Arranque y detención de sistemas GNU/Linux

Manolo Padrón Martínez
manolopm@softhome.net

Imobach González Sosa
imobachgs@softhome.net

16 de marzo de 2004

Licencia

Índice

1. Acerca de este documento

Cualquier administrador que se precie ha de conocer cómo funcionan los sistemas que se encuentran a su cargo. Con esta premisa, parece lógico que uno de los primeros aspectos que tenga que dominar sea su puesta en marcha y su detención.

Ambas fases tienen una incidencia importante tanto en el funcionamiento como en el rendimiento: en cuanto al inicio, parece obvio que si éste no se lleva a cabo de forma correcta, el administrador se verá obligado a corregir algunos aspectos cuando el sistema esté en producción; en cuanto a la detención, hay que resaltar que llevar a cabo este proceso de forma incorrecta puede causar múltiples problemas que pueden dejar secuelas.

Este documento está destinado a presentar los fundamentos del arranque y detención de los sistemas *GNU/Linux* siempre *desde el punto de vista del administrador*. No se trata de una referencia exhaustiva de herramientas o procedimientos, sino simplemente de una guía acerca de los conceptos fundamentales que un administrador de un sistema de esta clase debe ser capaz de controlar.

El texto está orientado a la distribución *Red Hat Linux 9* y, si bien muchos de los conceptos presentados son aplicables a otras distribuciones, existen algunas diferencias. En la medida de lo posible, se indicarán las más importantes, con el fin de no perder la visión general que se pretende obtener.

Las páginas del manual y la documentación en línea constituyen un buen complemento para esta lectura. Además, en Internet puede encontrarse abundante información. Al final del documento se encuentra una pequeña lista de fuentes de información utilizadas en su elaboración.

2. El proceso de arranque

2.1. Visión general

Los sistemas *GNU/Linux* han tomado de los sistemas UNIX gran parte de los conceptos que aplican en el proceso de arranque. De ellos han heredado la potencia, flexibilidad y transparencia de la que estos sistemas disfrutan. Sin embargo, pese a sus evidentes ventajas, no se trata de procesos complejos y difíciles de asimilar. Al contrario, disfrutan de una sencillez envidiable, lo que permite al administrador manipularlos a su antojo adaptándolos a sus necesidades fácilmente.

En líneas generales, el proceso de arranque de un sistema Linux transcurre por las siguientes fases:

1. El *firmware*[1] cede el control al gestor de arranque.
2. El gestor de arranque carga en memoria el núcleo.
3. El núcleo se descomprime a sí mismo, lleva a cabo un proceso de inicialización de los dispositivos, monta el disco RAM inicial o, en su defecto, monta la partición raíz en modo de solo lectura y, finalmente, invoca a `init`.
4. `init`, el antecesor de todos los procesos, llevará a cabo la configuración del software: montará —de nuevo— el sistema de ficheros raíz en modo lectura/escritura, inicializa algunos subsistemas, carga configuraciones y decide que servicios arrancar.

Durante el resto de esta sección, se profundizará en todas y cada una de las fases del arranque, sin perder el enfoque que debería darle el administrador del sistema. Es decir, no se hablará en profundidad de la misión del núcleo en el arranque, pero sí se hará hincapié en conceptos como el de *nivel de ejecución*.

2.2. Gestores de arranque

2.2.1. ¿Qué es un gestor de arranque?

Un gestor de arranque es un pequeño programa que se encarga de colocar el núcleo de linux en la memoria para que comience la ejecución. Con la evolución de los gestores

[1] La BIOS en el caso de plataformas x86-32.

de arranque se ha conseguido que inicialice otros sistemas operativos que se encuentren instalados en el sistema e incluso arrancar desde unidades de disquete o CD-ROM.

Generalmente estos programas se encuentran divididos en fases, ya que al arrancar el equipo simplemente lee el MBR[2] del disco y esto limitaría mucho el tamaño del programa.

La división permite evitar este problema, haciendo que una primera parte muy pequeña se encuentre en el MBR y que ésta sea la encargada de cargar el resto.

Actualmente existen muchos gestores de arranque, dependiendo de la arquitectura; por ejemplo, en máquinas con procesadores *Alpha* o antiguos *PowerPC* se usa `aboot`; en los nuevos *PowerPC*, `yaboot`; en máquinas basadas en procesadores de x86-64, `Elilo`; y en arquitecturas `x86-32` `LILO` o `GRUB`.

A continuación se realizará una presentación de los dos gestores de arranque más utilizados en arquitecturas x86: `LILO` y `Grub`.

2.2.2. LILO: LInux LOader

Las fases de arranque del LILO

El proceso de arranque del LILO se divide en dos fases. La primera de ellas ocupa un sólo sector de disco que es cargado en memoria por la BIOS. Esta primera fase se ocupa únicamente de cargar la segunda etapa de LILO en memoria. Esta forma de trabajar es debido a la limitación de las BIOS en las arquitecturas x86, que sólo permiten cargar un sector inicial del disco en memoria.

La segunda fase ya es un poco más compleja. Llegados a este punto, desaparece la limitación de ocupar un sólo sector de disco. Una vez esta fase esté cargada en memoria, se encargará de mostrar el *prompt* , procesar la entrada y o bien cargar la imagen asociada a la selección, o bien transferir el control a otra partición o disco para que realice el arranque.

