CHAPTER

15

Starting and Stopping Services

IN THIS CHAPTER

Understanding the various Linux init daemons

Auditing Linux init daemon–controlled services

Stopping and starting services

Changing the Linux server's default runlevel

Removing services

A *service*, also called a daemon, is a running program or a process that provides a particular function. Your Linux server's primary job is to offer services. As a Linux server administrator, one of your primary jobs is managing these services.

The `cupsd` daemon is an example of a service your Linux server can provide. This service is responsible for printing and is covered in Chapter 16, "Configuring a Print Server."

How do you start the `cupsd` daemon? How do you stop it? How do you get it to be started at system boot time? What if you have a special program you want to start on your server? How do you get it to start at boot time? All these questions and more will be answered in this chapter. Soon, you will have an excellent grasp on managing these services.

Understanding the Linux init Daemon

In order to understand service management, you need to understand the `init` daemon. The `init` daemon can be thought of as the "mother of all processes." This daemon is the first process to be started by the kernel on the Linux server. The Linux kernel has a process ID (PID) of 0. Thus, the `init` daemon has a parent process ID (PPID) of 0, and a PID of 1. Once started, `init` is responsible for spawning (launching) processes configured to be started at the server's boot time, such as the login shell (`getty` or `mingetty` process). It is also responsible for managing services. That's a lot of responsibility for one daemon!

The Linux `init` daemon was based upon the Unix System V `init` daemon. Thus, it is called the `SysVinit` daemon. However, it was not the only classic `init` daemon. The `init` daemon is not part of the Linux kernel. Therefore, it can come in different flavors, and Linux distributions can choose which flavor to use. Another classic `init` daemon was based on Berkeley Unix, also called BSD. Therefore, the two original Linux `init` daemons were `BSD init` and `SysVinit`.

The classic `init` daemons worked without problems for many years. However, these daemons were created to work within a static environment. As new hardware, such as USB devices, came along, the classic `init` daemons had trouble dealing with these and other hot-plug devices. Computer hardware had changed from static to event-based. New `init` daemons were needed to deal with these fluid environments.

In addition, as new services came along, the classic `init` daemons had to deal with starting more and more services. Thus, the entire system initialization process was less efficient and ultimately slower.

The modern `init` daemons have attempted to solve the problems of inefficient system boots and non-static environments. Two of these `init` daemons are `Upstart` and `systemd`. Many Linux distributions have made the move to the newer `init` daemons while maintaining backward compatibility to the classic `SysVinit` or `BSD init` daemons.

`Upstart`, available at `http://upstart.ubuntu.com`, was originally developed by Canonical, the parent of the Ubuntu distribution. However, many other distributions have adopted it, including:

- RHEL version 6
- Fedora versions 9 through 14
- Ubuntu versions 6–10 and up
- openSUSE versions 11.3 and up

A new daemon, `systemd`, available at `http://fedoraproject.org/wiki/Systemd`, was written primarily by Lennart Poettering, a Red Hat developer. It is currently used by Fedora versions 15 and up.

In order to properly manage your services, you need to know which `init` daemon your server has. Figuring that out can be a little tricky. Look through the following to help determine your Linux server's `init` daemon.

- Do your Linux distribution and version appear in the preceding list of `Upstart` adopters? Then your Linux `init` daemon is the `Upstart init` daemon.
- Is your Linux distribution Fedora version 15 or above? Then your Linux `init` daemon is the `systemd init` daemon.
- Try searching your Linux distribution's `init` daemon for clues, using the `strings` and the `grep` commands. The following code example shows the `init` daemon on a Linux Mint distribution being searched for `systemd` and `Upstart` `init` daemon references. The search for `systemd` yields nothing. However, the

search for `Upstart` produces results. Thus, you can see that this Linux Mint distribution uses the `Upstart init` daemon.

```
$ sudo strings /sbin/init | grep -i systemd
$
$ sudo strings /sbin/init | grep -i upstart
upstart-devel@lists.ubuntu.com
UPSTART_CONFDIR
UPSTART_NO_SESSIONS
…
```

On a Fedora server, the search for `Upstart` yields nothing. However, you can see the search for `systemd` yields the existence of the `systemd` daemon.

```
# strings /sbin/init | grep -i upstart
#
# strings /sbin/init | grep -i systemd
systemd.unit=
systemd.log_target=
systemd.log_level=
...
```

TIP

If you do not have the `strings` command on your Linux system, you can install it via the `binutils` package. On RHEL and Fedora, use the command **`yum install binutils`**. On Ubuntu, use the command **`sudo apt-get install binutils`**.

- If you still cannot tell what `init` daemon your server has, try looking on the `init` Wikipedia page (`http://wikipedia.org/wiki/Init`) under "Replacements for `init`?"
- Have you been unable to find any information using the preceding suggestions? Then, most likely, your distribution is still using the classic `SysVinit` or `BSD init` daemon.

Keep in mind that some Linux distributions have not moved to the newer daemons. Most of those that have, do maintain backward compatibility with the `SysVinit` and `BSD init` daemons.

Understanding the classic init daemons

The classic `init` daemons, `SysVinit` and `BSD init`, are worth understanding, even if your Linux server has a different `init` daemon. Not only is backward compatibility to the classics often used in the newer `init` daemons, but many are based upon them. Understanding the classic `init` daemons will help you to understand the modern `init` daemons.

15

The classic `SysVinit` and `BSD init` daemons operate in a very similar fashion. Although in the beginning they may have been rather different, over time, very few significant differences remained. For example, the older `BSD init` daemon would obtain configuration information from the `/etc/ttytab` file. Now, like the `SysVinit` daemon, the `BSD init` daemon's configuration information is taken at boot time from the `/etc/inittab` file. The following is a classic `SysVinit /etc/inittab` file:

```
# cat /etc/inittab
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
```

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The `/etc/inittab` file tells the `init` daemon which `runlevel` is the default `runlevel`. A `runlevel` is a categorization number that determines what services are started and what services are stopped. In the preceding example, a default `runlevel` of 5 is set with the code `id:5:initdefault:`. Table 15.1 shows the standard seven Linux runlevels.

TABLE 15.1 Standard Linux Runlevels

Runlevel #	Name	Description
0	Halt	All services are shut down and the server is stopped.
1 or S	Single User Mode	The root account is automatically logged in to the server. Other users cannot log in to the server. Only the command line interface is available. Network services are not started.
2	Multiuser Mode	Users can log in to the server, but only the command line interface is available. Network services are not started.
3	Extended Multiuser Mode	Users can log in to the server, but only the command line interface is available. Network services are started.
4	User Defined	Users can customize this `runlevel`.
5	Graphical Mode	Users can log in to the server. Command line and graphical interfaces are available. Network services are started.
6	Reboot	The server is rebooted.

Linux distributions can differ slightly on the definition of each `runlevel` as well as which runlevels are offered. The Ubuntu distribution, for example, offers runlevels 0–6, but `runlevels` 2–5 start the same services as standard `runlevel` 5 listed in Table 15.1.

CAUTION

The only runlevels that should be used in the `/etc/inittab` file are `2` through `5`. The other runlevels could cause problems. For example, if you put `runlevel 6` in the `/etc/inittab` file as the default, when the server reboots, it would go into a loop and continue to reboot over and over again.

15

Notice that it doesn't make sense to include some of the `runlevel` categories described in Table 15.1 in the `/etc/inittab` file. There is no reason why you would want your default `runlevel` to be `0`, which would halt the server every time it had just started.

The runlevels are not only used as a default `runlevel` in the `/etc/inittab` file. They can also be called directly using the `init` daemon itself. Thus, if you want to immediately halt your server, you type `init 0` at the command line:

```
# init 0
...
System going down for system halt NOW!
```

The `init` command will accept any of the `runlevel` numbers in Table 15.1, allowing you to quickly switch your server from one `runlevel` category to another. For example, if you need to perform troubleshooting, which requires the graphical interface to be down, you can type **`init 3`** at the command line:

```
# init 3
INIT: Sending processes the TERM signal
starting irqbalance:                    [ OK ]
Starting setroubleshootd:
Starting fuse:  Fuse filesystem already available.
...
Starting console mouse services:        [ OK ]
```

To see your Linux server's current `runlevel`, simply type in the command **`runlevel`**. The first item displayed will be the server's previous `runlevel`, which in the following example is `5`. The second item displayed shows the server's current `runlevel`, which in this example is `3`.

```
$ runlevel
5 3
```

In addition to the `init` command, you can also use the `telinit` command, which is functionally the same. In the example that follows, the `telinit` command is used to reboot the server by taking it to `runlevel 6`:

```
# telinit 6
INIT: Sending processes the TERM signal
Shutting down smartd:                          [ OK ]
Shutting down Avahi daemon:                    [ OK ]
Stopping dhcdbd:                               [ OK ]
Stopping HAL daemon:                           [ OK ]
...
Starting killall:
Sending all processes the TERM signal...       [ OK ]
Sending all processes the KILL signal...       [ OK ]
```

```
...
Unmounting filesystems                          [ OK ]
Please stand by while rebooting the system
...
```

On a freshly booted Linux server, the current `runlevel` number should be the same as the default `runlevel` number in the `/etc/inittab` file. However, notice that the previous `runlevel` in the example that follows is `N`. The `N` stands for “Nonexistent” and indicates the server has been freshly booted.

```
$ runlevel
N 5
```

How does the server know which services to stop and which ones to start when a particular `runlevel` is chosen? When a `runlevel` is chosen, the scripts located in the `/etc/rc.d/rc#.d` directory (where # is the chosen `runlevel`) are run. These scripts are run whether the `runlevel` is chosen via a server boot and the `/etc/inittab` `runlevel` setting, or when the `init` or `telinit` command is used. For example, if `runlevel 5` is chosen, then all the scripts in the `/etc/rc.d/rc5.d` directory are run.

```
# ls /etc/rc.d/rc5.d
K01smolt                    K88wpa_supplicant   S22messagebus
K02avahi-dnsconfd           K89dund             S25bluetooth
K02NetworkManager           K89netplugd         S25fuse
K02NetworkManagerDispatcher K89pand             S25netfs
K05saslauthd                K89rdisc            S25pcscd
K10dc_server                K91capi             S26hidd
K10psacct                   S00microcode_ctl    S26udev-post
K12dc_client                S04readahead_early  S28autofs
K15gpm                      S05kudzu            S50hplip
K15httpd                    S06cpuspeed         S55cups
K20nfs                      S08ip6tables        S55sshd
K24irda                     S08iptables         S80sendmail
K25squid                    S09isdn             S90ConsoleKit
K30spamassassin             S10network          S90crond
K35vncserver                S11auditd           S90xfs
K50netconsole               S12restorecond      S95anacron
K50tux                      S12syslog           S95atd
K69rpcsvcgssd               S13irqbalance       S96readahead_later
K73winbind                  S13mcstrans         S97dhcdbd
K73ypbind                   S13rpcbind          S97yum-updatesd
K74nscd                     S13setroubleshoot   S98avahi-daemon
K74ntpd                     S14nfslock          S98haldaemon
K84btseed                   S15mdmonitor        S99firstboot
K84bttrack                  S18rpcidmapd        S99local
K87multipathd               S19rpcgssd          S99smartd
```

Notice that some of the scripts within the /etc/rc.d/rc5.d directory start with a K and some start with an S. The K refers to a script that will kill (stop) a process. The S refers to a script that will start a process. Also, each K and S script has a number before the name of the service or daemon they control. This allows the services to be stopped or started in a particular controlled order. You would not want your Linux server's network services to be started before the network itself was started.

An /etc/rc.d/rc#.d directory exists for all of the standard Linux runlevels. Each one contains scripts to start and stop services for its particular runlevel.

```
# ls -d /etc/rc.d/rc?.d
/etc/rc.d/rc0.d  /etc/rc.d/rc2.d  /etc/rc.d/rc4.d  /etc/rc.d/rc6.d
/etc/rc.d/rc1.d  /etc/rc.d/rc3.d  /etc/rc.d/rc5.d
```

Actually, the files in the /etc/rc.d/rc#.d directories are not scripts, but instead symbolic links to scripts in the /etc/rc.d/init.d directory. Thus, there is no need to have multiple copies of particular scripts.

```
# ls -l /etc/rc.d/rc5.d/K15httpd
lrwxrwxrwx 1 root root 15 2011-10-27 19:59
 /etc/rc.d/rc5.d/K15httpd -> ../init.d/httpd
#
# ls /etc/rc.d/init.d
anacron             functions      multipathd                 rpcidmapd
atd                 fuse           netconsole                 rpcsvcgssd
auditd              gpm            netfs                      saslauthd
autofs              haldaemon      netplugd                   sendmail
avahi-daemon        halt           network                    setroubleshoot
avahi-dnsconfd      hidd           NetworkManager             single
bluetooth           hplip          NetworkManagerDispatcher   smartd
btseed              hsqldb         nfs                        smolt
bttrack             httpd          nfslock                    spamassassin
capi                ip6tables      nscd                       squid
ConsoleKit          iptables       ntpd                       sshd
cpuspeed            irda           pand                       syslog
crond               irqbalance     pcscd                      tux
cups                isdn           psacct                     udev-post
cups-config-daemon  killall        rdisc                      vncserver
dc_client           kudzu          readahead_early            winbind
dc_server           mcstrans       readahead_later            wpa_supplicant
dhcdbd              mdmonitor      restorecond                xfs
dund                messagebus     rpcbind                    ypbind
firstboot           microcode_ctl  rpcgssd                    yum-updatesd
```

Notice that each service has a single script in /etc/rc.d/init.d. There aren't separate scripts for stopping and starting a service. These scripts will stop or start a service depending upon what parameter is passed to them by the init daemon.

