runlevels, there are two different commands you can use. The one you employ depends on which distribution you are using. For Red Hat–based distros using SysV init, you'll want to use the `chkconfig` utility. For Debian-based Linux distributions using SysV init, the `update-rc.d` program is the one to use.

The `chkconfig` utility has several different formats. They allow you to check what runlevels a service will start or not start on. Also, you can enable (start at system boot) or disable (not start at system boot) a particular service for a particular runlevel. Table 6.12 describes these various commonly used `chkconfig` utility formats.

TABLE 6.12 Commonly used `chkconfig` utility formats

Command	Description
`chkconfig` *`service`*	Check if the *`service`* is enabled at the current runlevel. If yes, the command returns a true (`0`). If no, the command returns a false (`1`).
`chkconfig` *`service`* `on`	Enable the *`service`* at the current runlevel.
`chkconfig` *`service`* `off`	Disable the *`service`* at the current runlevel.
`chkconfig --add` *`service`*	Enable this *`service`* at runlevels 0–6.
`chkconfig --del` *`service`*	Disable this *`service`* at runlevels 0–6.
`chkconfig --levels [`*`levels`*`]` *`service`* `on/off`	Enable (on) or disable (`off`) this *`service`* at runlevels *`levels`*, where *`levels`* can be any number from 0 through 6.
`chkconfig --list` *`service`*	Display the runlevels and whether or not the *`service`* is enabled (on) or disabled (off) for each one.

The first command in Table 6.12 can be a little confusing. Be aware that when the utility checks if the service is enabled at the current runlevel, a true or false is returned in the ? variable. Listing 6.26 shows an example of using this command and displaying the variable results.

Listing 6.26 Using the `chkconfig` utility to check service status

```
# runlevel
3 5
#
# chkconfig --list sshd
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
#
# chkconfig sshd
# echo $?
0
# chkconfig --list ntpd
ntpd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
#
```

```
# chkconfig ntpd
# echo $?
1
#
```

Notice in Listing 6.26 that the system's current runlevel is 5. The sshd service is checked using the chkconfig --list command, and you can see from the display that this service does start on runlevel 5, indicated by the 5:on shown. Therefore, the chkconfig sshd command should return a true. As soon as the command is entered and the prompt is returned, an echo $? command is entered. This displays a 0, which indicates a true was returned. Yes, 0 means true. That is confusing!

For the ntpd service in Listing 6.26, the service is not started at runlevel 5. Therefore, the chkconfig ntpd command returns a false, which is a 1.

To enable services at multiple runlevels, you'll need to employ the --level option. For this option, the runlevel numbers are listed one after the other with no delimiter in between. An example is shown in Listing 6.27.

Listing 6.27 Using the chkconfig utility to enable/disable services

```
# chkconfig --list ntpd
ntpd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
#
# chkconfig --level 35 ntpd on
#
# chkconfig --list ntpd
ntpd            0:off   1:off   2:off   3:on    4:off   5:on    6:off
#
# chkconfig --level 35 ntpd off
#
# chkconfig --list ntpd
ntpd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
#
```

If you are using a Debian-based Linux SysV init distribution, instead of the chkconfig utility, you'll need to employ the update-rc.d utility. It has its own set of options and arguments.

To start a program at the default runlevel, just use the following format:

```
update-rc.d service defaults
```

To remove the program from starting at the default runlevel, use the following format:

```
update-rc.d service remove
```

If you want to specify what runlevels the program starts and stops in, you'll need to use the following format:

```
update-rc.d -f service start 40 2 3 4 5 . stop 80 0 1 6 .
```

The `40` and `80` specify the relative order within the runlevel when the program should start or stop (from 0 to 99). This allows you to customize exactly when specific programs are started or stopped during the boot sequence.

As you can see, managing the SysV init scripts and their associated runlevels can be tricky. However, if you have to take care of one of these systems, you now understand the tools that can help you.

Digging Deeper into systemd

Though handling storage and various issues such as mounting filesystems are thoroughly covered in Chapter 11, "Handling Storage," we want to look at systemd's mount and automount units while systemd is still fresh in your mind. We feel this will help you better retain this important certification information.

