Insider Tips: The Sys V Init Process

Starting Lineup

On booting, a typical Linux system launches numerous tools that configure the system and initialize services, which then carry on running in the background. Init is the parent of all these processes. Its structure is similar across most Linux distributions.

BY MARC ANDRÉ SELIG

Lada Formanek, visipix.com

If you type *ps ax* immediately after booting a Unix computer, you will be confronted with a mass of active programs. From arcane kernel processes through numerous daemons, whose names typically end with a *d*, to *getty* processes for the virtual consoles, and *xdm*, *gdm*, or *kdm* for the GUI-based login – a wonderful, wacky world.

Maybe you've been wondering who told this horde of processes to launch on your machine, who coordinates and monitors them. On booting, the kernel calls just one program which in turn initializes the system. The program is aptly called *init* and it handles the launching of the other programs with only one or two exceptions.

The system administrator, or the manufacturer of your Linux distro (if you leave the defaults unchanged) decides which programs the init process should launch, and under what circumstances. There is no real reason to replace the *init* program, as it offers lots of scope for configurations.

One for All

Traditionally, Linux uses two different init variants. The first is called simpleinit and is simple to use, as the name would suggest. What simpleinit does is to parse a long script file. The script launches all the programs required to run the system. If needed, the script can also parse configuration files, or even create them, before going on to launch the required daemons.

Although simpleinit's ease of use and elegance are convincing arguments, it is too restricting for most Linux distributions. It stores the full range of system status information in a single file. This makes life difficult for today's package-oriented distributions. Each time a user installs, or removes, a package, the package manager has to edit the central script file. While doing so, it has to respect the changes made by the user, if any – a task which is prone to error.

This has led most modern distributions to adopt the Sys V init model. This init process is modeled on Unix System V. It uses a fairly complex hierarchy of configuration and script files, along with symbolic links. This allows init to map various system states with respect to the software available for the user to install, and to the system's operating modes.

Little Helpers

A Sys V Init differentiates between single-user mode, which is used for system maintenance, a stand-alone workstation mode where the network connection is disabled, and a normal mode with the full range of services. Most systems also differentiate between GUI and non-GUI-based logins, which allows them to run on partially configured new installations, or with an incorrect graphics configuration.

These different system states are known as runlevels. Each runlevel is indicated by a single-figure number or letter. There is a widespread convention that uses runlevel 0 for shutting down the computer, and runlevel 6 for a reboot. Runlevel 1 (also known as s or S) enables single-user mode with networking and user logins either restricted or disabled. This allows the root user exclusive access to the system for maintenance work. Runlevels 2 through 5 are configurable, and distributions use them differently. As a rule of thumb, the higher the runlevel, the more features it should have.

As Sys V Init is so widespread, we will be focusing on it in this article.

Central Configuration

The first configuration file for *init* is called */etc/inittab*. The program parses the file immediately after booting the system, and the contents of the file

determine what happens next. The major components of inittab are shown in Listing 1.

Each line in the configuration file contains four colon-separated elements. This principle is common with traditional configuration files. For example, you will find that passwd or the traditional printcap printer database both use this approach.

The first column in an inittab line has a short mnemonic for the program with up to four characters, although one or two characters are more common. Admins are free to enter whatever they like for this column, but make sure your choice is unique. The second column indicates in which runlevels this service will be enabled. In our example, lines 12 through 18 execute exactly one *rc* script with appropriate parameters for each of the runlevels, 0 through 6. The shutdown command in line 21 is available in runlevels 1 through 5.

The third entry is for an action that describes how and when the init process should run the command in the line. The action is typically *wait* or *respawn*. *wait* tells init to run the program in column four once only, whereas *respawn* tells init to keep on launching the program whenever it terminates. Wait is typically used to initialize the system whereas respawn entries are for services that need to be running. Even if a service crashes, the system should be able to relaunch it.

The inittab in Listing 1 shows a few special actions. Line 2 sets the default runlevel (*initdefault*); this is the runlevel the system will enter on booting. The *sysinit* entry in line 6 runs init first, then the entries with *boot* (simultaneously) and *bootwait* (in sequence). There are a few more actions for special cases that allow the system to react to signals from an uninterrupted power supply, for example.

Listing 1: Init Control File

```
01 # Default runlevel
02 id:2:initdefault:
03
04 # Script to run at system boot
05 # initialization and
   configuration
06 si::sysinit:/etc/init.d/rcS
07
08 # Single-user mode
09 ~~:S:wait:/sbin/sulogin
10
11 # The rc scripts for the
   runlevels
12 l0:0:wait:/etc/init.d/rc 0
13 l1:1:wait:/etc/init.d/rc 1
14 l2:2:wait:/etc/init.d/rc 2
15 l3:3:wait:/etc/init.d/rc 3
16 l4:4:wait:/etc/init.d/rc 4
17 l5:5:wait:/etc/init.d/rc 5
18 l6:6:wait:/etc/init.d/rc 6
19
20 # React to
   [Ctrl]+[Alt]+[Delete]
21 ca:12345:ctrlaltdel:/sbin/
   shutdown -t1 -a -h now
22
23 # Getty processes
24 1:2345:respawn:/sbin/getty
   38400 tty1
25 2:23:respawn:/sbin/getty 38400
   tty2
26 3:23:respawn:/sbin/getty 38400
   tty3
27 4:23:respawn:/sbin/getty 38400
   tty4
28 5:23:respawn:/sbin/getty 38400
   tty5
29 6:23:respawn:/sbin/getty 38400
   tty6
```

Init scripts

A quick investigation of Listing 1 tells us that the init process does not have a lot to do. It parses *inittab*, discovers the default runlevel (2) in line 2, launches an initialization script, *rcS*, and then another called *rc 2*, followed by a few *getty* processes (in lines 24 through 29). The latter display the login prompts on virtual consoles *tty1* through *tty6*. Incidentally, the *rc* and *rcS* scripts are referred to as sequencer scripts on HP-UX. There is a good reason for this, as they call the start scripts in the right sequence.