Instalación de LILO

LILO es un programa relativamente fácil de instalar. Una vez se tiene un fichero de configuración preparado, es tan simple como lanzar `lilo` y él se encargará de configurar el sistema en base a dicho fichero.

LILO se puede instalar en dos partes de un disco duro según las necesidades, el sector

[2]Master Boot Record

de arranque de una partición o el MBR de un disco. Estando en el sector de arranque de una de una partición, ésta debe estar definida como la "partición activa" de la unidad. Así se conseguirá que, una vez cargado el MBR del disco, este salte dicha partición y cargue en memoria la primera parte del LILO.

La otra modalidad consiste en que el LILO se apropie del MBR. Esto es especialmente útil si se usan varios sistemas operativos, ya que algunos de ellos requieren que su partición sea la única "partición activa" del disco para poder arrancar.

Arranque de LILO

Cuando LILO arranca muestra `LILO:` en pantalla y espera a que el usuario introduzca la etiqueta de la imagen que se desea arrancar. Si no se ha especificado la opción `prompt` en el fichero de configuración, intentará arrancar directamente. Éste comportamiento se evita presionando `alt`, `ctrl` o `shift` o dejando encendidos `Capslock` o `ScrollLock`, consiguiendo que LILO muestre el *prompt* para indicarle que imagen debe arrancar. En este punto se puede presionar `?` o `tab` para obtener una lista de imágenes disponibles. Pulsando *enter* se cargará en memoria la imagen por defecto.

LILO permite pasarle parámetros al núcleo de linux y a `init`. Entre éstas destacan:

single Permite entrar al sistema en modo monousuario.

Nivel de ejecución Indica al `init` en qué nivel de ejecución debe entrar. Se puede usar la palabra reservada `single` para referirse al *modo monousuario* (ver 2.4.1, 10).

root=partición Permite especificar donde se encuentra la partición que sera montada como partición raíz.

init=programa Indica el programa a arrancar en vez del init.

noinitrd: Desactiva la carga automática del disco ram inicial.

vga=modo: Indica el modo de texto a cargar como se detalla mas adelante.

Configuración de LILO

El fichero de configuración de LILO, `/etc/lilo.conf`, se divide en una sección global y varias secciones correspondientes a las diferentes imágenes que se desee arrancar.

El fichero de configuración básico solamente usara las cláusulas `boot`, `root`, `image` y `label`.

La cláusula `boot` se usa para indicar cuál es el disco de arranque; `root` hace referencia a la partición que se montará como raíz. Estas dos cláusulas forman parte de la sección global.

Las entradas `image` y `label` sirven para especificar donde se encuentran las imágenes a cargar y las etiquetas asociadas a ellas. Así un ejemplo básico sería el siguiente:

```
boot=/dev/hda
root=/dev/hda3
image=/vmlinuz
        label=Linux
image=/boot/memtest86.bin
        label=memtest
```

Pero la funcionalidad de LILO no acaba aquí: LILO tiene otros muchos parámetros. A continuación se presentan los más usados:

Opciones globales:

BOOT=dispositivo Indica que dispositivo contiene el sector de arranque. Esta opción es obligatoria.

COMPACT Intenta intenta colocar todo el núcleo en sectores contiguos del disco para acelerar su carga en memoria.

DEFAULT=nombre Especifica la imagen a arrancar por defecto.

TIMEOUT=tiempo Indica el tiempo en décimas de segundo que se ha de espera antes de arrancar la imagen por defecto.

IGNORE-TABLE Ignora las tablas particiones corruptas.

LBA32 Genera direcciones lógicas de 32 bits. Esto es especialmente útil en sistemas que tienen discos de mas de 1024 cilindros ya que permitirá redimensionar los parámetros físicos para dejarlo con menos de 1024 cilindros.

LINEAR Genera direcciones lineales en vez de direcciones con parámetros físicos.

PROMPT Fuerza a que aparezca el *prompt*.

VERBOSE=nivel Indica el nivel de información que mostrará LILO.

Opciones por imágenes

ALIAS=nombre Indica nombres alternativos asociados a una imagen.

APPEND=cadena Añade las opciones indicadas al kernel.

INITRD=nombre Indica el fichero donde se encuentra la información a cargar en el disco RAM al arrancar.

LABEL=nombre Especifica una etiqueta para esta imagen.

OPTIONAL Omite esta imagen si el fichero no esta disponible.

OTHER=dispositivo Se usa para arrancar otros sistemas operativos que se encuentran en el dispositivo especificado.

PASSWORD=clave Indica la clave para bloquear el acceso al intentar cargar esta imagen.

RESTRICTED Solo usa la clave si se pasan parámetros al núcleo.

ROOT=dispositivo Indica el dispositivo a montar como raíz.

SINGLE-KEY Especifica la tecla para arrancar esta imagen.

VGA=modo Indica el modo de la VGA[3]. Los posibles valores son `normal` que entra en modo carácter de 80x25, `extended` modo carácter de 80x50, `ASK` pregunta cual de los modos disponibles utilizara.

Opciones de geometría del disco

SECTORS=sectores,HEADS=cabezas,CYLINDERS=cilindros Especifica los parámetros físicos del disco.

activate Define dicha partición como la partición activa.

deactivate Define dicha partición como una partición inactiva.

[3]Video Graphic Array

También pueden pasarse opciones de configuración al núcleo vía línea de comandos. De éstas destacan:

-C fichero Usa ese fichero de configuración.

-v Aumenta los detalles a mostrar.

-b dispositivo Indica el dispositivo de arranque.

