Learn Linux, 101: Runlevels, shutdown, and reboot

Bring your system to life

Ian Shields
Senior Programmer
IBM

18 September 2012
(First published 05 January 2011)

Learn to shut down or reboot your Linux system, warn users that the system is going down, and switch to a more or less restrictive runlevel. You can use the material in this article to study for the LPI 101 exam for Linux system administrator certification, or just to learn about shutting down, rebooting, and changing runlevels.
18 Sep 2012 - *This article updated to include material for the LPI Exam 101: Objective Changes as of July 2, 2012.*

Major updates occur in these sections: Beyond Init, Upstart, Systemd, and Resources.

View more content in this series

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009 with minor updates in July 2012) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, though, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

Overview

In this article, learn to shut down or reboot your Linux system, warn users that the system is going down, and switch to single-user mode or a more or less restrictive runlevel. Learn to:

- Set the default runlevel
- Change between runlevels
- Change to single-user mode
- Shut down or reboot the system from the command line
- Alert users about major system events, including switching to another runlevel

- Terminate processes properly

Unless otherwise noted, the examples in this article use a Fedora 8 system with a 2.6.26 kernel. The upstart examples use Fedora 13 with a 2.6.34 kernel, or Ubuntu 10.10 with a 2.6.35 kernel. The systemd examples use Fedora 17 with a 3.4.4 kernel. Your results on other systems may differ.

This article helps you prepare for Objective 101.3 in Topic 101 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 3.

Note: This article includes material for the LPI Exam 101: Objective Changes as of July 2, 2012.

Prerequisites

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which to practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here. In particular, the newer upstart and systemd systems are changing many things that might be familiar to users of the traditional System V init process (see Beyond Init for details). This article is directed primarily at the traditional System V init process, with brief overviews of the how upstart and systemd differ.

Runlevels

Runlevels define what tasks can be accomplished in the current state (or runlevel) of a Linux system. Every Linux system supports three basic runlevels, plus one or more runlevels for normal operation. The basic runlevels are shown in Table 1.

Table 1. Linux basic runlevels

Level	Purpose
0	Shut down (or halt) the system
1	Single-user mode; usually aliased as *s* or *S*
6	Reboot the system

Beyond the basics, runlevel usage differs among distributions. One common usage set is shown in Table 2.

Table 2. Other common Linux runlevels

Level	Purpose
2	Multiuser mode without networking

3	Multiuser mode with networking
5	Multiuser mode with networking and the X Window System

The Slackware distribution uses runlevel 4 instead of 5 for a full system running the X Window system. Debian and derivatives, such as Ubuntu, use a single runlevel for any multiuser mode, typically runlevel 2. Be sure to consult the documentation for your distribution.

Default runlevel

When a Linux system starts, the default runlevel is determined from the `id:` entry in /etc/inittab. Listing 1 illustrates a typical entry for a system such as Fedora 8 or openSUSE 11.2, both of which use runlevel 5 for the X Window System.

Listing 1. Default runlevel in /etc/inittab

```
[root@pinguino ~]# grep "^id:" /etc/inittab
id:5:initdefault:
```

Edit this value if you want your system to start in a different runlevel, say runlevel 3.

Changing runlevels

There are several ways to change runlevels. To make a permanent change, you can edit /etc/inittab and change the default level as you just saw above.

If you only need to bring the system up in a different runlevel for one boot, you can do this. For example, suppose you just installed a new kernel and need to build some kernel modules after the system booted with the new kernel, but before you start the X Window System. You might want to bring up the system in runlevel 3 to accomplish this. You do this at boot time by editing the kernel line (GRUB) or adding a parameter after the selected system name (LILO). Use a single digit to specify the desired runlevel (3, in this case). We'll illustrate the process with a GRUB example. Suppose your /boot/grub/menu.lst file contains the stanza shown in Listing 2.