Listing 2: A minimalistic init script

```
01 #! /bin/sh
02
03 test -x /usr/sbin/sshd || exit
   0
04
05 set -e
06 case "$1" in
07   start)
08     echo -n "Starting sshd"
09     /usr/sbin/sshd
10     echo "."
11     ;;
12   stop)
13     echo -n "Stopping sshd"
14     kill `cat
   /var/run/sshd.pid`
15     echo "."
16     ;;
17 esac
18
19 exit 0
```

The rc scripts do most of the work. The script names are configurable, and vary depending on your distribution. This said, they all do more or less the same thing: they launch the start scripts in a clearly defined order. The details are stored in specific directories. The *rcS* searches */etc/rc.S*, and *rc 2* searches */etc/rc2.d*, as you might have guessed. These directories typically point to scripts (mostly using symlinks or hardlinks) with more human sounding names. Figure 1 shows one of these directories on a Debian system.

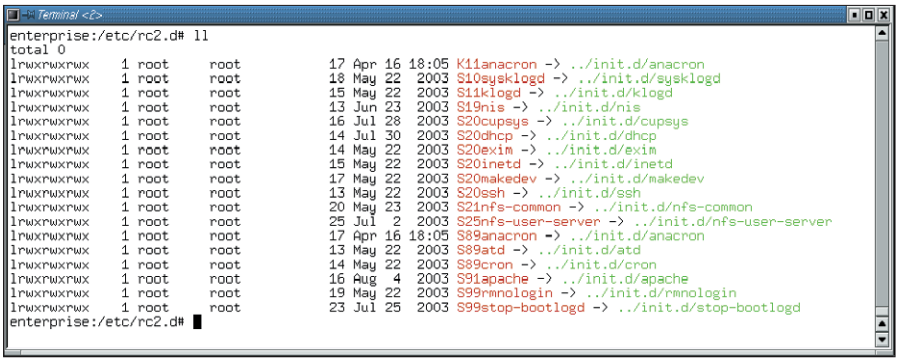

Figure 1: The */etc/rc2.d* directory on a Debian machine contains symbolic links to the start scripts for runlevel 2. the link name specifies which script *rc* should call when, and with what options.

The first letter in the script or link name indicates whether to start, *S*, or kill, *K*, the system service it refers to. In our example, init will be stopping *anacron*, and starting all the other services, including a new anacron instance. The first letters are followed by two numbers that indicate the priority. Scripts with lower priority are run before scripts with higher priority. The numbers simply indicate a hacking order.

When switching to runlevel 2, the machine will first kill the anacron process, if it is running. Having done so, it calls *sysklogd*, then *klogd*, and *nis*. It is a good idea to launch the syslog daemon right at the outset. This allows you to capture any error messages. The klog daemon needs syslog running, and must be launched after syslog. Complex services, such as Apache, are best left right to the end, when the rest of the system is up and running.

Init Scripts Under the Hood

Init scripts also follow a convention. Each script needs to understand at least two parameters: *start* and *stop*. If the init script is linked to a name that starts with *S* in the runlevel directory (such as *S10sysklogd* in our example), the rc script for the runlevel will call it with the *start* parameter. If the name starts with a *K* instead (as in *K11anacron*), the parameter is *stop* instead.

And now for Something Completely Different

This article describes the basic runlevel configuration steps for a Unix system. No matter whether you have Linux, a BSD variant, Solaris, HP-UX, or some other Unix derivative, Sys V Init will work in the same, or a similar way.

Unfortunately, some Linux distributions do not respect this convention, Suse for example. A member of Suse's staff warned us that Suse can interrupt the configuration steps, and reset or modify some settings.

If you need to create an init script on a Suse system, you should keep to the guidelines in */etc/init.d/skeleton*. We will be looking at the characteristics, and quirks, of this major distro in another article.

Almost all init scripts are recycled. The same script that launches a service at boot time, can ensure that the service terminates gracefully at shutdown. That helps admins to keep things tidy. If a daemon is moved to another location, the admin needs to edit the start and stop commands. This is why the init scripts are normally grouped in a directory of their own, typically */etc/init.d*. The runlevel directories simply contain links that point to the originals.

Listing 2 shows an init script that has been reduced to the max. It starts by checking the availability of the daemon; it might have been deinstalled in the meantime. It then goes on to call the daemon, or kill it, depending on the parameter. To kill the daemon, the script sends a *TERM* signal to the process ID that the daemon stores in */var/run*.

When you install an additional service, Sys V Init removes the need for the package manager to change the existing files. Instead, it simply copies the start script to the target directory and creates symlinks to match. ■