-D nombre Especifica la imagen a arrancar por defecto.

-r directorio Especifica el directorio dentro de la partición donde se contiene la partición raíz. Antes de hacer cualquier otra cosa se realizara un *chroot* a este directorio.

-s fichero Permite especificar un fichero alternativo para salvar el sector de arranque.

-t Solamente hace una prueba de la configuración dada, ni escribe en el disco ni modifica el sector de arranque.

-u dispositivo Restaura el sector de arranque del dispositivo. Se puede usar `-s` para especificar el fichero que se usara.

2.2.3. Grub: GRand Unified Bootloader

Las fases de arranque de Grub.

Grub divide su proceso de arranque en tres fases: fase 1, fase 1.5 y fase 2. La fase 1 se ocupa de cargar solamente el primer sector correspondiente a la 1.5 o la fase 2. Estas fases son cargadas por ellas mísmas.

La fase 1.5 se utiliza cuando la fase 2 es muy grande. La fase 1.5 se encarga de hacer de puente entre la fase 1 y la fase 2, permitiendo que la fase 2 se encuentre en un sistema de ficheros y no en sectores del MBR, evitando asi los problemas de espacio.

La fase 2 se encarga de todo lo demás: cargar el intérprete de comandos, aceptar una imagen y cargarla para que arranque. En comparación con LILO, Grub permite que su segunda fase sea mucho más grande y, gracias a ello, ofrece mayor funcionalidad.

Instalación de Grub

Hay dos formas de instalar Grub: usando el script `grub-install` o de forma manual.

La modalidad rápida para instalar Grub se basa en el uso del script `grub-install`, suministrado con el propio Grub. Simplemente se lanza `grub-install /dev/hdX` donde `X` es la letra correspondiente a la unidad de disco en la que queremos instalar Grub.

Para instalar Grub de forma manual, primero hay que hacer un disco de arranque de Grub. Este disco se hace copiando las fases 1 y 2 mediante el uso de la utilidad `dd`. Una vez se haya arrancado desde el disquete, se establecerá la partición raíz con el comando `root (hdX,Y)`, donde `X` e `Y` indican unidad y partición, respectivamente. El siguiente paso es cargar Grub en el MBR de la unidad usando `setup (hdX)` donde `X` es la unidad.

Grub utiliza números para identificar los dispositivos, frente a la nomenclatura clásica usada por LILO. La serie comienza con el 0, de modo que `hd0` sería equivalente a `hda` y `hda0,1` a `hda2`.

Configuración de Grub

El fichero de configuración de Grub es `/boot/grub/grub.conf`. Tiene una sintaxis muy parecida a la usada por LILO, ya que se compone de una seccion global, donde se especifican todos los parámetros comunes a las particiones, y de varias secciones locales donde se especifican las opciones para cada una de las imágenes.

Ahora pasaremos a mostrar un ejemplo de un fichero de configuración básico de Grub:

```
# Establece la partición que arrancara por defecto
  default 0
# Establece el tiempo de espera antes de arrancar la imagén por defecto.
  timeout 30
# Imagen de linux
  title GNU/Linux
  kernel (hd1,0)/vmlinuz root=/dev/hdb1
# Imagen de Windows
  title Windows
  root (hd0,0)
  makeactive
  chainloader +1
```

Grub tiene una ventaja fundamental sobre LILO: la capacidad de interactuar directamente con los ficheros de configuración desde un pequeño *minishell*. Para acceder a

esta funcionalidad bastará con apretar la tecla `e` justo cuando sale la carátula de selección de imágenes. Desde aquí se podrá modificar la configuración temporalmente, de forma que si se cometió un error al introducir algún parámetro en el fichero de configuración, podrá solucionarse el fallo. Esta corrección no es permanente a menos que se edite el fichero de configuración —`/boot/grub/grub.conf`—.

2.3. El núcleo en el arranque

El gestor de arranque ya se ha encargado de cargar el núcleo comprimido en memoria. Ahora, el propio núcleo se encarga de descomprimirse, quedando preparado para comenzar su ejecución.

Lo primero que hace el núcleo es inicializar algunos dispositivos. Acto seguido y, si procede, monta el disco RAM —`initrd`— como raíz. Normalmente, el uso de un disco RAM está motivado porque se necesitan cargar ciertos módulos para iniciar dispositivos fundamentales en el arranque, como por ejemplo un RAID o un disco SCSI.

El siguiente paso es montar la verdadera partición raíz —en modo sólo lectura—. En el caso de que la inicialización se hubiera apoyado en un disco RAM, éste es sustituido como raíz del sistema.

Finalmente, el núcleo invocará a `init` que se encargará de terminar el proceso de arranque. Al respecto, hay que destacar que el núcleo buscará a `init`, si no se le indica lo contrario, en `/sbin`, `/etc` y `/bin`. En caso de no encontrarlo, utilizará `/bin/sh` en su lugar.

2.4. El proceso `init` y los niveles de ejecución

Llegados a este punto, el núcleo ha cedido el control del inicio a `init`, que se encargará de realizar todos los ajustes que aún quedan pendientes. Antes de seguir explicando qué ocurren en el arranque, parece buena idea detener la mirada en dos conceptos previos. El primero de ellos, será el concepto de *nivel de ejecución*, una de las ideas básicas de esta clase de sistemas. Acto seguido, se abordará una descripción, algo superficial, del proceso `init`, que presenta interesantes peculiaridades.