NOTE

Depending upon your distribution and your `init` daemon, a service's stop and start scripts may be stored in a different location than the `/etc/rc.d/init.d` directory. The three possible locations are:

- `/etc/rc.d/init.d` **— A directory used by the `SysVinit` daemon.**
- `/etc/rc.d` **— The directory used by the `BSD init` daemon**
- `/etc/init.d` **— A directory also used by the `SysVinit` daemon, typically on Debian-based distributions, such as Ubuntu**

Each script in `/etc/rc.d/init.d` takes care of all that is needed for starting or stopping a particular service on the server. The following is a partial example of the `httpd` script on a Linux system that uses the `SysVinit` daemon. It contains a case statement for handling the parameter (`$1`) that was passed to it, such as `start`, `stop`, `status`, and so on.

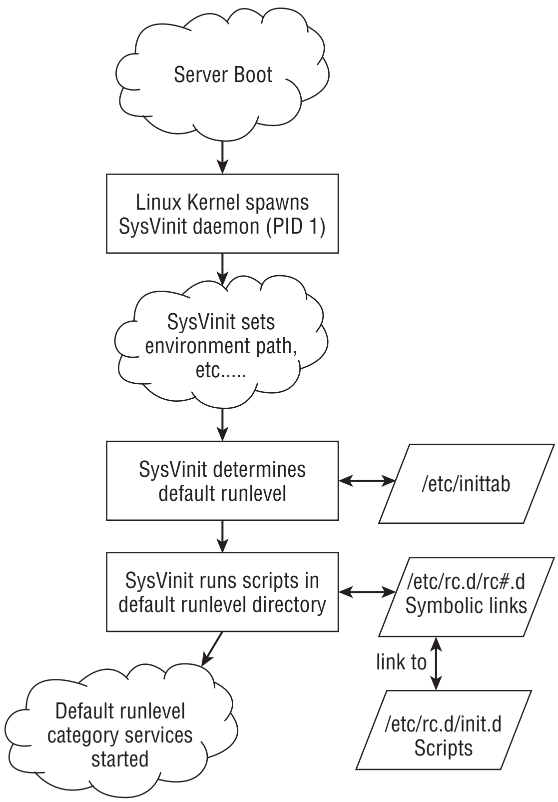
```
# cat /etc/rc.d/init.d/httpd
#!/bin/bash
#
# httpd        Startup script for the Apache HTTP Server
#
# chkconfig: - 85 15
# description: Apache is a World Wide Web server.
#              It is used to serve \
#              HTML files and CGI.
# processname: httpd
# config: /etc/httpd/conf/httpd.conf
# config: /etc/sysconfig/httpd
# pidfile: /var/run/httpd.pid

# Source function library.
. /etc/rc.d/init.d/functions
...
# See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status $httpd
        RETVAL=$?
        ;;
...
esac

exit $RETVAL
```

15

Once the desired `runlevel` scripts in the `/etc/init.d` directory are executed, then the `SysVinit` daemon's process spawning is complete. This entire course is a well-organized, step-by-step approach for starting up a Linux server with the appropriate services running. Figure 15.1 reviews the `SysVinit` daemon's role in spawning the correct processes at system startup.

FIGURE 15.1

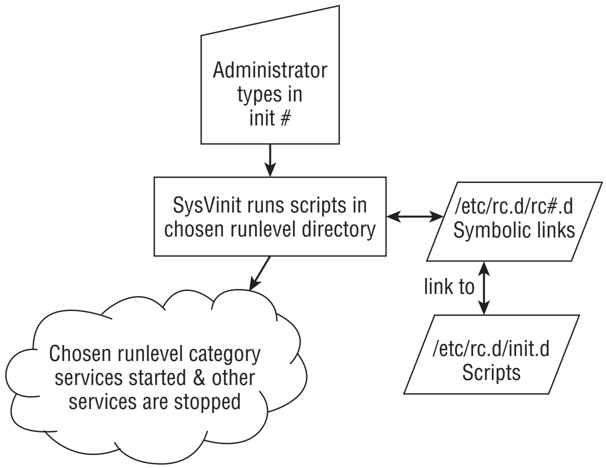
Classic SysVinit daemon course of action at server boot



The course is very similar when switching from one `runlevel` category to another. Figure 15.2 reviews the `SysVinit` daemon's roles in stopping and starting the correct processes during a `runlevel` switch.

FIGURE 15.2

Classic SysVinit daemon course of action at runlevel change



Now that you have a grasp of the classic `init` daemons, it's time to move on to the more modern `Upstart init` daemon.

Understanding the Upstart init daemon

As mentioned earlier, many Linux distributions have moved from the classic `init` daemons to the more modern `Upstart init` daemon. Included in that distribution list are the RHEL and Ubuntu distributions.

Learning Upstart init daemon basics

The primary difference between the classics and `Upstart` is the handling of stopping and starting services. The `SysVinit` daemon was created to operate in a static environment. The `Upstart init` daemon was created to operate in a flexible and ever-changing environment. With `SysVinit`, services are stopped and started based upon runlevels. The `Upstart init` daemon is not concerned with runlevels but with system events. Events are what determine when services are stopped and/or started. An *event* is a Linux server occurrence that triggers a needed system state change, which is communicated to the `Upstart init` daemon. The following are some examples of system events:

- The server boots up.
- The `init` command is used.
- A USB device is plugged into the server.

While the classic `init` daemons could handle the first two event examples, they could not deal well with the third.

15

`Upstart` handles services through defined jobs. An `Upstart` *job* can be either a task or a service. A *task* performs a limited duty, completes its work, and then returns to a waiting state. A *service*, on the other hand, is a long running program that never finishes its work or self-terminates, but instead stays in a running state. A daemon is an example of an `Upstart` service job.

The example that follows shows several `Upstart` jobs that include both task and service jobs. The task jobs are in a `stop/waiting` state, such as the task `rc`. The service jobs are in a `start/running` state, such as the `cups` daemon.

```
$ initctl list
avahi-daemon start/running, process 456
mountall-net stop/waiting
rc stop/waiting
rsyslog start/running, process 411
...
ssh start/running, process 405
udev-fallback-graphics stop/waiting
control-alt-delete stop/waiting
hwclock stop/waiting
mounted-proc stop/waiting
network-manager start/running, process 458
...
rc-sysinit stop/waiting
cups start/running, process 1066
...
tty6 start/running, process 833
ureadahead stop/waiting
```

These various jobs are defined via a jobs definition file. All the job definition files are located in the `/etc/init` directory as shown here:

```
$ ls /etc/init
acpid.conf                networking.conf
alsa-restore.conf         network-interface.conf
alsa-store.conf           network-interface-security.conf
anacron.conf              network-manager.conf
control-alt-delete.conf   procps.conf
cron.conf                 rc.conf
cups.conf                 rcS.conf
dbus.conf                 rc-sysinit.conf
dmesg.conf                rsyslog.conf
failsafe.conf             setvtrgb.conf
friendly-recovery.conf    ssh.conf
hostname.conf             tty1.conf
hwclock.conf              tty2.conf
hwclock-save.conf         tty3.conf
```

```
irqbalance.conf          tty4.conf
lightdm.conf             tty5.conf
...
```

The `Upstart init` daemon depends upon events to trigger certain services to start, stop, restart, and so on. Events are either communicated to the `Upstart init` daemon or they are created by the `Upstart` daemon. This is called an *emitted event*. The actions taken when an event is emitted are dependent upon the settings in a job's configuration file. Consider the following `Network Manager` daemon's configuration file:

```
$ cat /etc/init/network-manager.conf
# network-manager - network connection manager
#
# The Network Manager daemon manages the system's network connections,
# automatically switching between the best available.

description     "network connection manager"

start on (local-filesystems          and started dbus)
stop on stopping dbus

expect fork
respawn

exec NetworkManager
$
```

From the example, you can see that there are two events that must take place in order to trigger the `Upstart init` daemon to start the `Network Manager` daemon:

- **The** `local-filesystems` **event** — The `Upstart init` daemon will emit this event when all the local filesystems in the `/etc/fstab` configuration file have been mounted.
- **The** `dbus daemon started` **event** — The `Upstart init` daemon will emit this `started` event when the `dbus` daemon has reached the `start/running` state.

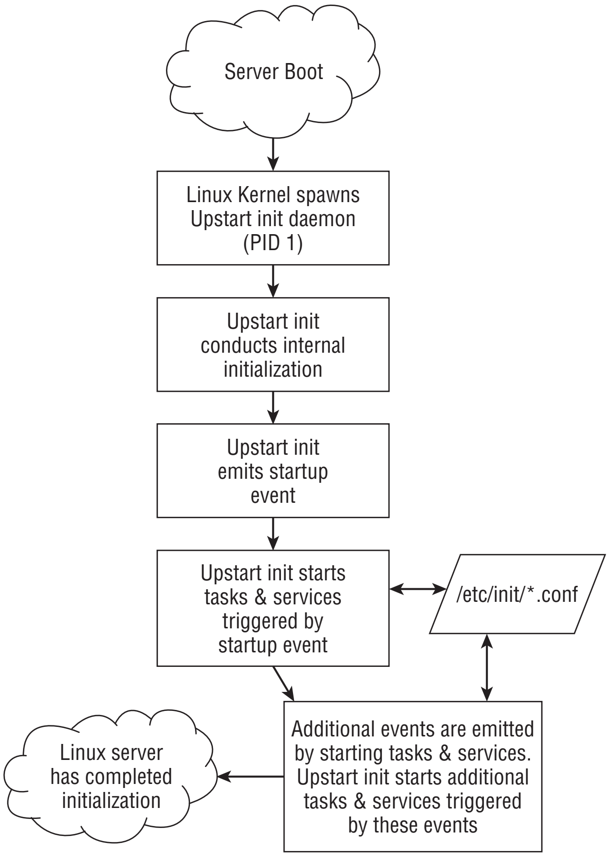
Thus, when these two events occur, the `Upstart init` daemon is informed and then will start the `NetworkManager` daemon.

Because the `Upstart init` daemon can handle these events and tracks the status (state) of processes, it is often referred to as a "state machine." The `Upstart init` daemon is also referred to as an "event engine" because it emits events itself.

The `Upstart init` daemon is a flexible, organized, and clever approach for dealing with server events. Figure 15.3 shows the `Upstart init` daemon's course of action in spawning the correct processes at system startup time.

FIGURE 15.3

Upstart init daemon course of action at server boot



Learning Upstart's backward compatibility to SysVinit

`Upstart` provides backward compatibility to the `SysVinit` daemon. This has allowed the Linux distributions time to slowly migrate to `Upstart`.

The `/etc/inittab` file is still on some distributions. RHEL and the Fedora distributions still using `Upstart` use `/etc/inittab` to boot to the default `runlevel` listed. The Ubuntu distribution no longer has the file. The example of the `/etc/inittab` file that follows comes from a server running a version of Fedora, which uses the `Upstart init` daemon.

```
$ cat /etc/inittab
# inittab is only used by upstart for the default runlevel.
#
```

```
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
...
#
id:5:initdefault:
```

As you can see from the comment lines in the `/etc/inittab` file, the only thing this file is used for on Linux distributions that maintain it is to change the default `runlevel` at server boot time.

TIP

To change the default `runlevel` on an Ubuntu distribution that uses `Upstart`, edit `/etc/init/rc-sysinit.conf` and change the line `env DEFAULT_RUNLEVEL=#` where `#` is 2 to 5. However, remember that the runlevels 2–5 on Ubuntu are equivalent to `SysVinit runlevel 5`. Therefore, this activity is rather pointless.