Looking at systemd Mount Units

Distributions using systemd have additional options for persistently attaching filesystems. Filesystems can be specified either within the `/etc/fstab` file or within a mount unit file. A mount unit file provides configuration information for systemd to mount and control designated filesystems.

NOTE

On Linux servers using systemd, if you only use the `/etc/fstab` file, systemd still manages these filesystems. The mount points listed in `/etc/fstab` are converted into native units when either the server is rebooted or systemd is reloaded. In fact, using `/etc/fstab` for persistent filesystems is the preferred method over manually creating a mount unit file. For more information on this process, type **`man systemd-fstab-generator`** at the command line.

A single mount unit file is created for each mount point, and the filename contains the mount point's absolute directory reference. However, the absolute directory reference has its preceding forward slash (/) removed, subsequent forward slashes are converted to dashes (`-`), and any trailing forward slash is removed. Mount unit filenames also have a `.mount` extension. For example, the mount point `/home/temp/` would have a mount unit file named `home-temp.mount`.

A mount unit file's contents mimic other systemd unit files, with a few special sections and options. In Listing 6.28, using the /home/temp/ mount point, an example mount unit file is shown.

Listing 6.28 Displaying an example systemd mount unit file

```
# cat /etc/systemd/system/home-temp.mount
[Unit]
Description=Test Mount Units

[Mount]
What=/dev/sdo1
Where=/home/temp
Type=ext4
Options=defaults
SloppyOptions=on
TimeOutSec=4

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

Notice that the file has the typical three sections for a unit file, with the middle section, [Mount], containing directives specific to mount type unit files. The What directive can use the device filename or a universally unique identifier (UUID), such as /dev/disk/by-uuid/*UUID*.

The SloppyOptions directive is helpful in that if set to on, it ignores any mount options not supported by a particular filesystem type. By default, it is set to off. Another helpful directive is TimeOutSec. If the mount command does not complete by the number of designated seconds, the mount is considered a failed operation.

Be sure to include the [Install] section and set either the WantedBy or the RequiredBy directive to the desired target. If you do not do this, the filesystem will not be mounted upon a server boot.

You can manually mount and unmount the unit using the standard systemctl utility commands. Listing 6.29 contains an example of deploying the home-temp.mount unit file.

Listing 6.29 Deploying a systemd mount unit file

```
# systemctl daemon-reload home-temp.mount
#
# systemctl start home-temp.mount
#
# ls /home/temp
lost+found
#
```

In Listing 6.29, the first command loads the newly configured mount unit file. The second command has systemd mount the filesystem using the `home-temp.mount` unit file. The second command is similar to how a service is started in that it uses the `start` command. While you don't have to have the `home-temp.mount` argument and the command will work without it, (a) the argument does add clarity/education to the situation being discussed, and (b) the argument prevents other services from being reloaded, which if other services were restarted could prove problematic.

To ensure that the filesystem is properly mounted, like a service unit, you use the `systemctl` utility to obtain a mounted filesystem's status. An example is shown in Listing 6.30.

Listing 6.30 Checking a systemd mount unit's status

```
# systemctl status home-temp.mount
• home-temp.mount - Test Mount Units
   Loaded: loaded (/etc/systemd/system/home-temp.mount; [...]
   Active: active (mounted) since Sat 2019-09-14 16:34:2[...]
    Where: /home/temp
     What: /dev/sdo1
  Process: 3990 ExecMount=/bin/mount /dev/sdo1 /home/temp[...]
[...]
#
```

One additional step is required. To ensure that systemd will mount the filesystem persistently, the mount unit file must be enabled to start at boot, as other systemd units are enabled. An example is shown in Listing 6.31.

Listing 6.31 Enabling a systemd mount unit