2.4.1. Conceptos previos

Niveles de ejecución

Antes de profundizar en el funcionamiento de `init`, resulta de vital importancia introducir el concepto de *nivel de ejecución* —o *run level*—. A grandes rasgos, podría decirse que se trata de un "modo de funcionamiento" del sistema, definido por un conjunto de servicios. Estos niveles determinan que procesos ha de lanzar `init` en cada caso.

También podría entenderse un *run level* simplemente como una configuración de software.

Esto permite que un mismo sistema pueda ofrecer un conjunto distinto de servicios —y configuración— dependiendo de la situación. Por ejemplo, podría definirse un nivel de ejecución dónde no se habiliten los servicios de red, otro donde no se lance interfaz gráfica alguna, etc.

Por lo general, en los sistemas GNU/Linux se consideran los siguientes niveles de ejecución:

0-6 Podría decirse que estos son los principales y, como se verá, son los que se usan comunmente. Los niveles 0, 1 y 6 están reservados y tienen un uso especial.

7-9 Aunque se trata de niveles de ejecución válidos, tradicionalmente no están documentados y no suelen utilizarse.

Exceptuando los niveles reservados —0, 1 y 6—, el uso que se da a cada *run level* es una de las tantas cosas que varía entre las diferentes distribuciones. Sin embargo, en el marco del proyecto Linux Standard Base, se ha determinado una misión para cada nivel. En la tabla 1 puede observarse esta especificación, respetada por *Red Hat Linux 9*.

Pero, ¿cómo se plasma todo esto en el sistema? Es decir, cuando se va a entrar en un nivel de ejecución, ¿de qué forma se sabe qué servicios han de iniciarse y cuáles, eventualmente, han de detenerse?

Siguiendo la filosofía de los sistemas UNIX, por cada servicio ha de existir un script que ofrece, entre otras cosas, la posibilidad de ponerlo en marcha o detenerlo. Éstos scripts se alojan en el directorio `/etc/rc.d/init.d`, aunque puede variar entre las distintas distribuciones. Según el *Linux Standard Base* deben admitir estas "acciones": `start`, `stop`, `restart`, `try-restart`, `reload`, `force-reload` y `status`. Así, para iniciar un servidor `apache`, el comando podría ser:

Nivel	**U**so
0	Apagado
1	Modo monousuario
2	Multiusuario sin NFS
3	Multiusuario completo
4	Reservado para uso local
5	Multiusuario con xdm o equivalente
6	Reinicio

Cuadro 1: Cometido de los niveles de ejecución

`/etc/rc.d/init.d/apache start`

Por otra parte, para cada nivel de ejecución existe un directorio `/etc/rc.d/rcN.d`, dónde `N` es su identificador. En él, existen una serie de enlaces a los scripts de `init.d` con nombres del tipo: `S99apache` ó `K20postfix`. Estos nombres no son un capricho: indican si un servicio debe iniciarse (`S`, acción `start`) o detenerse (`K`, acción `stop`) y en qué orden respecto al resto.

A continuación, un ejemplo:

```
K99xdm -> ../init.d/xdm
S10sysklogd -> ../init.d/sysklogd
S11hotplug -> ../init.d/hotplug
S11klogd -> ../init.d/klogd
S14ppp -> ../init.d/ppp
S20cupsys -> ../init.d/cupsys
S20fam -> ../init.d/fam
S20inetd -> ../init.d/inetd
S20makedev -> ../init.d/makedev
S20mysql -> ../init.d/mysql
S20postfix -> ../init.d/postfix
S20ssh -> ../init.d/ssh
S20xfs -> ../init.d/xfs
S20xprint -> ../init.d/xprint
S89atd -> ../init.d/atd
S89cron -> ../init.d/cron
S91apache2 -> ../init.d/apache2
S99rmnologin -> ../init.d/rmnologin
S99stop-bootlogd -> ../init.d/stop-bootlogd
```

En este caso, se pondrían en marcha todos los servicios, exceptuando `xdm`. Es más, si éste servicio estuviera ejecutándose, se detendría.

Presentación de `init`

El proceso `init` es realmente peculiar, ya que se trata del antecesor de todos los procesos que se ejecutan en un sistema. Por si esto fuera poco, `init` presenta, además, la particularidad de ser inmune a la señal `SIGKILL`, es decir, en cierto modo es "inmortal". Pese a su importancia, `init` no deja de ser un proceso de nivel de usuario.

La primera misión de `init` es, después de que el núcleo le haya cedido el control, terminar las tareas de inicialización para que el sistema quede operativo. Sin embargo, su área de actuación no se limita a la fase de arranque. Éste proceso jugará un papel básico que, poco a poco, se irá descubriendo.

Antes de continuar, hay que resaltar que en el mundo de los sistemas UNIX, existen dos enfoques respecto a `init`: *System V —SysV—* y *BSD*. No es objetivo de este documento explicar sus diferencias, ventajas y desventajas. Bastará con decir que en la mayoría de las distribuciones de *GNU/Linux* se utiliza un enfoque basado en *System V*[4], y *Red Hat* no es una excepción.

Para que el lector se haga una idea de la transcendencia de `init`, se presentará a continuación un ejemplo muy ilustrativo: el comando `pstree` despliega un árbol en el que sitúa a todos los procesos que se están ejecutando en el sistema, indicando sus relaciones padre-hijo.