Listing 2. Typical GRUB stanza to boot Fedora 8

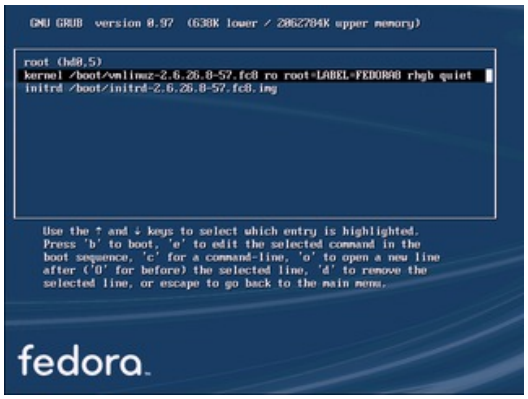
```
title Fedora (2.6.26.8-57.fc8)
        root (hd0,5)
        kernel /boot/vmlinuz-2.6.26.8-57.fc8 ro root=LABEL=FEDORA8 rhgb quiet
        initrd /boot/initrd-2.6.26.8-57.fc8.img
```

To bring this system up in runlevel 3, wait till the boot entries are displayed, select this entry and enter 'e' to edit the entry. Depending on your GRUB options, you may need to press a key to display the boot entries and also enter 'p' and a password to unlock editing. The GRUB screen on our Fedora 8 system looks like Figure 1.

Figure 1. Selecting a boot choice in GRUB

In this example, you should now see the root, kernel, and initrd lines displayed. Move the cursor to the line starting with "kernel" and press 'e' to edit the line. The GRUB screen on our Fedora 8 system now looks like Figure 2.

Figure 2. Selecting the kernel entry for editing



Finally, move the cursor to the end of the line, and add a space and the digit '3'. You may remove 'quiet' if you wish, or modify any other parameters as needed. The GRUB screen on our Fedora 8 system now looks like Figure 3.

Figure 3. Setting the starting runlevel to 3

Finally, press **Enter** to save the changes, then type 'b' to boot the system.

Note: The steps for doing this using LILO or GRUB2 differ from those for GRUB, but the basic principle of editing the way the kernel is started remains. Even GRUB screens on other systems or other distributions may look quite different to those shown here. Prompts will usually be available to help you.

Once you have finished your setup work in runlevel 3, you will probably want to switch to runlevel 5. Fortunately, you do not need to reboot the system. You can use the `telinit` command to switch to another runlevel. Use the `runlevel` command to show both the previous runlevel and the current one. If the first output character is 'N', the runlevel has not been changed since the system was booted. Listing 3 illustrates verifying and changing the runlevel.

Listing 3. Verifying and changing the runlevel

```
[root@pinguino ~]# runlevel
N 3
[root@pinguino ~]# telinit 5
```

After you enter `telinit 5` you will see several messages flash by and your display will switch to the configured graphical login screen. Open a terminal window and verify that the runlevel has been changed as shown in Listing 4.

Listing 4. Confirming the new runlevel

```
[root@pinguino ~]# runlevel
3 5
```

If you use the `ls` command to display a long listing of the `telinit` command, you will see that it really is a symbolic link to the `init` command. We illustrate this in Listing 5

Listing 5. telinit is really a symbolic link to init

```
[root@pinguino ~]# ls -l $(which telinit)
lrwxrwxrwx 1 root root 4 2008-04-01 07:50 /sbin/telinit -> init
```

The `init` executable knows whether it was called as `init` or `telinit` and behaves accordingly. Since `init` runs as PID 1 at boot time, it is also smart enough to know when you subsequently invoke it using `init` rather than `telinit`. If you do, it will assume you want it to behave as if you had called `telinit` instead. For example, you may use `init 5` instead of `telinit 5` to switch to runlevel 5.

Single-user mode

In contrast to personal computer operating systems such as DOS or Windows, Linux is inherently a multiuser system. However, there are times when that can be a problem, such as when you need to recover a major filesystem or database, or install and test some new hardware. Runlevel 1, or *single-user mode*, is your answer for these situations. The actual implementation varies by distribution, but you will usually start in a shell with only a minimal system. Usually there will be no networking and no (or very few) daemons running. On some systems, you must authenticate by logging in, but on others you go straight into a shell prompt as root. Single-user mode can be

a lifesaver, but you can also destroy your system, so always be careful whenever you are running with root authority. Reboot to normal multiuser mode as soon as you are done.