```
# systemctl enable home-temp.mount
Created symlink from
/etc/systemd/system/multi-user.target.wants/home-temp.mount to
/etc/systemd/system/home-temp.mount.
#
```

This should all look very familiar! Keep in mind that you should only use mount unit files if you need to tweak the persistent filesystem configuration. If you do not, it's best to use an `/etc/fstab` record to persistently mount the filesystem.

Exploring Automount Units

With systemd, you can also configure on-demand mounting as well as mounting in parallel using automount units. In addition, you can set filesystems to automatically unmount upon lack of activity.

An automount unit file operates similarly to a mount unit file. The naming convention is the same, except that the filename extension is `.automount`.

Within an automount unit file, for the `[Automount]` section, only the following three directives are available:

- `Where`
- `DirectoryMode`
- `TimeOutIdleSec`

The `Where` directive is required. It is configured the exact same way as it is in mount unit files. With this directive, you set the mount point.

The `DirectoryMode` directive is not a required option. This setting determines the permissions placed on any automatically created mount point and parent directories. By default it is set to the `0755` octal code.

NOTE You can also configure an automount point in the `/etc/fstab` file. However, keep in mind that if an automount point is configured in the `/etc/fstab` file and it has a unit file, the unit file configuration will take precedence.

The `TimeOutIdleSec` directive is also not required. This particular directive allows you to set the maximum amount of time (in seconds) a mounted filesystem can be idle. Once the time limit is reached, the filesystem is unmounted. By default this directive is disabled.

Focusing on Timer Unit Files

Timer unit files allow you to define events that occur at specific dates or times, similar to how the `cron` program works (see Chapter 26, "Automating Jobs"). The timer unit files allow you to fine-tune exactly when a program starts.

Timer unit files are designated by a `.timer` file extension and include a `[Timer]` section to define the directives required to determine when to start the event. Table 6.13 describes these directives.

TABLE 6.13 Commonly used timer unit file `[Timer]` section directives

Directive	Description
`AccuracySec`	Specifies the accuracy of the timer. The default is one minute.
`OnActiveSec`	Defines the timer relative to the moment the timer is activated.
`OnBootSec`	Defines the timer relative to when the system was booted.
`OnCalendar`	Defines the timer as a specific date/time value.
`OnStartupSec`	Defines the timer relative to when the systemd program started.
`OnUnitActiveSec`	Defines the timer relative to when the timer unit was last activated.

Directive	Description
`OnUnitInactiveSec`	Defines the timer relative to when the timer unit was last deactivated.
`Persistent`	When set, the time the timer unit was last triggered is stored on disk.
`RandomizedDelaySec`	Delays the timer activation by a random amount of time.
`RemainAfterElapse`	When set, the expired timer unit remains loaded, allowing you to query its status using `systemctl`.
`Unit`	Defines the unit file to start when the timer elapses.
`WakeSystem`	When set, the timer unit will cause the system to resume from being suspended.

As you can see from Table 6.13, timer units provide several options for how to set the timer. This allows you to choose exactly when a program should start on the system.

Summary

Managing your server's final system initialization phase is the job of the initialization daemon. This daemon must determine what services to start from the information you provide within the appropriate configuration files. In addition, the daemon can manage services while the system is running.

The classic system initialization daemon, SysV init, is still around, though typically only on older distributions. The popular and modern systemd is heavily used among current Linux distributions. It not only allows faster server boot times, it offers additional services as well, such as automounting filesystems. Often system administrators find themselves migrating from SysV init systems to systemd servers, and thus it is important to understand both system initialization methods.

Exam Essentials

Describe the init program. Either the `init` program or systemd is the parent process for every service on a Linux system. It typically has a PID of 1. The program is located in the `/etc/`, the `/bin/`, or the `/sbin/` directory. On systemd servers, this program is a symbolic link to `/usr/lib/systemd/systemd`.

Summarize systemd unit concepts. A systemd unit defines a service, a group of services, or an action, and there are currently 12 different systemd unit types. To view load units, use the `systemctl list-units` command. The four systemd units to focus on are service, target, mount, and automount.