System initialization compatibility to `SysVinit` is maintained on some distributions, such as Ubuntu, via the `/etc/init/rc-sysinit.conf` configuration file. This is one of the configuration files used at system boot, as shown previously in Figure 15.3. In the example that follows, you can see that `Upstart` checks for a `/etc/inittab` file and also runs any scripts that may still be in the `/etc/init.d/rcS` directory:

```
$ cat /etc/init/rc-sysinit.conf
# rc-sysinit - System V initialisation compatibility
#
# This task runs the old System V-style system initialisation scripts,
# and enters the default runlevel when finished.
...
start on (filesystem and static-network-up) or failsafe-boot
stop on runlevel

# Default runlevel, this may be overriden on the kernel command-line
# or by faking an old /etc/inittab entry
env DEFAULT_RUNLEVEL=2

emits runlevel
...
task

script
    # Check for default runlevel in /etc/inittab
    if [ -r /etc/inittab ]
    then
      eval "$(sed -nre 's/^[^#][^:]*:([0-6sS]):initdefault:
.*/DEFAULT_RUNLEVEL="\1";/p' /etc/inittab || true)"
    fi

    # Check kernel command-line for typical arguments
```

15

```
    for ARG in $(cat /proc/cmdline)
    do
      case "${ARG}" in
      -b|emergency)
          # Emergency shell
          [ -n "${FROM_SINGLE_USER_MODE}" ] || sulogin
          ;;
       [0123456sS])
          # Override runlevel
          DEFAULT_RUNLEVEL="${ARG}"
          ;;
      -s|single)
          # Single user mode
          [ -n "${FROM_SINGLE_USER_MODE}" ] || DEFAULT_RUNLEVEL=S
          ;;
      esac
    done

    # Run the system initialisation scripts
    [ -n "${FROM_SINGLE_USER_MODE}" ] || /etc/init.d/rcS

    # Switch into the default runlevel
    telinit "${DEFAULT_RUNLEVEL}"
end script
```

As you can see from the preceding example, the `runlevel` concept is maintained in the `Upstart init` daemon. In fact, there is even a `runlevel` signal that `Upstart` can emit.

```
# man -k "event signal"
control-alt-delete   (7) - event signalling console press of Control-Alt-Delete
keyboard-request     (7) - event signalling console press of Alt-UpArrow
power-status-changed (7) - event signalling change of power status
runlevel             (7) - event signalling change of system runlevel
started              (7) - event signalling that a job is running
starting             (7) - event signalling that a job is starting
startup              (7) - event signalling system startup
stopped              (7) - event signalling that a job has stopped
stopping             (7) - event signalling that a job is stopping
```

Switching to a different `runlevel` is still allowed through the `init` or `telinit` commands. Any `runlevel` event is handled by the `rc` task.

```
$ initctl status rc
rc stop/waiting
```

The `rc` task job's configuration file is shown next. When a `runlevel` event is emitted, the `rc` configuration file will call the `/etc/rc.d/rc` script. When called, the `/etc/rc.d/rc` script will run the scripts located in the `/etc/rc.d/rc#.d`, where # is the chosen `runlevel`. This provides `runlevel` backward compatibility to `SysVinit`.

```
$ cat /etc/init/rc.conf
# rc - System V runlevel compatibility
#
# This task runs the old sysv-rc runlevel scripts.  It
# is usually started by the telinit compatibility wrapper.

start on runlevel [0123456]

stop on runlevel [!$RUNLEVEL]

task

export RUNLEVEL
console output
exec /etc/rc.d/rc $RUNLEVEL
```

If you look back at the /etc/inittab in the classic SysVinit daemon section, you will notice that /etc/inittab also handled spawning the getty or mingetty processes. The Upstart init daemon handles this via the start-ttys task.

```
# initctl status start-ttys
start-ttys stop/waiting
```

The start-ttys task job's configuration file is shown next. When a runlevel event is emitted, the start-ttys configuration file will spawn the getty or mingetty process.

```
$ cat /etc/init/start-ttys.conf
#
# This service starts the configured number of gettys.

start on stopped rc RUNLEVEL=[2345]

env ACTIVE_CONSOLES=/dev/tty[1-6]
env X_TTY=/dev/tty1
task
script
   . /etc/sysconfig/init
   for tty in $(echo $ACTIVE_CONSOLES) ; do
     [ "$RUNLEVEL" = "5" -a "$tty" = "$X_TTY" ] && continue
     initctl start tty TTY=$tty
   done
end script
```

Although the Upstart init daemon provides backward compatibility to the classic SysVinit daemon, is a state-machine, and can handle ever-changing events on a server, it is not the only modern init daemon available for the Linux server. Another modern init daemon is systemd.

15

Understanding systemd init

The `systemd init` daemon is also called `systemd` or the `system` daemon. `systemd` can replace the `SysVinit` and the `Upstart init` daemons. This modern `init` daemon currently runs on Fedora 15 and above and is backward compatible with both `SysVinit` and `Upstart`. System initialization time is reduced by `systemd` because it starts fewer services and starts them in a parallel manner. In addition, `systemd` can handle a fluid environment because it supervises all the processes on the entire Linux server.

Learning systemd basics

With the `SysVinit` daemon, services are stopped and started based upon runlevels. The `systemd` is also concerned with runlevels, but they are called `target units`. Units are the focus of `systemd`. A *unit* is a group consisting of a name, type, and configuration file and is focused on a particular service or action. The eight `systemd units` types are:

- `automount`
- `device`
- `mount`
- `path`
- `service`
- `snapshot`
- `socket`
- `target`

The two primary `systemd` units you need to be concerned with for dealing with services are `service units` and `target units`. A *service unit* is for managing daemons on your Linux server. A *target unit* is simply a group of other units.

The example that follows shows several `systemd service units` and `target units`. The `service units` have familiar daemon names, such as `cups` and `sshd`. Note that each `service unit` name ends with `.service`. The `target units` shown have names like `sysinit`. (`sysinit` is used for starting up services at system initialization.) The `target unit` names end with `.target`.

```
# systemctl list-units | grep .service
...
cups.service             loaded active running CUPS Printing Service
dbus.service             loaded active running D-Bus System Message Bus
...
NetworkManager.service   loaded active running Network Manager
prefdm.service           loaded active running Display Manager
remount-rootfs.service   loaded active exited  Remount Root FS
rsyslog.service          loaded active running System Logging Service
...
```

```
sshd.service              loaded active running OpenSSH server daemon
system-s...yboard.service loaded active running System Setup Keyboard
systemd-logind.service    loaded active running Login Service
...
#
# systemctl list-units | grep .target
basic.target              loaded active active  Basic System
cryptsetup.target         loaded active active  Encrypted Volumes
getty.target              loaded active active  Login Prompts
graphical.target          loaded active active  Graphical Interface
local-fs-pre.target       loaded active active  Local File Systems (Pre)
local-fs.target           loaded active active  Local File Systems
multi-user.target         loaded active active  Multi-User
network.target            loaded active active  Network
remote-fs.target          loaded active active  Remote File Systems
sockets.target            loaded active active  Sockets
sound.target              loaded active active  Sound Card
swap.target               loaded active active  Swap
sysinit.target            loaded active active  System Initialization
syslog.target             loaded active active  Syslog
```

The Linux system unit configuration files are located in the `/lib/systemd/system` and `/etc/systemd/system` directories. You could use the `ls` command to look through those directories, but the preferred method is to use an option on the `systemctl` command as follows:

```
# systemctl list-unit-files --type=service
UNIT FILE                                    STATE
...
cups.service                                 enabled
...
dbus.service                                 static
...
NetworkManager.service                       enabled
...
poweroff.service                             static
...
sshd.service                                 enabled
sssd.service                                 disabled
...
134 unit files listed.
```

The unit configuration files shown in the preceding code are all associated with a `service unit`. Configuration files for `Target units` can be displayed via the following method.

```
# systemctl list-unit-files --type=target
UNIT FILE                 STATE
anaconda.target           static
basic.target              static
```

15

```
bluetooth.target           static
cryptsetup.target          static
ctrl-alt-del.target        disabled
default.target             enabled
...
shutdown.target            static
sigpwr.target              static
smartcard.target           static
sockets.target             static
sound.target               static
swap.target                static
sysinit.target             static
syslog.target              static
time-sync.target           static
umount.target              static
43 unit files listed.
```

Notice that both of the configuration units' file examples shown display units with a status of either `static`, `enabled`, or `disabled`. The `enabled` status means that the unit is currently enabled. The `disabled` status means that the unit is currently disabled. The next status, `static`, is slightly confusing. It stands for "statically enabled," and it means that the unit is enabled by default and cannot be disabled, even by root.

The `service unit` configuration files contain a lot of information, such as what other services must be started, when this service can be started, which environmental file to use, and so on. The following example shows the `sshd`'s unit configuration file:

```
# cat /lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
After=syslog.target network.target auditd.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

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

This basic `service unit` configuration file has the following options:

- `Description` — This is a free-form description (comment line) of the service.
- `After` — This setting configures ordering. In other words, it lists which units should be activated before this service is started.
- `Environment File` — The service's configuration file.
- `ExecStart` — The command used to start this service.

- `ExecReload` — The command used to reload this service.
- `WantedBy` — This identifies what `target unit` this service belongs to.

Notice that the `target unit`, `multi-user.target`, is used in the `sshd service unit` configuration file. The `sshd service unit` is wanted by the `multi-user.target`. In other words, when the `multi-user.target` unit is activated, the `sshd service unit` will be started.

You can view the various units a `target unit` will activate by using the following command:

```
# systemctl show --property "Wants" multi-user.target
Wants=multipathd.service avahi-daemon.service sshd-keygen.se
(END) q
```

Unfortunately, the `systemctl` command does not format the output for this well. It literally runs off the right edge of the screen so that you cannot see the full results. And you have to enter a `q` to return to the command prompt. To fix this problem, pipe the output through some formatting commands to produce a nice alphabetically sorted display, as shown in the example that follows.

```
# systemctl show --property "Wants" multi-user.target |
> fmt -10 | sed 's/Wants=//g' | sort
abrt-ccpp.service
abrtd.service
abrt-oops.service
abrt-vmcore.service
atd.service
auditd.service
avahi-daemon.service
crond.service
cups.path
dbus.service
fcoe.service
getty.target
irqbalance.service
iscsid.service
iscsi.service
livesys-late.service
livesys.service
lldpad.service
mcelog.service
mdmonitor.service
multipathd.service
netfs.service
NetworkManager.service
plymouth-quit.service
plymouth-quit-wait.service
remote-fs.target
```

15

```
rsyslog.service
sendmail.service
sm-client.service
sshd-keygen.service
sshd.service
systemd-ask-password-wall.path
systemd-logind.service
systemd-update-utmp-runlevel.service
systemd-user-sessions.service
#
```

This display shows all the services and various other units that will be activated (started), including the `sshd`, when the `multi-user.target` unit is activated. Remember that a `target unit` is simply a grouping of other units, as shown in the preceding example. Also notice that the units in this group are not all `service units`. There are `path units` and another `target unit` as well.

A `target unit` has both `Wants` and requirements, called `Requires`. A *Wants* means that all the units listed are triggered to activate (start). If they fail or cannot be started, no problem — the `target unit` continues on its merry way. The preceding example is a display of `Wants` only.

A `Requires` is much more stringent and potentially catastrophic than a `Wants`. A *Requires* means that all the units listed are triggered to activate (start). If they fail or cannot be started, the entire unit (group of units) is deactivated.

You can view the various units a `target unit` `Requires` (must activate or the unit will fail), using the command in the example that follows. Notice the `Requires` output is much shorter than the `Wants` for the `multi-user target`. Thus, no special formatting of the output is needed.

```
# systemctl show --property "Requires" multi-user.target
Requires=basic.target
```

The `target units` also have configuration files, as do the `service units`. The following example shows the contents of the `multi-user.target` configuration file.

```
# cat /lib/systemd/system/multi-user.target
#  This file is part of systemd.
#
...