As with switching to regular multiuser runlevels, you can also switch to single-user mode using `telinit 1`. As noted in Table 1, 's' and 'S' are aliases for runlevel 1, so you could, for example, use `telinit s` instead.

Clean shutdown

While you can use `telinit` or `init` to stop multiuser activity and switch to single-user mode, this can be rather abrupt and cause users to lose work and processes to terminate abnormally. The preferred method to shut down or reboot the system is to use the `shutdown` command, which first sends a warning message to all logged-in users and blocks any further logins. It then signals `init` to switch runlevels. The `init` process then sends all running processes a SIGTERM signal, giving them a chance to save data or otherwise properly terminate. After 5 seconds, or another delay if specified, `init` sends a SIGKILL signal to forcibly end each remaining process.

By default, `shutdown` switches to runlevel 1 (single-user mode). You may specify the `-h` option to halt the system, or the `-r` option to reboot. A standard message is issued in addition to any message you specify. The time may be specified as an absolute time in *hh:mm* format, or as a relative time, *n*, where n is the number of minutes until shutdown. For immediate shutdown, use *now*, which is equivalent to *+0*.

If you have issued a delayed shutdown and the time has not yet expired, you may cancel the shutdown by pressing **Ctrl-c** if the command is running in the foreground, or by issuing `shutdown` with the `-c` option to cancel a pending shutdown. Listing 6 shows several examples of the use of `shutdown`, along with ways to cancel the command.

Listing 6. Shutdown examples

```
[root@pinguino ~]# shutdown 5 File system recovery needed

Broadcast message from root (pts/1) (Tue Jan  4 08:05:24 2011):

File system recovery needed
The system is going DOWN to maintenance mode in 5 minutes!
^C
Shutdown cancelled.
[root@pinguino ~]# shutdown -r 10 Reloading updated kernel&
[1] 18784
[root@pinguino ~]#
Broadcast message from root (pts/1) (Tue Jan  4 08:05:53 2011):

Reloading updated kernel
The system is going DOWN for reboot in 10 minutes!

[root@pinguino ~]# fg
shutdown -r 10 Reloading updated kernel
^C
Shutdown cancelled.
[root@pinguino ~]# shutdown -h 23:59&
[1] 18788
[root@pinguino ~]# shutdown -c

Shutdown cancelled.
```

```
[1]+  Done                    shutdown -h 23:59
```

You may have noticed that our last example did not cause a warning message to be sent. If the time till shutdown exceeds 15 minutes, then the message is not sent until 15 minutes before the event as shown in Listing 7. Listing 7 also shows the use of the `-t` option to increase the default delay between SIGTERM and SIGKILL signals from 5 seconds to 60 seconds.

Listing 7. Another shutdown example

```
[root@pinguino ~]# date;shutdown -t60 17 Time to do backups&
Tue Jan  4 08:12:55 EST 2011
[1] 18825
[root@pinguino ~]# date
Tue Jan  4 08:14:13 EST 2011
[root@pinguino ~]#
Broadcast message from root (pts/1) (Tue Jan  4 08:14:55 2011):

Time to do backups
The system is going DOWN to maintenance mode in 15 minutes!
```

If you do cancel a shutdown, you should use the `wall` command to send a warning to all users alerting them to the fact that the system is **not** going down.

As we said earlier, it is also possible to use `telinit` (or `init`) to shut down or reboot the system. As with other uses of `telinit`, no warning is sent to users, and the command takes effect immediately, although there is still a delay between SIGTERM and SIGKILL signals. For additional options of `telinit`, `init`, and `shutdown`, consult the appropriate man pages.

Halt, reboot, and poweroff

You should know about a few more commands related to shutdown and reboot.

- The `halt`command halts the system.
- The `poweroff` command is a symbolic link to the `halt` command, which halts the system and then attempts to power it off.
- The `reboot`command is another symbolic link to the `halt` command, which halts the system and then reboots it.

If any of these are called when the system is not in runlevel 0 or 6, then the corresponding `shutdown` command will be invoked instead.

For additional options that you may use with these commands, as well as more detailed information on their operation, consult the man page.