Ejecutando este comando en un sistema GNU/Linux, se obtendrá una salida de este estilo:

```
$ pstree -p
init(1)-+-aio/0(9)
        |-bash(342)---pstree(413)
        |-bash(343)---vi(372)
        |-cron(331)
        |-dhclient(384)
        |-events/0(3)
        |-getty(344)
        |-getty(345)
        |-getty(346)
        |-getty(347)
        |-kblockd/0(4)
        |-khubd(5)
        |-kjournald(12)
        |-klogd(279)
        |-kseriod(11)
        |-ksoftirqd/0(2)
        |-kswapd0(8)
        |-lisa(408)
        |-pdflush(6)
```

[4]Escrito por Miquel van Smoorenburg

```
|-pdflush(7)
`-syslogd(276)
```

Puede verse claramente cómo todos los procesos —bash, cron, getty...— "cuelgan" a partir de `init`, cuyo *PID*, además, es 1. De los procesos que se muestran en el árbol, quizás el más interesante sea `getty`, que permitirá la entrada a los usuarios por medio de terminales. El modo en que `init` hace uso de `getty` —o similares— se estudiará más adelante.

2.4.2. ¿Qué ocurre en el arranque?

Tras haber presentado a `init` y el concepto de *nivel de ejecución*, llega el momento de ver qué ocurre después de que el núcleo se encuentre ya cargado y convenientemente inicializado.

Como se mencionó anteriormente, el núcleo es el encargado de invocar a `init`. Éste consulta el fichero `/etc/inittab`, que contiene información acerca de qué procesos deben ejecutarse tanto durante el inicio como durante el funcionamiento normal, además de otros parámetros como pudiera ser el nivel de ejecución por defecto —normalmente entre el 1 y el 5—.

En el caso de *Red Hat*, este fichero indicará a `init` que lo primero que debe hacer es ejecutar el script `/etc/rc.d/rc.sysinit`. Éste lleva a cabo una larga lista de tareas de inicialización, tales como, aunque no necesariamente en este orden:

- Desmontar `initrd` —si el núcleo hizo uso de él—.
- Iniciar el demonio `devfsd` si es necesario.
- Montar `/proc` y `/sys`, si procede.
- Configurar algunos parámetros del kernel —`/etc/sysctl.conf`—.
- Inicializar el reloj del sistema —`/etc/sysconfig/clock`—.
- Cargar el mapa de teclado.
- Establecer el nombre de la máquina.
- Inicializar subsistemas como ACPI, USB o ISA.

- Montar de nuevo la partición raíz (/), pero ahora en modo lectura-escritura (rw). Si encontrara que el sistema no fue desactivado anteriormente de manera limpia, realizará un chequeo. Ésto ocurrirá también cuando se haya superado cierto número de operaciones de montajes sin haberlo chequeado.
- Inicializar LVM o RAID —si es necesario—.
- Activar las particiones de `swap`.
- Inicializar —borrando y reconstruyendo convenientemente— el fichero `/etc/mtab`.
- Montar los sistemas de archivos que aún no se han montado, después de hacerle su correspondiente chequeo. No se montan ni `/proc` ni `sys` —ya montado— ni sistemas de archivo vía `NFS`.
- Comprobar cuotas.
- Configurar los puertos serie.
- Cargar módulos.
- Activar optimizaciones de discos.

La lista de tareas es mayor aún. Sin embargo, y puesto que este manual no pretende ser una guía exhaustiva, se ha decidido enumerar sólo aquellas que parecen más importantes —u obvias—.

Una vez llegados a este punto, el sistema está convenientemente inicializado para llevarlo al *run level* oportuno. `init` ejecutará los scripts situados en el directorio `/etc/rc.d/rcN.d`, donde `N` es el caracter que identifica al nivel de ejecución. Por defecto, en Red Hat, es el 5, así que ejecutaría todos los scripts situados en `/etc/rc.d/rc5.d`.

Habiendo entrado ya en el nivel de ejecución necesario, a `init` sólo le falta hacer un "fork" al proceso `/sbin/mingetty` para cada consola virtual. Dependiendo del nivel de ejecución, se dispondrá de un número diferente de consolas virtuales: ninguna para los niveles 0 y 6, una para el nivel 1 y seis para los niveles del 2 al 5. Esto, como tantas otras cosas, es configurable. El proceso `mingetty` permitirá que un usuario introduzca su nombre —login— y su contraseña pudiendo acceder así al sistema.

Por fin, el sistema ya está en marcha.

2.4.3. `inittab`

Estructura general

Tal y como se mencionó con anterioridad, cuando `init` toma el control, lo primero que hace es leer el fichero `inittab`, que le va a dictar qué tiene que hacer.[5]

En este archivo, `init` encontrará el nivel de ejecución por defecto, que será el que tenga que poner en marcha en el arranque si no se le indica lo contrario.

Este fichero está formado por un conjunto de entradas de la forma:

```
id:runlevels:acción:proceso
```

id Secuencia única de entre uno y cuatro caracteres que identifica unívocamente a la entrada —antiguamente había un límite de dos caracteres—.

runlevels Niveles de ejecución en los que la acción especificada se lleva a cabo. Se pueden especificar cualquier cantidad de ellos. Si no se indica ninguno, la acción se llevará a cabo en todos los niveles de ejecución.

acción Acción a realizar.

proceso Proceso a ejecutar.