Explain systemd service units and their files. Service units control how services are started, stopped, and managed. Their unit files contain configuration information via directives in one of the three primary unit file sections: `[Unit]`, `[Service]`, and `[Install]`. Directives, such as `After` and `Before`, configure when a service will be started. While the `[Unit]` and `[Install]` file sections are common to all unit files, the `[Service]` section and its directives are unique to services. Unit files may exist in one of three directory locations, and their location is important because if multiple files exist for a particular unit, one takes precedence over the other depending on its whereabouts.

Explain systemd target units and their files. Target units are responsible for starting groups of services. At system initialization, the `default.target` unit ensures that all required and desired services are launched. It is set up as a symbolic link to another target unit file. The primary target units used for system initialization are `graphical.target`, `multi-user.target`, and `runlevel`*n*`.target`, where *n* = 1–5 for the desired SysV init runlevel experience. There are additional target units, which handle system power off, halt, and reboot as well as emergency and rescue modes. The target type unit files are similar to service unit files, but they typically contain fewer directives.

Demonstrate how to manage systemd systems via commands. The `systemctl` utility contains many commands that allow you to manage and control systemd units. You can jump between targets using the `systemctl isolate` command. You can set particular services to start at system boot time via the `systemctl enable` command and vice versa via the `systemctl disable` command. Additional commands allow you to start, stop, restart, and reload units as well as reload their unit files via the `systemctl daemon-reload` command. Helpful commands such as `systemctl is-system-running` and `systemctl get-default` aid you in assessing your current systemd system. You can employ the `systemd-analyze` series of commands to evaluate your server's initialization process and find ways to improve it.

Summarize SysV init concepts. The classic SysV init method consists of the `/etc/inittab` file, which sets the default runlevel via the `initdefault` record. Runlevels determine what services are started, and the default runlevel determines what services are started at system initialization. The `rc` script starts and stops services depending on what runlevel is chosen. It executes the scripts in the appropriate runlevel directory and passes the appropriate `stop` or `start` parameter. The scripts located in the various runlevel directories are symbolic links to the files within the `/etc/init.d/` directory.

Demonstrate how to manage SysV init systems via commands. You can determine a SysV init system's previous and current runlevel via the `runlevel` command. Runlevels can be jumped into via the `init` or `telinit` command. Services can have their status checked; have their configuration files be reloaded; or be stopped, started, or restarted with the `status` command. You can view all currently loaded services on a SysV init system by using the `service --status-all` command. Services are enabled or disabled through either the `chkconfig` or the `update-rc.d` command, depending on your distribution.

Describe systemd mount and automount unit files. If your server employs systemd, besides managing system initialization, it can also persistently attach filesystems. These filesystems can be mounted or automounted via their associated unit files. Mount and automount unit filenames are based on the filesystem mount point but use the `.mount` or `.automount` filename extension, respectively. Their unit file contents have three sections, similar to service unit files, except the mount unit file's middle section is `[Mount]`, whereas the automount unit file's middle section is `[Automount]`. Each unit file has its own special directives that designate what partition is supposed to be mounted at the mount point and other items such as, for automount units, how long a filesystem must be idle before it can be unmounted.

Review Questions

1. The `init` program may be located in which of the following directories? (Choose all that apply.)
   - **A.** `/etc/rc.d/`
   - **B.** `/etc/`
   - **C.** `/sbin/`
   - **D.** `/usr/lib/systemd/`
   - **E.** `/bin/`
2. Which of the following is true concerning systemd service units? (Choose all that apply.)
   - **A.** Services can be started at system boot time.
   - **B.** Services can be started in parallel.
   - **C.** A service can be started based on a timer.
   - **D.** A service can be started after all other services are started.
   - **E.** A service can be prevented from starting at system boot time.
3. Which of the following is not a systemd target unit?
   - **A.** `runlevel7.target`
   - **B.** `emergency.target`
   - **C.** `graphical.target`
   - **D.** `multi-user.target`
   - **E.** `rescue.target`