/etc/inittab

By now, you may be wondering why pressing **Ctrl-Alt-Delete** on some systems causes a reboot, or how all this runlevel stuff is configured. Remember the `id` field in /etc/inittab? Well, there are several other fields in /etc/inittab, along with a set of init scripts in directories such as rc1.d or rc5.d, where the digit identifies the runlevel to which the scripts in that directory apply. Listing 8 shows the full inittab from our Fedora 8 system.

Listing 8. Full inittab from Fedora 8

```
#
# 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
```

As usual, lines starting with # are comments. Other lines have several fields with the following format:
`id:runlevels:action:process`

id

is a unique identifier of one to four characters. Older versions limited this to two characters, so you will often see only two characters used.

runlevels
: lists the runlevels for which the action for this id should be taken. If no runlevels are listed, do the action for all runlevels.

action
: describes which of several possible actions should be taken

process
: tells which process, if any, should be run when the action on this line is performed.

Some of the common actions that may be specified in /etc/inittab are shown in Table 3. See the man pages for inittab for other possibilities.

Table 3. Some common inittab actions

Action	Purpose
respawn	Restart the process whenever it terminates. Usually used for getty processes, which monitor for logins.
wait	Start the process once when the specified runlevel is entered and wait for its termination before init proceeds.
once	Start the process once when the specified runlevel is entered.
initdefault	Specifies the runlevel to enter after system boot.
ctrlaltdel	Execute the associated process when init receives the SIGINT signal, for example, when someone on the system console presses CTRL-ALT-DEL.

Listing 9 shows just the entry for **Ctrl-Alt-Delete** from Listing 8. So now you see why pressing Ctrl-Alt-Delete causes the system to be rebooted.

Listing 9. Trapping Ctrl-Alt-Delete

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

Initialization scripts

You may have noticed several lines in Listing 8, such as

```
l5:5:wait:/etc/rc.d/rc 5
```

In this example, `init` will run the `/etc/rc.d/rc` script (or command) with the parameter of `5` whenever runlevel 5 is entered. `init` will wait until this command completes before doing anything else.

These scripts used by `init` when starting the system, changing runlevels, or shutting down are typically stored in the /etc/init.d or /etc/rc.d directory. A series of symbolic links in the rc*n*.d

directories, one directory for each runlevel *n*, control whether a script is started when entering a runlevel or stopped when leaving it. These links start with either a K or an S, followed by a two-digit number and then the name of the service, as shown in Listing 10.

Listing 10. Init scripts

```
[root@pinguino ~]# find /etc -path "*rc[0-9]*.d/???au*"
/etc/rc.d/rc2.d/S27auditd
/etc/rc.d/rc2.d/K72autofs
/etc/rc.d/rc4.d/S27auditd
/etc/rc.d/rc4.d/S28autofs
/etc/rc.d/rc5.d/S27auditd
/etc/rc.d/rc5.d/S28autofs
/etc/rc.d/rc0.d/K72autofs
/etc/rc.d/rc0.d/K73auditd
/etc/rc.d/rc6.d/K72autofs
/etc/rc.d/rc6.d/K73auditd
/etc/rc.d/rc1.d/K72autofs
/etc/rc.d/rc1.d/K73auditd
/etc/rc.d/rc3.d/S27auditd
/etc/rc.d/rc3.d/S28autofs
[root@pinguino ~]# cd /etc/rc.d/rc5.d
[root@pinguino rc5.d]# ls -l ???a*
lrwxrwxrwx 1 root root 16 2008-04-07 11:29 S27auditd -> ../init.d/auditd
lrwxrwxrwx 1 root root 16 2008-04-01 07:51 S28autofs -> ../init.d/autofs
lrwxrwxrwx 1 root root 15 2008-04-01 14:03 S44acpid -> ../init.d/acpid
lrwxrwxrwx 1 root root 13 2008-04-01 07:50 S95atd -> ../init.d/atd
lrwxrwxrwx 1 root root 22 2008-04-01 07:54 S96avahi-daemon -> ../init.d/avahi-daemon
lrwxrwxrwx 1 root root 17 2008-11-17 13:40 S99anacron -> ../init.d/anacron
```

Here you see that the `audit` and `autofs` services have K*nn* entries in all runlevels and S*nn* entries for both in runlevels 3 and 5. The S indicates that the service is started when that runlevel is entered, while the K entry indicates that it should be stopped. The *nn* component of the link name indicates the priority order in which the service should be started or stopped. In this example, `audit` is started before `autofs`, and it is stopped later.