[Unit]
Description=Multi-User
Requires=basic.target
Conflicts=rescue.service rescue.target
```

```
After=basic.target rescue.service rescue.target
AllowIsolate=yes

 [Install]
Alias=default.target
```

This basic `target unit` configuration file has the following options:

- `Description` — This is just a free-form description of the target.
- `Requires` — If this `multi-user.target` gets activated, then the listed `target unit` will also be activated. If the listed `target unit` is deactivated or fails, then `multi-user.target` will be deactivated. If there are no `After` and `Before` options, then both `multi-user.target` and listed `target unit` will activate simultaneously.
- `Conflicts` — This setting avoids conflicts in services. Starting `multi-user.target` will stop the listed targets and services, and vice-versa.
- `After` — This setting configures ordering. In other words, it determines which units should be activated before starting this service.
- `AllowIsolate` — This option is a Boolean setting of `yes` or `no`. If set to `yes`, then this `target unit`, `multi-user.target`, is activated along with its dependencies and all others are deactivated.
- `ExecStart` — This command starts the service.
- `ExecReload` — This command reloads the service.
- `Alias` — With this command, `systemd` will create a symbolic link from the `target unit` names listed to this unit, `multi-user.target`.

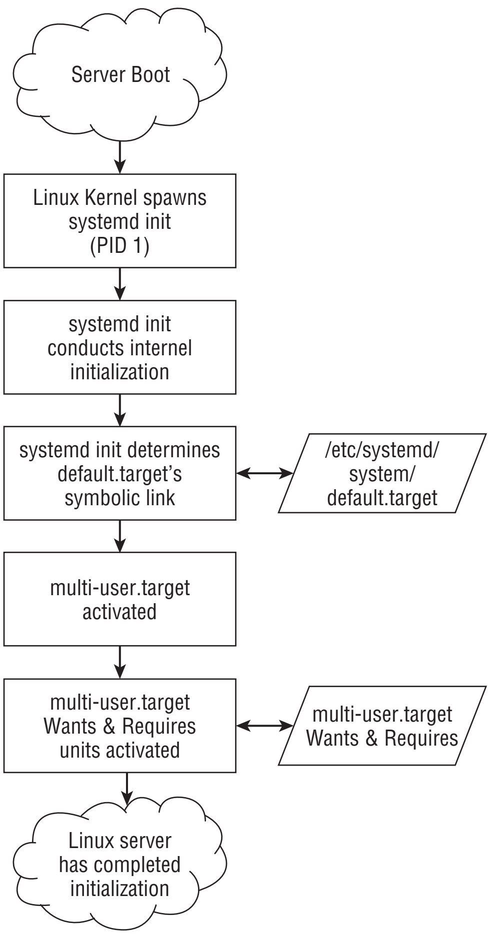
To get more information on these configuration files and their options, enter **`man systemd.service`**, **`man systemd.target`**, and **`man systemd.unit`** at the command line.

For the Linux server using `systemd init`, the boot process will be easier to follow, now that you understand `systemd target units`. At boot, `systemd` activates the `default.target` unit. This unit is aliased to either `multi-user.target` or `graphical.target`. Thus, depending upon the `alias` set, the services targeted by the `target unit` are started. Figure 15.4 shows the `systemd init`'s course of action in spawning the correct processes at system startup. If you are following along with the examples used in this section, the `default.target` is aliased to `multi-user.target`, as shown in Figure 15.4.

If you need more help on understanding `systemd init`, there is excellent documentation by its author at `http://0pointer.de/blog/projects/systemd-docs.html`. Also, you can enter **`man -k systemd`** at the command line to get a listing of the various `systemd` utilities' documentation in the man pages.

FIGURE 15.4

systemd init course of action at server boot



Learning systemd's backward compatibility to SysVinit

The `systemd init` has maintained backward compatibility to the `SysVinit` daemon. This allows Linux distributions time to slowly migrate to `systemd`.

While runlevels are not truly part of `systemd`, the `systemd` infrastructure has been created to provide compatibility with the concept of runlevels. There are seven `target units` configuration files specifically created for backward compatibility to `SysVinit`:

- `runlevel0.target`
- `runlevel1.target`

- runlevel2.target
- runlevel3.target
- runlevel4.target
- runlevel5.target
- runlevel6.target

As you probably have already figured out, there is a target unit configuration file for each of the seven classic SysVinit runlevels. These target unit configuration files are symbolically linked to target unit configuration files that most closely match the idea of the original runlevel. In the example that follows, the symbolic links are shown for runlevel target units. Notice that the runlevel target units for runlevel 2, 3, and 4 are all symbolically linked to multi-user.target. The multi-user.target unit is similar to the legacy Extended Multi-user Mode.

```
# ls -l /lib/systemd/system/runlevel*.target
lrwxrwxrwx. 1 root root 15 Mar 27 15:39
 /lib/systemd/system/runlevel0.target -> poweroff.target
lrwxrwxrwx. 1 root root 13 Mar 27 15:39
 /lib/systemd/system/runlevel1.target -> rescue.target
lrwxrwxrwx. 1 root root 17 Mar 27 15:39
 /lib/systemd/system/runlevel2.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Mar 27 15:39
 /lib/systemd/system/runlevel3.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Mar 27 15:39
 /lib/systemd/system/runlevel4.target -> multi-user.target
lrwxrwxrwx. 1 root root 16 Mar 27 15:39
 /lib/systemd/system/runlevel5.target -> graphical.target
lrwxrwxrwx. 1 root root 13 Mar 27 15:39
 /lib/systemd/system/runlevel6.target -> reboot.target
#
```

The /etc/inittab file still exists, but it only contains comments stating this configuration file is not used and gives some basic systemd information. The /etc/inittab file no longer has any true functional use. The following is an example of a /etc/inittab file on a Linux server that uses systemd.

```
# cat /etc/inittab
# inittab is no longer used when using systemd.
#
# ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# Ctrl-Alt-Delete is handled by
/etc/systemd/system/ctrl-alt-del.target
#
# systemd uses 'targets' instead of runlevels.
 By default, there are two main targets:
#
# multi-user.target: analogous to runlevel 3
```

15

```
# graphical.target: analogous to runlevel 5
#
# To set a default target, run:
#
# ln -s /lib/systemd/system/<target name>.target
 /etc/systemd/system/default.target
```

The /etc/inittab explains that if you want something similar to a classic 3 or 5 runlevel as your default runlevel, you will need to create a symbolic link from the default.target unit to the runlevel target unit of your choice. To check what default.target is currently symbolically linked to (or in legacy terms, to check the default runlevel), use the command shown here. You can see that on this Linux server, the default is to start up at legacy runlevel 3.

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Mar 13 17:27
 /etc/systemd/system/default.target ->
 /lib/systemd/system/runlevel3.target
```

The capability to switch runlevels using the init or telinit command is still available. When issued, either of the commands will be translated into a systemd target unit activation request. Therefore, typing **init 3** at the command line really issues the command systemctl isolate multi-user.target. Also, you can still use the runlevel command to determine the current legacy runlevel, but it is strongly discouraged.

The classic SysVinit /etc/inittab handled spawning the getty or mingetty processes. The systemd init handles this via the getty.target unit. The getty.target is activated by the multi-user.target unit. You can see how these two target units are linked by the following command:

```
# systemctl show --property "WantedBy" getty.target
WantedBy=multi-user.target
```

As of Fedora 16, the conversion of SysVinit core services to systemd is complete. However, to find out the current status of a particular service, visit the systemd compatibility web page at http://fedoraproject.org/wiki/User:Johannbg/QA/Systemd/compatability.

Now that you have a basic understanding of classic and modern init daemons, it's time to look at doing some practical server administrator actions that involve the init daemon.

Auditing Services

As a Linux administrator, you need to audit the services being offered on your server for documentation, security, and troubleshooting purposes. Typically, you will audit services for documentation purposes if you have "inherited" a server, although some sites do this on a regular basis. For security reasons, you will need to disable and remove any unused system services discovered through the audit process. Most importantly for troubleshooting purposes, you need to be able to quickly know what should and should not be running on your Linux server.

Of course, knowing which init daemon is being used by your Linux server is the first piece of information to obtain. How to determine this was covered in the "Understanding the Linux init Daemon" section of this chapter. The rest of this section is organized into subsections on the various init daemons.

Auditing the classic SysVinit daemon

To see all the services that are being offered by a Linux server using the classic SysVinit daemon, use the chkconfig command. The example that follows shows the services available on a classic SysVinit Linux server. Note that each runlevel (0–6) is shown for each service with a status of on or off. The status denotes whether a particular service is started (on) or not (off) for that runlevel.

```
# chkconfig --list
ConsoleKit      0:off  1:off  2:off  3:on   4:on   5:on   6:off
NetworkManager  0:off  1:off  2:off  3:off  4:off  5:off  6:off
...
crond           0:off  1:off  2:on   3:on   4:on   5:on   6:off
cups            0:off  1:off  2:on   3:on   4:on   5:on   6:off
...
sshd            0:off  1:off  2:on   3:on   4:on   5:on   6:off
syslog          0:off  1:off  2:on   3:on   4:on   5:on   6:off
tux             0:off  1:off  2:off  3:off  4:off  5:off  6:off
udev-post       0:off  1:off  2:off  3:on   4:on   5:on   6:off
vncserver       0:off  1:off  2:off  3:off  4:off  5:off  6:off
winbind         0:off  1:off  2:off  3:off  4:off  5:off  6:off
wpa_supplicant  0:off  1:off  2:off  3:off  4:off  5:off  6:off
xfs             0:off  1:off  2:on   3:on   4:on   5:on   6:off
ypbind          0:off  1:off  2:off  3:off  4:off  5:off  6:off
yum-updatesd    0:off  1:off  2:off  3:on   4:on   5:on   6:off
```

Some services in the example are never started, such as vncserver. Other services, such as the cups daemon, are started on runlevels 2 through 5.

Using the chkconfig command, you cannot tell if a service is currently running. To do that, you will need to use the service command. To help isolate only those services that are currently running, the service command is piped into the grep command and then sorted, as follows.

15

```
# service --status-all | grep running... | sort
anacron (pid 2162) is running...
atd (pid 2172) is running...
```

```
auditd (pid 1653) is running...
automount (pid 1952) is running...
console-kit-daemon (pid 2046) is running...
crond (pid 2118) is running...
cupsd (pid 1988) is running...
...
sshd (pid 2002) is running...
syslogd (pid 1681) is running...
xfs (pid 2151) is running...
yum-updatesd (pid 2205) is running...
```

You can also use both the `chkconfig` and the `service` commands to view an individual service's settings. Using both commands in the example that follows, you can view the `cups` daemon's settings.

```
# chkconfig --list cups
cups           0:off   1:off   2:on    3:on    4:on    5:on    6:off
#
# service cups status
cupsd (pid 1988) is running...
```

You can see that `cupsd` is set to start on every `runlevel` but `0`, `1`, and `6`, and from the `service` command, you can see that it is currently running. Also, the process ID (PID) number is given for the daemon.

Auditing the Upstart init daemon

To see all the services running on a Linux server using the `Upstart init` daemon, use the following command:

```
# initctl list | grep start/running
tty (/dev/tty3) start/running, process 1163
...
system-setup-keyboard start/running, process 656
prefdm start/running, process 1154
```

Keep in mind that many services may have not yet been ported to the `Upstart init` daemon. Therefore, you will also need to use the classic `SysVinit` command, `service`, to check for any leftover `SysVinit` services. Note that on some distributions, you may see a few services in *both* the `initctl` and the `service` command output.

```
# service --status-all | grep running
abrtd (pid  1118) is running...
acpid (pid  996) is running...
atd (pid  1146) is running...
...
rsyslogd (pid  752) is running...
sendmail (pid  1099) is running...
...
```

CAUTION

Just because a service is not in a `running` state does not mean it is unavailable. The service could be in a `stopped/wait` state, awaiting an event on the system. To see all the services, no matter what their state, remove the `grep` portion of the preceding `initctl list` and `service --status-all` commands.

To show the status of a single service, use `initctl` if the service has been ported to `Upstart` and the `service` command if it has not been ported yet. The following example shows two service statuses — one that has been ported to `Upstart` and one that has not.

```
# initctl status vpnc-cleanup
vpnc-cleanup stop/waiting
#
# service ssh status
sshd (pid  970) is running...
```

The `ssh` daemon has not yet been ported, on this Linux server, to `Upstart`. Therefore, `ssh` needs the `service` command with the status option to be used for auditing. The `vpnc-cleanup` service is an `Upstart` service. Thus, it needed the `initctl status` command to be used. In some distributions, such as Ubuntu, you can also use the `initctl status` command for services that have not yet been migrated to `Upstart`.

Auditing the systemd init

To see all the services that are being offered by a Linux server using `systemd`, use the following command:

```
# systemctl list-unit-files --type=service | grep -v disabled
UNIT FILE                                    STATE
abrt-ccpp.service                            enabled
abrt-oops.service                            enabled
abrt-vmcore.service                          enabled
abrtd.service                                enabled
alsa-restore.service                         static
alsa-store.service                           static
anaconda-shell@.service                      static
arp-ethers.service                           enabled
atd.service                                  enabled
auditd.service                               enabled
avahi-daemon.service                         enabled
bluetooth.service                            enabled
console-kit-log-system-restart.service       static
console-kit-log-system-start.service         static
console-kit-log-system-stop.service          static
crond.service                                enabled
cups.service                                 enabled
...
```

15

```
sshd-keygen.service                              enabled
sshd.service                                     enabled
system-setup-keyboard.service                    enabled
...
134 unit files listed.
```

Remember that the three status possibilities for a `systemd` service are `enabled`, `disabled`, or `static`. There's no need to include `disabled` in this audit, which is effectively accomplished by using the `-v` option on the `grep` command, as shown in the preceding example. The state of `static` is essentially `enabled`, and thus should be included.

NOTE

Most of the system services have been ported to `systemd`. However, if you want to double check, you can see the status of a particular service's migration at `http://fedoraproject.org/wiki/User:Johannbg/QA/Systemd/compatability`.

To see if a particular service is running, use the following command:

```
# systemctl status cups.service
cups.service - CUPS Printing Service
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: active (running) since Mon, 30 Apr 2015 12:36:31 -0400; 13h ago
  Main PID: 1315 (cupsd)
   CGroup: name=systemd:/system/cups.service
           └ 1315 /usr/sbin/cupsd -f
```

The `systemctl` command can be used to show the status of a single service. In the preceding example, the printing service was chosen. Notice that the name of the service is `cups.service`. A great deal of helpful information about the service is given here, such as the fact that it is enabled and active, its start time, and its process ID (PID) as well.