De entre las acciones que pueden llevarse a cabo, destacan:

respawn El proceso se lanza de nuevo cada vez que termina. `mingetty`[6] es uno de los ejemplos más claros.

wait El proceso se inicia cuando se entra en el nivel de ejecución indicado e `init` espera a que termine.

boot El proceso se ejecuta durante el inicio del sistema. Se ignora el campo `runlevels` y se ejecuta para cualquier nivel.

bootwait Igual que *boot*, pero `init` espera por la finalización del proceso.

sysinit El proceso se ejecuta durante el inicio, antes que cualquier entrada *bootwait* o *boot*.

initdefault Especifica el nivel de ejecución por defecto.

[5] Hay una excepción, y es el caso del nivel de ejecución monousuario, que se estudiará más adelante.
[6] `mingetty` es una versión minimalista de `getty` sin soporte, por ejemplo, para líneas serie.

ctrlaltdel El proceso se ejecutará cuando `init` reciba la señal `SIGINT`.

Existen algunas acciones más, como *powerwait*, *once* o *kbrequest*. Para información acerca de ellas, no hay más que acudir a la página del manual de `inittab`.

Las líneas que comiencen por una almohadilla (#), se consideran comentarios y son ignoradas a la hora de interpretar el contenido del fichero.

El `inittab` de *Red Hat 9*

Vista la estructura general de un fichero `/etc/inittab`, va a analizarse a continuación el que viene con *Red Hat 9*. Esto ayudará a comprender con algo más de profundidad el proceso de inicio del sistema.

La primera línea efectiva del fichero, tras los comentarios iniciales, especifica el nivel de ejecución por defecto, el 3:

```
id:3:initdefault:
```

Tal y como se vio anteriormente, `init` ejecuta el script `rc.sysinit`. Ésto viene indicado por la línea:

```
si::sysinit:/etc/rc.d/rc.sysinit
```

Dependiendo del nivel de ejecución, `init` ejecutará los scripts contenidos en el directorio `/etc/rc.d/rcN.d` —donde N es el nivel de ejecución—. Ésto, sin embargo, no lo hace `init` directamente, sino que invoca al script `/etc/rc.d/rc` pasándole como argumento el *nivel de ejecución* correspondiente:

```
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
```

La siguiente línea especifica la acción a realizar en el caso de pulsar la famosa combinación `ctrl+alt+del`: se reiniciará el sistema.

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

Si la corriente falla —y obviamente se cuenta con un SAI—, el sistema se apagará. En caso de que se reestablezca, se cancelará el proceso.

```
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
```

El siguiente fragmento provoca que `init` inicie un proceso `mingetty` en seis consolas diferentes para los *niveles de ejecución* del 2 al 5:

```
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
```

Si se desea habilitar más consolas virtuales, no hay más que añadir una línea que lo indique. Una línea como la siguiente podría habilitar una consola más en `alt+f8`.

```
8:2345:respawn:/sbin/mingetty tty8
```

Finalmente, en caso de que se entrara en el nivel de ejecución 5, se cargaría la interfaz gráfica según lo especificado en esta línea:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

En estos dos últimos fragmentos, nótese que la acción especificada es `respawn`, con lo que las consolas y el entorno gráfico se inician de nuevo cada vez que terminen.

3. Detención del sistema

Tan importante como poner en marcha un sistema es detenerlo correctamente. Durante esta fase, se llevan a cabo una serie de pasos que permiten que el sistema quede en un estado estable y consistente. No en vano, son de sobra conocidos los problemas derivados de, por ejemplo, un mal apagado.

Por lo tanto, es fundamental que el administrador conozca qué ocurre durante esta fase y cómo sacar partido a las herramientas que el sistema pone a su alcance.

Cuatro son los comandos asociados a estas tareas: `shutdown`, `halt`, `reboot` y `poweroff`. El resto de la sección, estará dedicada a estudiarlos con algo más de profundidad.

3.1. `shutdown`

Por medio de `shutdown` el sistema puede detenerse, reiniciarse o apagarse de forma segura y ordenada. Esto, en los sistemas *GNU/Linux*, es algo fundamental para evitar problemas. No es un suceso extraño perder datos, por ejemplo, con un corte de suministro eléctrico, si bien los sistemas de archivos con *journaling*[7] han paliado en gran parte este problema.

Funcionamiento

A grandes rasgos, cuando se lanza `shutdown`, se notifica a los usuarios de este hecho y, además, se bloquea el sistema para que nadie más pueda acceder —creando el archivo `/etc/nologin`—, exceptuando el *root*. Acto seguido, se envía la señal `SIGTERM` a todos los procesos no definidos en `inittab` para el siguiente *run level*, provocando que terminen su ejecución de modo ordenado. Poco después, se envía una señal `SIGKILL` para que los procesos que no hayan atendido a `SIGTERM` concluyan también su ejecución —pero en este caso no de una manera "limpia"—.

`shutdown` lleva a cabo su cometido enviando una señal a `init` para que cambie a uno de estos *niveles de ejecución*, en función del efecto que se desee conseguir:

- Apagar o detener el sistema (nivel 0, opción "`-r`").
- Entrar en modo *monousuario* (nivel 1, opción por defecto).

[7] GNU/Linux dispone de varios sistemas de archivos de este tipo, principalmente: *ext3*, *ReiserFS*, *XFS*, y *JFS*.

- Reiniciar el sistema (nivel 6, "`-h`").

Una vez cambiado el nivel de ejecución, si procede, se invoca a `halt`, `reboot` o `poweroff` según sea necesario.