Consult the man pages for `init` and `inittab` for more information.

Beyond Init

As we have seen here, the traditional method of booting a Linux system is based on the UNIX System V init process. It involves loading an initial RAM disk (initrd) and then passing control to a program called `init`, a program that is usually installed as part of the sysvinit package. The `init` program is the first process in the system and has PID (Process ID) 1. It runs a series of scripts in a predefined order to bring up the system. If something that is expected is not available, the init process typically waits until it is. While this worked adequately for systems where everything is known and connected when the system starts, modern systems with hot-pluggable devices, network file systems, and even network interfaces that may not be available at start time present new challenges. Certainly, waiting for hardware that may not come available for a long time, or even just a relatively long time, is not desirable.

In the following sections of this article we will describe two alternatives to System V init, *upstart* and *systemd*.

Upstart

A new initialization process called *upstart* was first introduced in Ubuntu 6.10 ("Edgy Eft") in 2006. Fedora 9 through 14 and Red Hat Enterprise Linux (RHEL) 6 use upstart, as do distributions derived from these. Upstart has now supplanted the init process in Ubuntu among others, although vestiges of init remain and the full power of upstart may not be realized for some time yet.

In contrast to the static set of init scripts used in earlier systems, the upstart system is driven by *events*. Events may be triggered by hardware changes, starting or stopping or tasks, or by any other process on the system. Events are used to trigger *tasks* or *services*, collectively known as *jobs*. So, for example, connecting a USB drive might cause the udev service to send a *block-device-added* event, which would cause a defined task to check /etc/fstab and mount the drive if appropriate. As another example, an Apache web server may be started only when both a network and required filesystem resources are available.

The upstart initialization program replaces /sbin/init. Upstart jobs are defined in the /etc/init directory and its subdirectories. The upstart system will currently process /etc/inittab and System V init scripts. On systems such as recent Fedora releases, /etc/inittab is likely to contain only the id entry for the initdefault action. Recent Ubuntu systems do not have /etc/inittab by default, although you can create one if you want to specify a default runlevel.

Upstart also has the initctl command to allow interaction with the upstart init daemon. This allows you to start or stop jobs, list jobs, get status of jobs, emit events, restart the init process, and so on. Listing 11 shows how to use `initctl` to obtain a list of upstart jobs on a Fedora 13 system.

Listing 11. Interacting with upstart init daemon using initctl

```
[ian@echidna ~]$ initctl list
rc stop/waiting
tty (/dev/tty3) start/running, process 1486
tty (/dev/tty2) start/running, process 1484
tty (/dev/tty6) start/running, process 1492
tty (/dev/tty5) start/running, process 1490
tty (/dev/tty4) start/running, process 1488
plymouth-shutdown stop/waiting
control-alt-delete stop/waiting
system-setup-keyboard start/running, process 1000
readahead-collector stop/waiting
vpnc-cleanup stop/waiting
quit-plymouth stop/waiting
rcS stop/waiting
prefdm start/running, process 1479
init-system-dbus stop/waiting
ck-log-system-restart stop/waiting
readahead stop/waiting
ck-log-system-start stop/waiting
start-ttys stop/waiting
readahead-disable-services stop/waiting
ck-log-system-stop stop/waiting
rcS-sulogin stop/waiting
serial stop/waiting
```

To learn more about upstart, see Resources.

Systemd

Another new initialization system called *systemd* is also emerging. Systemd was developed by Lennart Poettering in early 2010. He described the rationale and design in a blog post (see Resources. It has been adopted for Fedora 15, openSUSE 12.1 and Mandriva 2011 among others.

Many daemon processes communicate using sockets. In order to gain speed and enhance parallelism in the system startup, systemd creates these sockets at startup, but only starts the associated task when a connection request for services on that socket is received. In this way, services can be started only when they are first required and not necessarily at system initialization. Services that need some other facility will block until it is available, so only those services that are waiting for some other process need block while that process starts.