Now that you can audit services and determine some information about them, you need to know how to accomplish starting, stopping, and reloading the services on your Linux server.

Stopping and Starting Services

The tasks of starting, stopping, and restarting services typically refer to immediate needs — in other words, managing services without a server reboot. For example, if you want to temporarily stop a service, then you are in the right section. However, if you want to stop a service and not allow it to be restarted at server reboot, then you need to actually disable the service, which is covered in the section "Configuring Persistent Services," later in this chapter.

Stopping and starting the classic SysVinit daemon

The primary command for stopping and starting `SysVinit` services is the `service` command. With the `service` command, the name of the service you are wishing to control comes second in the command line. The last option is what you want to do to the service, `stop`, `start`, `restart`, and so on. The following example shows how to stop the `cups` daemon. Notice that an `OK` is given, which lets you know that `cupsd` has been successfully stopped.

```
# service cups status
cupsd (pid 5857) is running...
#
# service cups stop
Stopping cups:        [  OK  ]
#
# service cups status
cupsd is stopped
```

To start a service, you simply add a `start` option instead of a `stop` option on the end of the `service` command as follows.

```
# service cups start
Starting cups:         [  OK  ]
#
# service cups status
cupsd (pid 6860) is running...
```

To restart a `SysVinit` service, the `restart` option is used. This option will stop the service and then immediately start it again.

```
# service cups restart
Stopping cups:          [  OK  ]
Starting cups:          [  OK  ]
#
# service cups status
cupsd (pid 7955) is running...
```

When a service is already stopped, a `restart` will generate a `FAILED` status on the attempt to stop it. However, as shown in the example that follows, the service will be successfully started when a restart is attempted.

```
# service cups stop
Stopping cups:           [  OK  ]
#
# service cups restart
Stopping cups:           [FAILED]
Starting cups:           [  OK  ]
#
# service cups status
cupsd (pid 8236) is running...
```

Reloading a service is different from restarting a service. When you `reload` a service, the service itself is not stopped. Only the service's configuration files are loaded again. The following example shows how to reload the `cups` daemon.

```
# service cups status
cupsd (pid 8236) is running...
#
# service cups reload
Reloading cups:            [  OK  ]
#
# service cups status
cupsd (pid 8236) is running...
```

If a `SysVinit` service is stopped when you attempt to reload it, you will get a `FAILED` status. This is shown in the following example:

```
# service cups status
cupsd is stopped
#
# service cups reload
Reloading cups:             [FAILED]
```

Stopping and starting the Upstart init daemon

The primary command for stopping and starting `Upstart init` services is the `initctl` command. The options are very similar to `SysVinit`'s `service` command:

- **Stopping a service with Upstart init** — In the following example, the status of the `cups` daemon is checked and then stopped using the `initctl stop cups .service` command.

  ```
  # initctl status cups
  cups start/running, process 2390
  #
  # initctl stop cups
  cups stop/waiting
  #
  # initctl status cups
  cups stop/waiting
  ```

- **Starting a service with Upstart init** — In the following example, the `cups` daemon is stopped using the `initctl stop cups.service` command.

  ```
  # initctl start cups
  cups start/running, process 2408
  #
  # initctl status cups
  cups start/running, process 2408
  ```

- **Restarting a service with Upstart init** — Restarting a service with `Upstart init` will stop and then start the service. However, the configuration file will not be reloaded.

```
# initctl restart cups
cups start/running, process 2490
#
# initctl status cups
cups start/running, process 2490
#
```

- **Reloading a service with Upstart init** — Reloading will *not* stop and start the service. It only loads the configuration file again. This is the option to use when you have made changes to the configuration file.

  The example below illustrates how to reload the `cups` daemon with `initctl`. Notice that the process ID (PID) is still `2490`, which is the same as it was in the example for restarting the `cups` daemon because the process was not stopped and started in the reload process.

```
# initctl reload cups
#
# initctl status cups
cups start/running, process 2490
```

NOTE

You need root privileges to stop and start services. However, you do not need root privileges to check a service's status.

Stopping and starting the systemd daemon

For the `systemd` daemon, the `systemctl` command will work for stopping, starting, reloading, and restarting. The options to the `systemctrl` command should look familiar.

Stopping a service with systemd

In the example that follows, the status of the `cups` daemon is checked and then stopped using the `systemctl stop cups.service` command:

```
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: active (running) since Mon, 30 Apr 2015 12:36:3...
  Main PID: 1315 (cupsd)
    CGroup: name=systemd:/system/cups.service
           └ 1315 /usr/sbin/cupsd -f
#
# systemctl stop cups.service
```

15

```
#
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: inactive (dead) since Tue, 01 May 2015 04:43:4...
    Process: 1315 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
    CGroup: name=systemd:/system/cups.service
```

Notice that when the status is taken, after stopping the `cups` daemon, the service is `inactive (dead)` but still considered `enabled`. This means that the `cups` daemon will still be started upon server boot.

Starting a service with systemd

Starting the `cups` daemon is just as easy as stopping it. The example that follows demonstrates this ease.

```
# systemctl start cups.service
#
# systemctl status cups.service
cups.service - CUPS Printing Service
    Loaded: loaded (/lib/systemd/system/cups.service; enabled)
    Active: active (running) since Tue, 01 May 2015 04:43:5...
  Main PID: 17003 (cupsd)
    CGroup: name=systemd:/system/cups.service
            └ 17003 /usr/sbin/cupsd -f
```

After the `cups` daemon is started, using `systemctl` with the status option shows the service is `active (running)`. Also, its process ID (PID) number, `17003`, is shown.

Restarting a service with systemd

Restarting a service means that a service is stopped and then started again. If the service was not currently running, restarting it will simply start the service.

```
# systemctl restart cups.service
#
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; enabled)
   Active: active (running) since Tue, 01 May 2015 04:45:2...
  Main PID: 17015 (cupsd)
   CGroup: name=systemd:/system/cups.service
           └ 17015 /usr/sbin/cupsd -f
```

You can also perform a conditional restart of a service using `systemctl`. A conditional restart only restarts a service if it is currently running. Any service in an inactive state will not be started.

```
# systemctl status cups.service
cups.service - CUPS Printing Service
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: inactive (dead) since Tue, 01 May 2015 06:03:32...
 Process: 17108 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
  CGroup: name=systemd:/system/cups.service
#
# systemctl condrestart cups.service
#
# systemctl status cups.service
cups.service - CUPS Printing Service
  Loaded: loaded (/lib/systemd/system/cups.service; enabled)
  Active: inactive (dead) since Tue, 01 May 2015 06:03:32...
 Process: 17108 ExecStart=/usr/sbin/cupsd -f
 (code=exited, status=0/SUCCESS)
  CGroup: name=systemd:/system/cups.service
```

Notice in the example that the cups daemon was in an inactive state. When the conditional restart was issued, no error messages were generated! The cups daemon was not started because conditional restarts will affect active services. Thus, it is always a good practice to check the status of a service, after stopping, starting, conditionally restarting, and so on.

Reloading a service with systemd

Reloading a service is different from restarting a service. When you reload a service, the service itself is not stopped. Only the service's configuration files are loaded again.

```
# systemctl reload cups.service
Failed to issue method call: Job type reload is
 not applicable for unit cups.service.
#
# systemctl reload sshd.service
#
# systemctl status sshd.service
sshd.service - OpenSSH server daemon
    Loaded: loaded (/lib/systemd/system/sshd.service; enabled)
    Active: active (running) since Mon, 30 Apr 2015 12:35:2...
    Process: 17009 ExecReload=/bin/kill -HUP $MAINPID
 (code=exited, status=0/SUCCESS)
  Main PID: 786 (sshd)
    CGroup: name=systemd:/system/sshd.service
           └ 786 /usr/sbin/sshd -D
```

Doing a reload of a service, instead of a restart, will prevent any pending service operations from being aborted. A reload is a better method for a busy Linux server.

Now that you know how to stop and start services for troubleshooting and emergency purposes, you can learn how to enable and disable services.

Configuring Persistent Services

You use `stop` and `start` for immediate needs, not for services that need to be persistent. A *persistent* service is one that is started at server boot time. Services that need to be set as persistent are typically new services that the Linux server will be offering.

Configuring the classic SysVinit daemon persistent services

One of the nice features of the classic `SysVinit` daemon is that making a particular service persistent or removing its persistence is very easy to do. Consider the following example:

```
# chkconfig --list cups
cups            0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

On this Linux server, the `cups` daemon is not started at any `runlevel`, as shown with the `chkconfig` command. You can also check and see if there are any start (`S`) symbol links set up in each of the seven `runlevel` directories, `/etc/rc.d/rc?.d`. Remember that `SysVinit` keeps symbolic links here for starting and stopping various services at certain runlevels. Each directory represents a particular `runlevel`; for example, `rc5.d` is for `runlevel 5`. Notice that only files starting with a `K` are listed, so there are links for killing off the `cups` daemon. None are listed with `S`, which is consistent with `chkconfig` that the `cups` daemon does not start at any `runlevel` on this server.

```
# ls /etc/rc.d/rc?.d/*cups
/etc/rc.d/rc0.d/K10cups  /etc/rc.d/rc3.d/K10cups
/etc/rc.d/rc1.d/K10cups  /etc/rc.d/rc4.d/K10cups
/etc/rc.d/rc2.d/K10cups  /etc/rc.d/rc5.d/K10cups
/etc/rc.d/rc6.d/K10cups
```

To make a service persistent at a particular `runlevel`, the `chkconfig` command is used again. Instead of the `--list` option, the `--level` option is used, as shown in the following code:

```
# chkconfig --level 3 cups on
#
# chkconfig --list cups
cups            0:off  1:off  2:off  3:on   4:off  5:off  6:off
#
# ls /etc/rc.d/rc3.d/S*cups
/etc/rc.d/rc3.d/S56cups
```

The service's persistence at `runlevel 3` is verified by using both the `chkconfig --list` command and looking at the `rc3.d` directory for any files starting with the letter `S`.

To make a service persistent on more than one `runlevel`, you can do the following:

```
# chkconfig --level 2345 cups on
#
```

```
# chkconfig --list cups
cups            0:off  1:off  2:on   3:on   4:on   5:on   6:off
#
# ls /etc/rc.d/rc?.d/S*cups
/etc/rc.d/rc2.d/S56cups  /etc/rc.d/rc4.d/S56cups
/etc/rc.d/rc3.d/S56cups  /etc/rc.d/rc5.d/S56cups
```

Disabling a service is just as easy as enabling one with SysVinit. You just need to change the on in the chkconfig command to off. The following example demonstrates using the chkconfig command to disable the cups service at runlevel 5.

```
# chkconfig --level 5 cups off
#
# chkconfig --list cups
cups            0:off  1:off  2:on   3:on   4:on   5:off   6:off
#
# ls /etc/rc.d/rc5.d/S*cups
ls: cannot access /etc/rc.d/rc5.d/S*cups: No such file or directory
```

As expected, there is now no symbolic link, starting with the letter S, for the cups daemon in the /etc/rc.d/rc5.d directory.

Configuring Upstart init daemon persistent services

The Upstart init daemon emits the startup signal that triggers the service jobs to start. At server boot time, various jobs may themselves emit signals. These emitted signals then cause other jobs to start. Thus, the key to making a service persistent is to ensure the service's definition file is triggered by one of the signals emitted as the server boots.

Remember that the Upstart init daemon's job definition files are located in /etc/init. Consider the following job definition file for the ssh daemon:

```
# cat /etc/init/ssh.conf
# ssh - OpenBSD Secure Shell server
#
# The OpenSSH server provides secure shell access to the system.

description   "OpenSSH server"

start on filesystem or runlevel [2345]
stop on runlevel [!2345]

respawn
```

To determine what emitted event(s) trigger a service, look for start on in the configuration file. The ssh daemon is triggered by several possible emitted events, filesystem, runlevel 2, runlevel 3, runlevel 4, or runlevel 5. Basically, the ssh daemon starts upon server boot and is set as persistent. The syntax for the runlevel events, runlevel [2345], is used in many of the job files and denotes that the name "runlevel" can end in either 2, 3, 4, or 5.

To make a job persistent (start at boot), you will need to modify the `start on` line in its configuration file so that it starts on certain events emitted at server boot. To disable a job at boot, just comment out the `start on` line with a pound sign (#). See the section "Adding New or Customized Services" for `Upstart` for a more thorough explanation of these configuration files.

Configuring systemd init persistent services

For the `systemd` daemon, again the `systemctl` command is used. With it, you can disable and enable services on the Linux server.

Enabling a service with systemd

Using the `enable` option on the `systemctl` command will set a service to always start at boot (be persistent). The following shows exactly how to accomplish this:

```
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; disabled)
   Active: inactive (dead) since Tue, 01 May 2015 06:42:38 ...
 Main PID: 17172 (code=exited, status=0/SUCCESS)
   CGroup: name=systemd:/system/cups.service
#
# systemctl enable cups.service
ln -s '/lib/systemd/system/cups.service'
'/etc/systemd/system/printer.target.wants/cups.service
ln -s '/lib/systemd/system/cups.socket'
'/etc/systemd/system/sockets.target.wants/cups.socket'
ln -s '/lib/systemd/system/cups.path' '
/etc/systemd/system/multi-user.target.wants/cups.path'
#
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; enabled)
   Active: inactive (dead) since Tue, 01 May 2015 06:42:38...
 Main PID: 17172 (code=exited, status=0/SUCCESS)
   CGroup: name=systemd:/system/cups.service
```

Notice that the status of `cups.service` changes from `disabled` to `enabled` after using the `enable` option on `systemctl`. Also, notice that the `enable` option simply creates a few symbolic links. You may be tempted to create these links yourself. However, the preferred method is to use the `systemctl` command to accomplish this.

Disabling (removing) a service with systemd

You can use the `disable` option on the `systemctl` command to keep a service from starting at boot. However, it does not immediately stop the service. You need to use the stop option discussed in the section "Stopping a service with systemd." The following example shows how to disable a currently enabled service.