Sintaxis

La invocación de `shutdown` tiene la forma:

```
/sbin/shutdown [-t segundos] [-opciones] tiempo [mensaje de aviso]
```

Como puede observarse, el único argumento obligatorio es `tiempo`, mediante el cuál se indica el momento en que debe producirse la operación. Principalmente, este parámetro puede adoptar dos formas:

- El clásico formato horario `hh:mm`.
- El número de minutos que deben transcurrir antes de proceder, indicado como `+m`. Para indicar que no se retrase el proceso, puede utilizarse la palabra `now` —equivalente a `+0`—.

Si se indica un retardo, `shutdown` avisará a los usuarios de la inminente operación cuando falten diez minutos, y a partir de cinco lo hará cada minuto que pase. Precisamente cuando resten tan sólo cinco minutos —o menos— se creará el archivo `/etc/nologin`. Sin embargo, no llevará a cabo ninguna acción más hasta que el tiempo se haya agotado. Como no, el *root* podrá cancelar el proceso usando el argumento `-c` —o pulsando el consabido `ctrl+c`—.

Otras dos opciones, mencionadas anteriormente, resultan de especial interés: éstas son `-h` y `-r`, que servirán para indicar a `shutdown` si debe invocar, al final, a `halt` o `reboot`. Como ya se vio anteriormente, la opción por defecto es entrar en *modo monousuario*, así que ésto será lo que se haga si no se especifica ninguno de las dos opciones anteriores.

Pero, ¿qué ocurre con `poweroff`? Hasta ahora sólo se ha hablado de como provocar la invocación de `reboot` y `halt`. Sin embargo, no se ha mencionado nada de `poweroff`. Esto se debe a que `shutdown` no invoca a `poweroff`, sino a `halt`, que será quién se encargue de, si procede, invocar a `poweroff`. No obstante, usando las opciones `-P` y `-H` se indica a `shutdown` que, al lanzar `halt`, le indique si debe o no apagar la máquina —éste es el comportamiento por defecto, si el núcleo lo permite—.

Para terminar, mencionar que existen otras posibilidades, como indicar que en el próximo inicio no se realicen chequeos de los sistemas de archivos o, justamente, todo lo contrario. Como siempre, la lectura de la página del manual de `shutdown` está más que aconsejada.

Control de acceso

En principio, sólo el *root* puede invocar a `shutdown` —en línea de comandos—. Sin embargo, como ya se estudio con anterioridad, es posible que cualquier usuario lo lance utilizando la combinación de teclas `ctrl+alt+del`. Entonces, ¿cualquier usuario que tenga acceso a físico al teclado de la máquina es capaz de dejar el sistema fuera de servicio?

Obviamente, cualquier persona con acceso físico absoluto al sistema no tendría más que desenchufarla de la corriente eléctrica para dejarla fuera de servicio. Sin embargo, existe un mecanismo para evitar soluciones menos drásticas, como el uso de la combinación `ctrl+alt+del`.

Además de la obvia solución de comentar la línea correspondiente en el `inittab`, es posible instruir a `shutdown` para que, antes de realizar su cometido, consulte una lista de usuarios autorizados. Si el *root* o uno de los usuarios de la lista está en el sistema en una de las consolas virtuales, se ejecutará normalmente. La lista está contenida, con un nombre por fila e ignorando las comenzadas por (#), en el fichero `/etc/shutdown.allow` y a `shutdown` debe pasársele la opción `-a`.

La línea del `/etc/inittab` podría quedar así:

```
ca::ctrlaltdel:/sbin/shutdown -t3 -a -r now
```

3.2. `halt`, `poweroff` y `reboot`

En principio, `halt`, `poweroff` y `reboot` son comandos destinados a detener, apagar y reiniciar la máquina, respectivamente. Sin embargo, lo realmente curioso, es que tanto `poweroff` como `reboot` son enlaces simbólicos a `halt`. Es decir, los tres son el mismo programa.

```
$ cd /sbin && ls -lh halt reboot poweroff
-rwxr-xr-x    1 root     root         9,0K 2003-12-23 11:19 halt
lrwxrwxrwx    1 root     root            4 2004-02-04 22:36 poweroff -> halt
lrwxrwxrwx    1 root     root            4 2004-02-04 22:36 reboot -> halt
```

El “truco” está en que `halt` actúa de una u otra forma según el modo en que se lo invoque.

Una invocación de `halt` provoca que este lea el fichero `/var/log/wtmp` para averiguar si el sistema está siendo detenido —por ejemplo, debido a una ejecución de `shutdown`—. En ese caso, le indica al núcleo que detenga, reinicie o apague la máquina, según proceda.

Si, por contra, el sistema no se encontrara en los niveles de ejecución 0 ó 6 —recordar que correspondían a detener/apagar y reiniciar—, `halt` invocaría él mismo al propio `shutdown` —pasándole la opción `-h` o `-r`, según el caso.

Hay que destacar que este comportamiento se realiza sólo con versiones de *init* posteriores a la 2.74. Anteriormente, se realizaba la detención o el reinicio del sistema sin llevar a cabo el proceso de `shutdown` correspondiente. Para forzar este comportamiento, debe especificarse la opción `-f`.

`halt` admite, además, otras opciones: como uso y costumbre, `man halt` o `info halt` ofrecerán más documentación al respecto.

Apéndices

A. Limitaciones y fallos comunes usando LILO

LILO se ve afectado por las limitaciones de la BIOS. Esto implica que si la BIOS no soporta discos de mas de 1024 cilindros o el acceso a mas de dos discos para establecer el arranque, LILO no solucionara el problema.