Extending the idea of waiting for services systemd uses autofs to define mount points, so the mount point for a file system is available, but the actual mount may be delayed until some process attempts to open a file on the file system or otherwise use it.

These ideas not only delay startup of services until needed, they also reduce the need for dependency checking between services, as the interface for the service can be ready long before the service itself needs to be available.

Like upstart, systemd can process existing initialization from /etc/inittab. It can also process /etc/fstab to control file system mounting. Native systemd initialization revolves around the concept of *units*, which can be grouped into *control groups* or *cgroups*.

- Service units are daemons that can be started, stopped, restarted, reloaded.
- Socket units encapsulate a socket in the file-system or on the Internet.
- Device units encapsulate a device in the Linux device tree.
- Mount units encapsulate a mount point in the file system hierarchy.
- Automount units encapsulate an automount point in the file system hierarchy.
- Target units group other units together, providing a single control unit for multiple other units.
- Snapshot units reference other units and can be used to save and roll back the state of all services and units of the init system, for example during suspend.

Units are configured using a configuration file which includes the unit type as a suffix. For example, cups.service, rpcbind.socket or getty.target. The location of system configuration files, for example /etc/systemd/system can be determined using the `pkg-config` command as shown in Listing 11 which shows the location on a Fedora 17 system. Systemd also checks /usr/local/lib/systemd/system and /usr/lib/systemd/system for configuration information.

Listing 12. Locating the systemd system configuration directory

```
[ian@attic4 ~]$ pkg-config systemd --variable=systemdsystemconfdir
/etc/systemd/system
```

The `systemctl` command allows you to interrogate and control the systemd daemon, including starting and stopping units or listing their status. Listing 13 illustrates the use of `systemctl` to display the status of systemd units.

Listing 13. Partial output from systemctl

```
[ian@attic4 ~]$ systemctl --no-pager
UNIT                      LOAD   ACTIVE SUB       JOB DESCRIPTION
proc-sys...misc.automount loaded active running       Arbitrary Executable File
sys-devi...et-eth0.device loaded active plugged       RTL8111/8168B PCI Express
sys-devi...da-sda1.device loaded active plugged       WDC_WD6401AALS-00L3B2
sys-devi...a-sda10.device loaded active plugged       WDC_WD6401AALS-00L3B2
sys-devi...a-sda11.device loaded active plugged       WDC_WD6401AALS-00L3B2
sys-devi...a-sda12.device loaded active plugged       WDC_WD6401AALS-00L3B2
sys-devi...da-sda2.device loaded active plugged       WDC_WD6401AALS-00L3B2
...
systemd-...ssions.service loaded active exited        Permit User Sessions
systemd-...-setup.service loaded active exited        Setup Virtual Console
tcsd.service              loaded failed failed        LSB: Init script for TCSD
udev-settle.service       loaded active exited        udev Wait for Complete Dev
udev-trigger.service      loaded active exited        udev Coldplug all Devices
udev.service              loaded active running       udev Kernel Device Manager
udisks2.service           loaded active running       Storage Daemon
upower.service            loaded active running       Daemon for power managemen
avahi-daemon.socket       loaded active listening     Avahi mDNS/DNS-SD Stack Ac
cups.socket               loaded active running       CUPS Printing Service Sock
...
syslog.target             loaded active active        Syslog
systemd-...ted-ntp.target loaded active active        Network Time Protocol
systemd-...ead-done.timer loaded active elapsed       Stop Read-Ahead Data Colle
systemd-...es-clean.timer loaded active waiting       Daily Cleanup of Temporary

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
JOB    = Pending job for the unit.

132 units listed. Pass --all to see inactive units, too.
```

To learn more about systemd, see Resources.

This completes your introduction to Linux runlevels, shutdown, and reboot.

Resources

Learn

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives. Note the Exam 101: Objective Changes as of July 2, 2012.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- The Linux BootPrompt-HowTo helps you understand boot parameters.
- See the Upstart overview for more information on upstart.
- See Rethinking PID 1 for more information on the rationale for systemd.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. You can contact Ian at ishields@us.ibm.com.