4. You need to modify a systemd service unit configuration. Where should the modified file be located?
   - **A.** `/etc/system/systemd/`
   - **B.** `/usr/lib/system/systemd/`
   - **C.** `/etc/systemd/system/`
   - **D.** `/usr/lib/systemd/system/`
   - **E.** `/run/system/systemd/`
5. On your server, you need Service-B to start immediately before Service-A. Within the systemd Service-A unit configuration file, what directive should you check and potentially modify?
   - **A.** `Conflicts`
   - **B.** `Wants`
   - **C.** `Requires`
   - **D.** `Before`
   - **E.** `After`

6. For setting environment parameters within a unit configuration file, which directives should you potentially employ? (Choose all that apply.)

A. `Type`

B. `Environment`

C. `EnvironmentParam`

D. `EnvironmentFile`

E. `PATH`

7. You attempt to jump to a systemd target using the `systemctl isolate` command, but it will not work. You decide to look at the target unit file. What might you see there that is causing this problem?

A. `static`

B. `AllowIsolate=yes`

C. `Type=oneshot`

D. `AllowIsolate=no`

E. `disabled`

8. You have modified an OpenSSH service's configuration file, `/etc/ssh/ssh_config`. The service is already running. What is the best command to use with `systemctl` to make this modified file take immediate effect?

A. `reload`

B. `daemon-reload`

C. `restart`

D. `mask`

E. `unmask`

9. Your system uses systemd and has a service currently set to not start at system boot. You want to change this behavior and have it start. What `systemctl` command should you employ for this service?

A. `restart`

B. `start`

C. `isolate`

D. `disable`

E. `enable`

10. You need to change the system's default target. What `systemctl` command should you use to accomplish this task?

A. `get-default`

B. `set-default`

C. `isolate`

D. `is-enabled`

E. `is-active`

11. Your systemd system is taking a long time to boot and you need to reduce the boot time. Which `systemd-analyze` command is the best to start narrowing down which units need to be investigated first?

- **A.** `time`
- **B.** `dump`
- **C.** `failure`
- **D.** `blame`
- **E.** `verify`

12. Your older Debian-based Linux distribution system uses SysV init. It will soon be upgraded to a Debian-based distro that uses systemd. To start some analysis, you enter the `runlevel` command. Which of the following are results you may see? (Choose all that apply.)

- **A.** `N 5`
- **B.** `3 5`
- **C.** `N 2`
- **D.** `2 3`
- **E.** `1 2`

13. You've recently become the system administrator for an older Linux server, which still uses SysV init. You determine that its default runlevel is 3. What file did you find that information in?

- **A.** `/etc/inittab`
- **B.** `/etc/rc.d`
- **C.** `/etc/init.d/rc`
- **D.** `/etc/rc.d/rc`
- **E.** `/etc/rc.local`

14. Which directory on an old SysV init system stores the service startup scripts?

- **A.** `/usr/lib/systemd/system/`
- **B.** `/etc/rc.d/rc`*`n`*`.d/`
- **C.** `/etc/init.d/`
- **D.** `/etc/systemd/system/`
- **E.** `/run/systemd/system/`

15. You are managing a SysV init system and need to perform some emergency maintenance at runlevel 1. To do this, you need to jump runlevels. What command could you employ? (Choose all that apply.)

- **A.** `telinit S`
- **B.** `telinit 1`
- **C.** `init one`
- **D.** `init s`
- **E.** `init 1`

16. A customer has complained that a service on your SysV init system is not working. Which of the following commands is the best command to use to check the service?
    - **A.** `service start`
    - **B.** `service status`
    - **C.** `service --status-all`
    - **D.** `service stop`
    - **E.** `service reload`

17. You need to enable the DHCP service on your Red Hat–based SysV init system for runlevels 3 and 5. Which of the following commands should you use?
    - **A.** `service enable dhcp 3,5`
    - **B.** `chkconfig --levels 3,5 dhcp on`
    - **C.** `chkconfig --levels 35 on dhcp`
    - **D.** `chkconfig --levels 35 dhcp on`
    - **E.** `service enable dhcp 35`

18. You need to enable the DHCP service on your Debian-based SysV init system for the default runlevels. Which of the following commands should you use?
    - **A.** `update-rc.d dhcp default`
    - **B.** `chkconfig --default dhcp on`
    - **C.** `update-rc.d default dhcp`
    - **D.** `update-rc.d defaults dhcp`
    - **E.** `update-rc.d dhcp defaults`

19. Which of the following would be the appropriate base name for a mount unit file that mounts a filesystem at the `/var/log/` mount point?
    - **A.** `/var/log.mount`
    - **B.** `/var/log.unit`
    - **C.** `var-log.mount`
    - **D.** `var-log.unit`
    - **E.** `var/log.mount`

20. You are managing a systemd system and need to create an automount unit file. Which of the following directives should you review to possibly include in this file's `[Automount]` section? (Choose all that apply.)
    - **A.** `Where`
    - **B.** `Options`
    - **C.** `DirectoryMode`
    - **D.** `TimeOutIdleSec`
    - **E.** `What`