```
# systemctl disable cups.service
rm '/etc/systemd/system/printer.target.wants/cups.service'
rm '/etc/systemd/system/sockets.target.wants/cups.socket'
rm '/etc/systemd/system/multi-user.target.wants/cups.path'
#
# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded (/lib/systemd/system/cups.service; disabled)
   Active: active (running) since Tue, 01 May 2015 06:06:41...
 Main PID: 17172 (cupsd)
   CGroup: name=systemd:/system/cups.service
           └ 17172 /usr/sbin/cupsd -f
```

The disable option simply removes a few files via the preferred method of the systemctl command. Notice also in the preceding example that although the cups service is now disabled, the cups daemon is still active (running).

The systemd init has some services that cannot be disabled. These services are static services. Consider the following service, dbus.service:

```
# systemctl status dbus.service
dbus.service - D-Bus System Message Bus
  Loaded: loaded (/lib/systemd/system/dbus.service; static)
  Active: active (running) since Mon, 30 Apr 2015 12:35:...
 Main PID: 707 (dbus-daemon)
...
#
# systemctl disable dbus.service
#
# systemctl status dbus.service
dbus.service - D-Bus System Message Bus
  Loaded: loaded (/lib/systemd/system/dbus.service; static)
  Active: active (running) since Mon, 30 Apr 2015 12:35:...
 Main PID: 707 (dbus-daemon)
...
```

When the systemctl disable command is issued on dbus.service, it is simply ignored. Remember that static stands for "statically enabled" and it means that the service is enabled by default and cannot be disabled, even by root. Any services that should not be disabled on your Linux server are set to static.

Now that you understand how to enable (or disable) individual services to be persistent, you need to look at service groups as a whole. The next section covers how to start groups of services at boot time.

Configuring a Default runlevel or target unit

Whereas a persistent service is one that is started at server boot time, a persistent (default) `runlevel` or `target unit` is a group of services that are started at boot time. Both classic `SysVinit` and `Upstart` define these groups of services as runlevels, while `systemd` calls them `target units`.

Configuring the classic SysVinit daemon default runlevel

You set the persistent `runlevel` for a Linux server using the `SysVinit` daemon in the `/etc/inittab` file. A portion of this file is shown here:

```
# cat /etc/inittab
#
# inittab       This file describes how the INIT process should
#               set up the system in a certain run-level.
...
id:5:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
...
```

The bold line in the example shows that the current default `runlevel` is `runlevel 5`. To change this, simply edit the `/etc/inittab` file using your favorite editor and change the `5` to one of the following runlevels: `2`, `3`, or `4`. Do *not* use the runlevels `0` or `6` in this file! This would cause your server to either halt or reboot when it is started up.

Configuring the Upstart init daemon default runlevel

Some distributions still use the `/etc/inittab` file to set the default `runlevel`, whereas others use the `/etc/init/rc-sysinit.conf` file.

Fedora and RHEL's `Upstart init` daemon still uses the `/etc/inittab` file. Therefore, just change the default `runlevel` as you would on a `SysVinit` system.

Ubuntu's `Upstart init` daemon uses the `/etc/init/rc-sysinit.conf` file to set the default `runlevel`, a portion of which is shown in the code that follows. The code line to change is `env DEFAULT_RUNLEVEL=`. Simply edit this file and change that number to the `runlevel` you desire. However, remember that Ubuntu's `runlevel 2` is equivalent to runlevels `3`, `4`, and `5`.

```
$ cat /etc/init/rc-sysinit.conf
# rc-sysinit - System V initialisation compatibility
...
# Default runlevel, this may be overriden on the kernel command-line
# or by faking an old /etc/inittab entry
env DEFAULT_RUNLEVEL=2
```

Configuring the systemd init default target unit

`systemd init` uses the term "`target units`" for groups of services to be started. The following shows the various `target units` you can configure to be persistent and their equivalent backward-compatible, runlevel-specific `target units`.

- `multi-user.target =`
  - `runlevel2.target`
  - `runlevel3.target`
  - `runlevel4.target`
- `graphical.target = runlevel5.target`

The persistent `target unit` is set via a symbolic link to the `default.target` unit file. Consider the following:

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Mar 13 17:27
 /etc/systemd/system/default.target ->
 /lib/systemd/system/runlevel5.target
#
# ls -l /lib/systemd/system/runlevel5.target
lrwxrwxrwx. 1 root root 16 Mar 27 15:39
 /lib/systemd/system/runlevel5.target ->
 graphical.target
```

The example shows that the current persistent `target unit` on this server is `runlevel5.target` because `default.target` is a symbolic link to the `runlevel5.target` unit file. However, notice that `runlevel5.target` is also a symbolic link and it points to `graphical.target`. Thus, this server's current persistent `target unit` is `graphical.target`.

To set a different `target unit` to be persistent, you simply need to change the symbolic link for `default.target`. To be consistent, stick with the `runlevel target units` if they are used on your server.

The following example changes the server's persistent `target unit` from `graphical.target` to `multi-user.target` by changing the `default.target` symbolic link from `runlevel5.target` to `runlevel3.target`. The `f` option is used on the `ls -s` command to force any current symbolic link to be broken and the new designated symbolic link to be enforced.

```
# ls -l /lib/systemd/system/runlevel3.target
lrwxrwxrwx. 1 root root 17 Mar 27 15:39
 /lib/systemd/system/runlevel3.target ->
 multi-user.target
#
# ln -sf /lib/systemd/system/runlevel3.target
 /etc/systemd/system/default.target
```

```
#
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 May  1 10:06
 /etc/systemd/system/default.target ->
 /lib/systemd/system/runlevel3.target
```

When the server is rebooted, the `multi-user.target` will be the persistent `target unit`. Any services in the `multi-user.target` unit will be started (activated) at that time.

Adding New or Customized Services

Occasionally you will have to add a new service to your Linux server. Also, you may have to customize a particular service. When these needs arise, you must follow specific steps for your Linux server's `init` daemon to either take over the management of the service or recognize the customization of it.

Adding new services to classic SysVinit daemon

When adding a new or customized service to a Linux `SysVinit` server, you have to complete three steps in order to have the service managed by `SysVinit`.

1. Create a new or customized service script file.
2. Move the new or customized service script to the proper location for `SysVinit` management.
3. Add the service to a specific `runlevel`(s).

Step 1: Create a new or customized service script file

If you are customizing a service script, simply make a copy of the original unit file from `/etc/rc.d/init.d` and add any desired customizations.

If you are creating a new script, you will need to make sure you handle all the various options you want the `service` command to accept for your service, such as `start`, `stop`, `restart`, and so on.

For a new script, especially if you have never created a service script before, it would be wise to make a copy of a current service script from `/etc/rc.d/init.d` and then modify it to meet your new service's needs. Consider the following partial example of the `cupsd` service's script:

```
# cat /etc/rc.d/init.d/cups
#!/bin/sh
#
...
```

```
start () {
        echo -n $"Starting $prog: "

        # start daemon
        daemon $DAEMON
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && touch /var/lock/subsys/cups
        return $RETVAL
}

stop () {
        # stop daemon
        echo -n $"Stopping $prog: "
        killproc $DAEMON
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && rm -f /var/lock/subsys/cups
}

restart() {
        stop
        start
}

case $1 in
...
```

The `cups` daemon's service script starts out by creating functions for each of the `start`, `stop`, and `restart` options. If you feel uncomfortable with shell script writing, review Chapter 7, "Writing Simple Shell Scripts," or consult a resource such as *Linux Command Line and Shell Scripting Bible* (Wiley, 2011) to improve your skills.

Step 2: Move the service script

Once you have modified or created and tested your service's script file, you can move it to the proper location, `/etc/rc.d/init.d`:

```
# cp My_New_Service /etc/rc.d/init.d
#
# ls /etc/rc.d/init.d/My_New_Service
/etc/rc.d/init.d/My_New_Service
```

Step 3: Add the service to runlevels

This final step is needed only if you want the service to be persistent at certain runlevels. You must create a symbolic link for every `runlevel` at which you want the service to be persistent. Below are the actions needed to accomplish this final step.

1. Check each runlevel directory you want the service to start on and determine what the appropriate S *number* should be for your service. For example, the My_New_Service should be started at runlevel 3 and after all the runlevel 3 services have been started. Therefore, S100 is an appropriate S number for the symbolic filename, as shown here:

```
# ls /etc/rc.d/rc3.d/S*
/etc/rc.d/rc3.d/S00microcode_ctl
/etc/rc.d/rc3.d/S04readahead_early
/etc/rc.d/rc3.d/S05kudzu
...
/etc/rc.d/rc3.d/S98haldaemon
/etc/rc.d/rc3.d/S99firstboot
/etc/rc.d/rc3.d/S99local
/etc/rc.d/rc3.d/S99smartd
# ln -s /etc/rc.d/init.d/My_New_Service
 /etc/rc.d/rc3.d/S100My_New_Service
#
# ls -l /etc/rc.d/rc3.d/S100My_New_Service
lrwxrwxrwx 1 root root 31 2015-05-07 11:01
 /etc/rc.d/rc3.d/S100My_New_Service ->
 /etc/rc.d/init.d/My_New_Service
```

2. Once you have made the symbolic link(s), test that your new or modified service will work as expected before performing a server reboot.

```
# service My_New_Service start
Starting My_New_Service:       [  OK  ]
#
# service My_New_Service stop
Stopping My_New_Service:       [  OK  ]
```

Once everything is in place, your new or modified service will start at every runlevel you have selected on your system. Also, you can start or stop it manually using the service command.

Adding new services to the Upstart init daemon

You need to complete only one step to add a new service or customize an existing service with the Upstart init daemon. Just add a new job configuration file or modify an existing one. However, this one step can be rather complicated.

The Upstart service job configuration files are all located in the /etc/init directory. These files are plaintext only. They use a special syntax for directing the Upstart init daemon on how to deal with a particular service. The following example of a configuration file has some very simple syntax:

```
# cat ck-log-system-restart.conf
# Upstart event
# ck-log-system-restart - write system restart to log
```

```
#

start on runlevel 6

task
exec /usr/sbin/ck-log-system-restart
```

Any pound sign (#) denotes a comment line and is ignored by `Upstart`. The other lines are called stanzas and have special syntax for controlling `Upstart` jobs. The stanzas from the above file are as follows:

- `start on` — This stanza defines what emitted event will start the service or task. In this particular case, when the `runlevel 6` event is emitted, the `ck-log-system-restart` starts.
- `task` — The stanza here defines that this particular job is a `task` job as opposed to a `service`.
- `exec` — This stanza defines what program will be run to start the task. Instead of the `exec` stanza, you can embed an actual command line script to run here by using the script stanza before the actual code and `end script` after it.

A slightly more complicated job configuration file is shown next — for the `cron` daemon. There are some additional stanzas that were not in the previous example. Notice that the `task` stanza is missing in the file. This indicates that this particular job is a `service` job instead of a `task` job.

```
# cat cron.conf
# cron - regular background program processing daemon
#
# cron is a standard UNIX program that runs user-specified
# programs at periodic scheduled times

description   "regular background program processing daemon"

start on runlevel [2345]
stop on runlevel [!2345]

expect fork
respawn

exec cron
```

The additional stanzas in this example are as follows:

- `description` — This stanza is optional and simply describes the service.
- `start on` — Though the `start on` portion of this stanza was previously covered, the `[2345]` syntax was not. Using brackets means that the stanza is valid for any of those numbers. Thus, the service will start on `runlevel 2`, `3`, `4`, *or* `5`.

15

- `stop on` — The stanza here defines what emitted events the service will stop on. The `[!2345]` in this stanza means *not* `runlevel 2` or `3` or `4` or `5`. In other words, it will only stop on `runlevel 0`, `runlevel 1`, or `runlevel 6`.
- `expect` — This particular stanza is rather important and a little tricky. The `expect fork` syntax will allow `Upstart` to track this daemon and any of its child processes (forks).
- `respawn` — The stanza here tells `Upstart` to restart this service should it ever be terminated via a means outside of its normal `stop on`.