En el proceso de instalación de LILO es posible que nos muestre algunos mensajes de aviso como el siguiente: *Warning: BIOS drive 0xnúmero* Esto se debe precisamente a la limitación de las BIOS antiguas que no soportaban mas de dos discos. Este mensaje es simplemente un aviso ya que si la BIOS carece de este defecto este mensaje debe de ser ignorado.

Al cargar LILO muestra LILO en la pantalla según las letras que se muestren indicaran los siguientes fallos:

L error: No se puede cargar la segunda fase. El numero indica el error.

0x00: Error interno.Ficheros corruptos, intentar cargar con LINEAR para que se ponga por debajo del cilindro 1024.

0x01: Comando ilegal. Esto no debería pasar. Intenta acceder a un disco que no es soportado por la BIOS.

0x02: Marca de la dirección no encontrada. Fallo en el disco.

0x03: Disco protegido contra escritura.

0x04: Sector no encontrado. Fallo en la geometría.

0x06: Linea de cambio activa. Es un error temporal, reiniciar.

0x07: Inicialización no valida. La BIOS fallo al inicializar la controladora. Comprobar los parámetros de la BIOS y reiniciar.

0x08: DMA overrun. Esto no debería pasar, reinicia.

0x09: DMA intenta pasar la barrera de 64 K. Esto no debería pasar pero indicaría un fallo en la geometría del disco.

0x0A: Fallo en el flag del sector.

0x0B: Fallo en el flag de la pista.

0x0C: Medio no valido. No debería pasar. Fallo en el disco.

0x10: Error de CRC. Fallo en el disco, intentar reinstalar si sigue pasando cambiar el disco

0x11: Corrección ECC conseguida. Se produjo un fallo de lectura que el disco recupero, LILO no sabe como tratar esto así que hay que reiniciar y debería de funcionar.

0x20: Fallo de la controladora. No debería de pasar. Cambiar la controladora.

0x40: Fallo de búsqueda. Fallo del disco.

0x80: Timeout. El disco no está listo. El disco está mal o no está girando.

0x99: Índice del sector incorrecto en la segunda fase. Reinstalar.

0x9A: No hay firma de la segunda fase. Reinstalar.

0xAA: Unidad no preparada. El disco está mal.

0xBB: Error en la BIOS. Esto no debería pasar. Si pasa intentar reinstalar quitando COMPACT, LINEAR o LBA32.

0xFF: Fallo en el sensor de lectura.

LI: Fallo al ejecutar la segunda fase, eso puede ser por un fallo de la geometría o porque se ha movido /boot/boot.b sin ejecutar el instalador del mapas.

LIL: No puede cargar el descriptor de las tablas del fichero de mapas. Fallo de geometría o fallo en el medio.

LIL?: Se ha cargado la segunda fase en una dirección incorrecta. eso puede ser por un fallo de la geometría o porque se ha movido /boot/boot.b sin ejecutar el instalador del mapas.

LIL-: La tabla de descripción está corrompida. eso puede ser por un fallo de la geometría o porque se ha movido /boot/boot.b sin ejecutar el instalador del mapas.

LILO: LILO se ha cargado correctamente

B. wtmp

Éste documento se ha referido al fichero `/var/log/wtmp` en un par de ocasiones, así que parece justo explicar, aunque sea brevemente, cuál es su cometido.

El fichero `wtmp` registra todos los inicios y salidas de sesión —*logins* y *logouts*—. Su formato es prácticamente exacto al de `/var/run/utmp`, fichero que permite obtener información de quiénes están utilizando el sistema actualmente, y que contiene información acerca de:

- Tipo de *login*.
- PID del proceso.
- Nombre de dispositivo `tty`.
- Nombre de usuario.
- Nombre de la máquina para el caso de *login* remoto.
- ID de sesión, usado para el manejo de ventanas.
- Estado de salida de un proceso marcado como `DEAD_PROCESS`.
- Dirección IP de la máquina remota.

`wtmp` registra las operaciones de cierre y reinicio del sistema, asociando el nombre de terminal "∼" y el usuario `shutdown` o `reboot`, respectivamente.

Como se estudio, `halt` se encargará de consultar esta información.

C. Referencias

Además de la documentación en línea del sistema —de donde se extrajo gran parte de la información—, existe un conjunto de referencias que podrían resultar de interés.

[1] M. Carling, Stephen Degler y James Dennis, *Administración de Sistemas Linux* Prentice Hall, 1999

[2] Vicente López Camacho y otros, *Linux. Guía de Instalación y Administración* Mc Graw Hill, 2001

[3] Red Hat, Inc., *Red Hat Linux Reference Guide* *http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/index.html*

[4] Mark Allen, *How Linux Works CTDP Guide, versión 0.6.0* *http://www.comptechdoc.org/os/linux/howlinuxworks/index.html*

[5] Roberto Alsina, *The Linux Booting Process Unveiled* *http://www.pycs.net/lateral/stories/23.html*

[6] Kim Oldfield, *The Linux Boot Process (or What happens before the login prompt)* *http://www.pycs.net/lateral/stories/23.html*

[7] Jens Benecke, *The boot process* *http://www.linuxnetmag.com/en/issue4/m4boot1.html*

[8] Wayne Marshall, *Boot with GRUB* *http://www.wbglinks.net/pages/reads/linux/grub.html* Linux Journal, 2001