TIP

To test your new or modified job configuration files, you can set the `start on` stanza to a non-standard event. In other words, you can make up your own event name. For example, use the event name `MyTest`. To test the new configuration file, you type `initctl emit MyTest` at the command line. If your configuration file works correctly, then modify the `start on` stanza to the correct `Upstart` event.

Every job configuration file must follow at least three rules. The job configuration file must:

- Not be empty
- Be syntactically correct
- Contain at least one legal stanza

Although there are only three rules, creating or modifying a service job configuration file correctly can be a rather difficult task. See `http://upstart.ubuntu.com/getting-started.html` or `http://upstart.ubuntu.com/cookbook` for help on the syntax needed for these files. Also, you can find out more about events that the `Ubuntu init` daemon emits by typing **`man upstart-events`** at the command line.

Adding new services to systemd init

When adding a new or customized service to a Linux `systemd` server, you have to complete three steps in order to have the service managed by `systemd`:

1. Create a new or customized service configuration unit file for the new or customized service.
2. Move the new or customized service configuration unit file to the proper location for `systemd` management.
3. Add the service to a specific `target unit`'s `Wants` if you want to have the new or customized service start automatically with other services.

Step 1: Create a new or customized service configuration unit file

If you are customizing a service configuration unit file, simply make a copy of the original unit file from `/lib/systemd/system` and add any desired customizations.

For new files, obviously, you will be creating a `service unit` configuration file from scratch. Consider the following basic `service unit` file template. At bare minimum, you need `Description` and `ExecStart` options for a `service unit` configuration file.

```
# cat My_New_Service.service
[Unit]
Description=My New Service

[Service]
ExecStart=/usr/bin/My_New_Service
```

For additional help on customizing or creating a new configuration unit file and the various needed options, you can use the man pages. At the command line, type **`man systemd.service`** to find out more about the various `service unit` file options.

Step 2: Move the service configuration unit file

Before you move the new or customized service configuration unit file, you need to be aware that there are two potential locations to store service configuration unit files. The one you choose will determine whether or not the customizations take effect and if they remain persistent through software upgrades.

You can place your system service configuration unit file in one of the following two locations:

- `/etc/systemd/system`
  - This location is used to store customized local service configuration unit files.
  - Files in this location are not overwritten by software installations or upgrades.
  - Files here are used by the system *even* if there is a file of the same name in the `/lib/systemd/system` directory.
- `/lib/systemd/system`
  - This location is used to store system service configuration unit files.
  - Files in this location are overwritten by software installations and upgrades.
  - Files here are used by the system *only* if there is *not* a file of the same name in the `/etc/systemd/system` directory.

Thus, the best place to store your new or customized service configuration unit file is in `/etc/systemd/system`.

TIP

When you create a new or customized service, in order for the change to take effect without a server reboot, you will need to issue a special command. At the command line, type **`systemctl daemon-reload`**.

15

Step 3: Add the service to the Wants directory

This final step is optional. It needs to be done only if you want your new service to start with a particular `systemd target unit`. For a service to be activated (started) by a particular `target unit`, it must be in that `target unit`'s `Wants` directory.

First, add the line **`WantedBy=`*`desired.target`*** to the bottom of your service configuration unit file. The following example shows that the desired `target unit` for this new service is `multi-user.target`.

```
# cat /etc/systemd/system/My_New_Service.service
[Unit]
Description=My New Fake Service

[Service]
ExecStart=/usr/bin/My_New_Service

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

To add a new `service unit` to a `target unit`, you need to create a symbolic link. The following example shows the files located in the `multi-user.target` unit's `Wants` directory. Previously, in the section "Understanding systemd init," the `systemctl` command was used to list out `Wants`, and it is still the preferred method. Notice that in this directory, the files are symbolic links pointing to `service unit` configuration files in the `/lib/systemd/system` directory.

```
# ls /etc/systemd/system/multi-user.target.wants
abrt-ccpp.service     cups.path           remote-fs.target
abrtd.service         fcoe.service        rsyslog.service
abrt-oops.service     irqbalance.service  sendmail.service
abrt-vmcore.service   lldpad.service      sm-client.service
atd.service           mcelog.service      sshd-keygen.service
auditd.service        mdmonitor.service   sshd.service
...
#
# ls -l /etc/systemd/system/multi-user.target.wants
total 0
lrwxrwxrwx. 1 root root 37 Nov  2 22:29
 abrt-ccpp.service ->
 /lib/systemd/system/abrt-ccpp.service
lrwxrwxrwx. 1 root root 33 Nov  2 22:29
 abrtd.service ->
 /lib/systemd/system/abrtd.service
...
lrwxrwxrwx. 1 root root 32 Apr 26 20:05
 sshd.service ->
 /lib/systemd/system/sshd.service
```

The following illustrates the process of adding a symbolic link file for My_New_Service:

```
# ln -s /etc/systemd/system/My_New_Service.service
 /etc/systemd/system/multi-user.target.wants/My_New_Service.service
```

A symbolic link has been created in the multi-user.target.wants directory. Now, the new service, My_New_Service, will be activated (started) when the multi-user.target unit is activated.

TIP

If you want to change the `systemd target unit` for a service, you will need to change the symbol link to point to a new target `Wants` directory location. Use the `ls -sf` command to force any current symbolic link to be broken and the new designated symbolic link to be enforced.

Together, the three steps will get your new or customized service added to a Linux systemd server. Remember that at this point, a new service will not be running until a server reboot. To start the new service before a reboot, review the commands in the section "Stopping and Starting Services."

Summary

How you start and stop services is dependent upon what init daemon is used by your Linux server. Before you do any service management, be sure to use the examples in this chapter to help you determine your Linux server's init daemon.

The concepts of starting and stopping services go along with other service management concepts, such as making a service persistent, starting certain services at server boot time, reloading a service, and restarting a service. All of these concepts will be very helpful as you learn about configuring and managing a Linux print server in the next chapter.

Exercises

Refer to the material in this chapter to complete the tasks that follow. If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task). Try each of the exercises before referring to the answers. These tasks assume you are running a Fedora or Red Hat Enterprise Linux system (although some tasks will work on other Linux systems as well).

1. Determine which init daemon your server is currently using.
2. What init daemon is sshd using on your Linux server?

3. Determine your server's previous and current `runlevel`.
4. How can you change the default `runlevel` or `target unit` on your Linux server?
5. For each `init` daemon, what command(s) will list out services running (or active) on your server?
6. List out the running (or active) services on your Linux server.
7. For each `init` daemon, what command(s) will show a particular service's current status?
8. Show the status of the `cups` daemon on your Linux server.
9. Attempt to restart the `cups` daemon on your Linux server.
10. Attempt to reload the `cups` daemon on your Linux server.

CHAPTER

16

Configuring a Print Server

IN THIS CHAPTER

Understanding printing in Linux

Setting up printers

Using printing commands

Managing document printing

Sharing printers

You can configure your Linux system to use printers that are connected directly to it (via a USB or parallel port) or that are available for printing over the network. Likewise, any printer you configure on your local system can be shared with users on other Linux, Windows, or Mac systems by opening up your printer as a print server.

You configure a printer as a native Linux printer in Fedora, RHEL, Ubuntu, and other Linux systems with the Common UNIX Printing System (CUPS). To configure a printer to work as a Microsoft Windows style of print server, you can use the Samba service in Linux.

This chapter focuses on CUPS. In particular, it shows you the graphical front end to CUPS, called the Printer Configuration window, which comes with Fedora, Red Hat Enterprise Linux, and other Linux distributions. Using Printer Configuration, you can also configure your printers as print servers so people can print to your printer from their own computers.

If you don't have a desktop or want to print from within a shell script, this chapter shows you how to use printing commands. From the command line, print commands such as `lpr` are available for carrying out printing. Commands also exist for querying print queues (`lpq`), manipulating print queues (`lpc`), and removing print queues (`lprm`).

Common UNIX Printing System

CUPS has become the standard for printing from Linux and other UNIX-like operating systems. It was designed to meet today's needs for standardized printer definitions and sharing on Internet Protocol–based networks (as most computer networks are today). Nearly every Linux distribution today comes with CUPS as its printing service. Here are some of the service's features:

- **IPP** — CUPS is based on the Internet Printing Protocol (`http://www.pwg.org/ipp`), a standard that was created to simplify how printers can be shared over IP networks. In the IPP model, printer servers and clients who want to print can exchange information about the model and features of a printer using HTTP (that is, web content) protocol. A server can also broadcast the availability of a printer so a printing client can easily find a list of locally available printers without configuration.
- **Drivers** — CUPS also standardized how printer drivers are created. The idea was to have a common format that could be used by printer manufacturers so that a driver could work across all different types of UNIX systems. That way, a manufacturer had to create the driver only once to work for Linux, Mac OS X, and a variety of UNIX derivatives.
- **Printer classes** — You can use printer classes to create multiple print server entries that point to the same printer or one print server entry that points to multiple printers. In the first case, multiple entries can each allow different options (such as pointing to a particular paper tray or printing with certain character sizes or margins). In the second case, you can have a pool of printers so that printing is distributed. In this instance, a malfunctioning printer or a printer that is dealing with very large documents won't bring all printing to a halt. CUPS also supports *implicit classes*, which are print classes that form by merging identical network printers automatically.
- **Printer browsing** — With printer browsing, client computers can see any CUPS printers on your local network with browsing enabled. As a result, clients can simply select the printers they want to use from the printer names broadcast on the network, without needing to know in advance what the printers are named and where they are connected. You can turn off the feature to prevent others on the local network from seeing a printer.
- **UNIX print commands** — To integrate into Linux and other UNIX environments, CUPS offers versions of standard commands for printing and managing printers that have been traditionally offered with UNIX systems.

Instead of using the Printer Configuration window, there are other ways you can configure CUPS printing as well:

- **Configuring CUPS from a browser** — The CUPS project itself offers a web-based interface for adding and managing printers. With the `cupsd` service running, type **`localhost:631`** from a web browser on the computer running the CUPS service to manage printing. (See the section "Using web-based CUPS administration" later in this chapter.)
- **Configuring CUPS manually** — You also can configure CUPS manually (that is, edit the configuration files and start the `cupsd` daemon from the command line). Configuration files for CUPS are contained in the `/etc/cups` directory. In particular, you might be interested in the `cupsd.conf` file, which identifies

permission, authentication, and other information for the printer daemon, and `printers.conf`, which identifies addresses and options for configured printers. Use the `classes.conf` file to define local printer classes.

Coming From Windows

You can print to CUPS from non-UNIX systems as well. For example, you can use a PostScript printer driver to print directly from Windows XP to your CUPS server. You can use CUPS without modification by configuring the XP computer with a PostScript driver that uses `http://printservername:631/printers/targetPrinter` as its printing port.

You may also be able to use the native Windows printer drivers for the printer instead of the PostScript driver. If the native Windows driver does not work out-of-the-box on your CUPS print queue, you can create a Raw Print Queue under CUPS and use that instead. The Raw Print Queue will directly pass through the data from the Windows native print driver to the printer.

To use CUPS, you need to have the cups package installed in Fedora or RHEL. Most desktop Linux distributions include CUPS during the initial system install. If it is not installed in a Fedora or RHEL install, install it by typing the following:

```
# yum install cups
```

Setting Up Printers

Although using the printer administration tools specifically built for your distribution is usually best, many Linux systems simply rely on the tools that come with the CUPS software package.

This section explores how to use CUPS web-based administration tools that come with every Linux distribution and then examines the printer configuration tool `system-config-printer`, which comes with Fedora and Red Hat Enterprise Linux systems to enable you to set up printers. In some cases, no configuration is necessary, because connected printers can be automatically detected and configured.

Adding a printer automatically

CUPS printers can be configured to automatically broadcast their availability on the network so a client system can detect and use them without configuration. Connect a USB printer to your computer, and the printer can be automatically detected and made available. In fact, if you attach a local printer in Fedora and the print driver is not yet installed, you are prompted to install the software packages needed to use the printer.

The first time you go to print a document or view your printer configuration tool, the printers will be there ready to use. Further configuration can be done using the web-based CUPS administration tool or the Printer Configuration window.

Using web-based CUPS administration

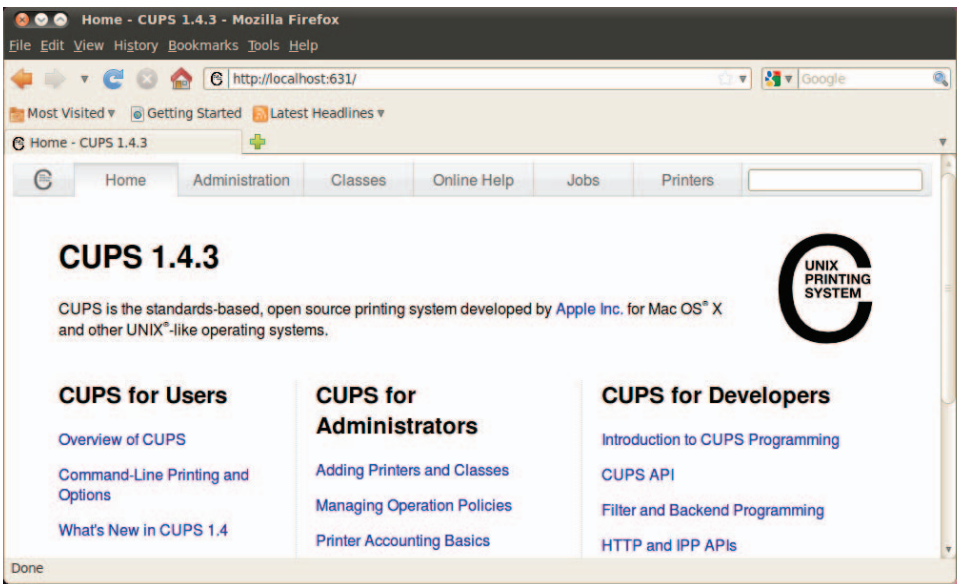
CUPS offers its own web-based administrative tool for adding, deleting, and modifying printer configurations on your computer. The CUPS print service (using the `cupsd` daemon) listens on port 631 to provide access to the CUPS web-based administrative interface and share printers.

If CUPS is already running on your computer, you can immediately use CUPS web-based administration from your web browser. To see whether CUPS is running and to start setting up your printers, open a web browser on the local computer and type the following into its location box: **`http://localhost:631/`**.

A prompt for a valid login name and password may appear when you request functions that require it. If so, type the root login name and the root user's password, and then click OK. A screen similar to the one shown in Figure 16.1 appears.

FIGURE 16.1

CUPS provides a web-based administration tool.



By default, web-based CUPS administration is available only from the local host. To access web-based CUPS administration from another computer, from the main CUPS page, select the Administration tab, select the check box next to Allow remote administration, and select the Change Settings button. Then, from a remote browser, you can access the CUPS Administration page by going to port 631 on the CUPS server (for example, `http://host.example.com:631`).

To configure a printer that is not automatically detected, you can add a printer from the Admin screen. With the Admin screen displayed, you can add a printer as follows:

1. **Click the Add Printer button.** The Add New Printer screen appears.
2. **Select the device to which the printer is connected.** The printer can be connected locally to a parallel, SCSI, serial, or USB port directly on the computer. Alternatively, you can select a network connection type for Apple printers (appSocket/HP JetDirect), Internet Printing Protocol (`http` or `ipp`), or a Windows printer (using Samba or SMB).
3. **If prompted for more information, you may need to further describe the connection to the printer.** For example, you may need to enter the baud rate and parity for a serial port, or you might be asked for the network address for an IPP or Samba printer.
4. **Type a Name, Location, and Description for the printer and select if you want to share this printer.** Then click Continue.
5. **Select the make of the print driver.** If you don't see the manufacturer of your printer listed, choose PostScript for a PostScript printer or HP for a PCL printer. For the manufacturer you choose, you will be able to select a specific model.
6. **Set options.** If you are asked to set options for your printer, you may do so. Then select Set Printer Options to continue.
7. **Your printer should be available.** If the printer is added successfully, click the name of your printer to have the new printer page appear; from the printer page, you can select Maintenance or Administration to print a test page or modify the printer configuration.

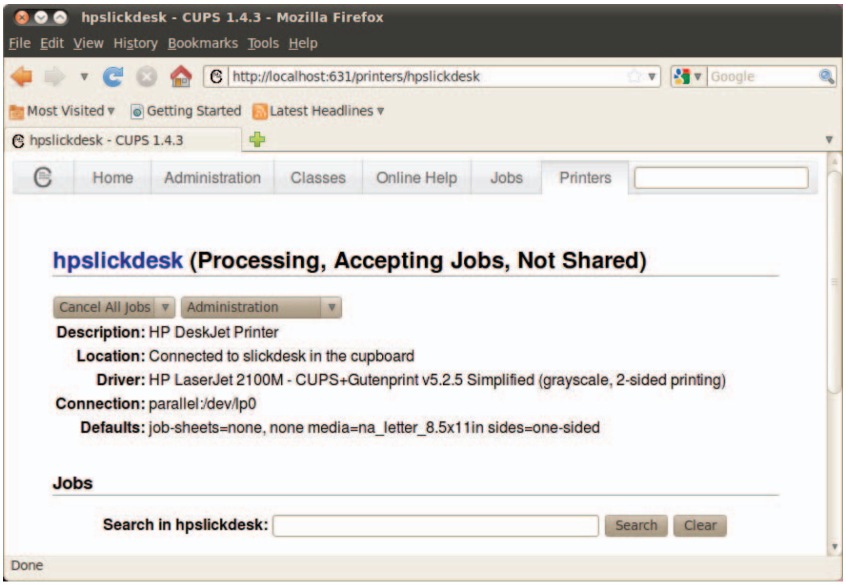
With the basic printer configuration done, you can now do further work with your printers. Here are a few examples of what you can do:

- **List print jobs.** Click Show All Jobs to see what print jobs are currently active from any of the printers configured for this server. Click Show Completed Jobs to see information about jobs that are already printed.
- **Create a printer class.** Click the Administration tab and choose Add Class and identify a name, description, and location for a printer class. From the list of Printers (Members) configured on your server, select the ones to go into this class.

- **Cancel a print job.** If you print a 100-page job by mistake or if the printer is spewing out junk, the Cancel feature can be very handy. From the Administration tab, click Manage Jobs; then click Show Active Jobs to see what print jobs are currently in the queue for the printer. Select the Cancel Job button next to the print job you want to cancel.
- **View printers.** You can click the Printers tab from the top of any of the CUPS web-based administration pages to view the printers you have configured. For each printer that appears, you can select Maintenance or Administrative tasks. Under Maintenance, click Pause Printer (to stop the printer from printing but still accept print jobs for the queue), Reject Jobs (to not accept any further print jobs for the moment), or Print Test Page (to print a page). Figure 16.2 shows the information on the Printers tab.

FIGURE 16.2

You can do administration tasks from the Printers tab.



Using the Printer Configuration window

If you are using Fedora, RHEL, or other Red Hat–based systems, you can use the printer configuration window to set up your printers. In fact, I recommend that you use it instead of CUPS web administration because the resulting printer configuration files are tailored to work with the way the CUPS service is started on those systems.

To install a printer from your GNOME desktop, start the Printer Configuration window by selecting System ➪ Administration ➪ Printing, or as root user by typing

system-config-printer. This tool enables you to add and delete printers and edit printer properties. It also enables you to send test pages to those printers to make sure they are working properly.

The key here is that you are configuring printers that are managed by your print daemon (`cupsd` for the CUPS service). After a printer is configured, users on your local system can use it. You can refer to the section "Configuring Print Servers" to learn how to make the server available to users from other computers on your network.

The printers that you set up can be connected directly to your computer (as on a parallel port) or to another computer on the network (for example, from another UNIX system or Windows system).

Configuring local printers with the Printer Configuration window

Add a local printer (in other words, a printer connected directly to your computer) with the Printer Configuration window using the procedure that follows.

Adding a local printer

To add a local printer from a GNOME desktop in Fedora 16, follow these steps:

1. Select Applications ➪ Other ➪ Printing or type the following as root user from a Terminal window:

   ```
   # system-config-printer &
   ```

   The Printing window appears.
2. Click Add. (If asked, select the button to Adjust Firewall to allow access to the printer port 631.) A New Printer window appears.
3. If the printer you want to configure is detected, simply select it and click Forward. If it is not detected, choose the device to which the printer is connected (`LPT #1` and `Serial Port #1` are the first parallel and serial ports, respectively) and click Forward. (Type **`/usr/sbin/lpinfo -v | less`** in a shell to see printer connection types.) You are asked to identify the printer's driver.
4. To use an installed driver for your printer, choose Select Printer From Database, and then choose the manufacturer of your printer. As an alternative, you could select Provide PPD File and supply your own PPD file (for example, if you have a printer that is not supported in Linux and you have a driver that was supplied with the printer). PPD stands for PostScript Printer Description. Select Forward to see a list of printer models from which you can choose.

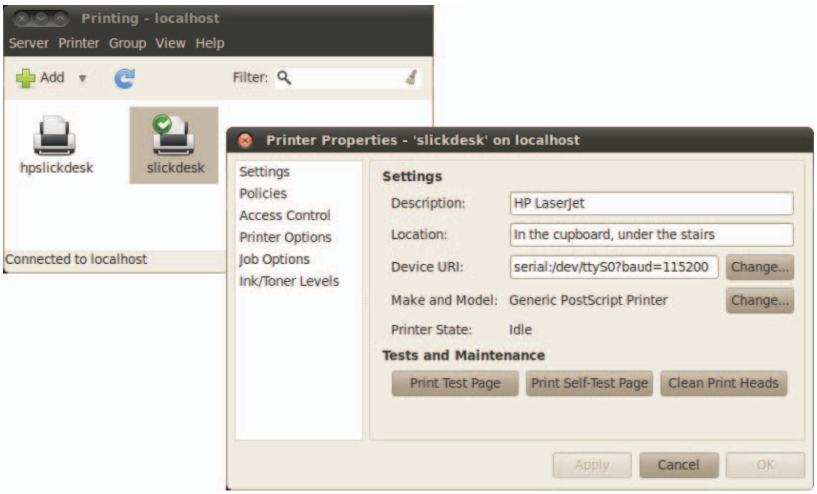
TIP

If your printer doesn't appear on the list but supports PCL (HP's Printer Control Language), try selecting one of the HP printers (such as HP LaserJet). If your printer supports PostScript, select PostScript printer from the list. Selecting Raw Print Queue enables you to send documents that are already formatted for a particular printer type to a specific printer.

5. With your printer model selected, click the driver you want to use with that printer. Click Forward to continue.
6. Add the following information and click Forward:
   - **Printer Name** — Add the name you want to give to identify the printer. The name must begin with a letter, but after the initial letter, it can contain a combination of letters, numbers, dashes (-), and underscores (_). For example, an HP printer on a computer named maple could be named hp-maple.
   - **Description** — Add a few words describing the printer, such as its features (for example, an HP LaserJet 2100M with PCL and PS support).
   - **Location** — Add some words that describe the printer's location (for example, "In Room 205 under the coffeepot").
7. When the printer is added, you may be prompted to print a test page (click No or Yes). The new printer entry appears in the Printing window, as shown in Figure 16-3.

FIGURE 16.3

The Printer Properties window after adding a printer



8. If you want the printer to be your default printer, right-click the printer and select Set As Default. As you add other printers, you can change the default printer by selecting the one you want and Set As Default again.
9. Printing should be working at this point. To make sure, open a Terminal window and use the `lpr` command to print a file (such as `lpr/etc/hosts`). (If you want to share this printer with other computers on your network, refer to the section "Configuring Print Servers," later in this chapter.)

Editing a local printer

After double-clicking the printer you want to configure, choose from the following menu options to change its configuration:

- **Settings** — The Description, Location, Device URI, and Make and Model information you created earlier are displayed in this dialog box.
- **Policies** — Click Policies to set the following items:
  - **State** — Select check boxes to indicate whether the printer will print jobs that are in the queue (Enabled), accept new jobs for printing (Accepting Jobs), or be available to be shared with other computers that can communicate with your computer (Shared). You also must select Server Settings and click the Share Published printers connected to this system check box before the printer will accept print jobs from other computers.
  - **Policies** — In case of error, the stop-printer selection causes all printing to that printer to stop. You can also select to have the job discarded (abort-job) or retried (retry-job) in the event of an error condition.
  - **Banner** — There are no starting or ending banner pages by default for the printer. Choose starting or ending banner pages that include text such as Classified, Confidential, Secret, and so on.
- **Access Control** — If your printer is a shared printer, you can select this window to create a list that either allows users access to the printer (with all others denied) or denies users access to the printer (with all others allowed).
- **Printer Options** — Click Printer Options to set defaults for options related to the printer driver. The available options are different for different printers. Many of these options can be overridden when someone prints a document. Here are a few of the options you might (or might not) have available:
  - **Watermark** — Several Watermark settings are available to enable you to add and change watermarks on your printed pages. By default, Watermark and Overlay are off (None). By selecting Watermark (behind the text) or Overlay (over the text), you can set the other Watermark settings to determine how watermarks and overlays are done. Watermarks can go on every page (All) or only the first page (First Only).

    Select Watermark Text to choose what words are used for the watermark or overlay (Draft, Copy, Confidential, Final, and so on). You can then select the font type, size, style, and intensity of the watermark or overlay.
  - **Resolution Enhancement** — You can use the printer's current settings or choose to turn resolution enhancement on or off.
  - **Page Size** — The default is U.S. letter size, but you can also ask the printer to print legal size, envelopes, ISO A4 standard, or several other page sizes.
  - **Media Source** — Choose which tray to print from. Select Tray 1 to insert pages manually.