

Clase 6: Inicio del sistema y proceso de login

Introducción

En el capítulo 2 aprendimos cómo instalar la última versión estable de Debian. Luego de finalizar, el sistema quedó listo para su uso por parte de una de las cuentas de usuario que habíamos creado previamente.

En este capítulo hablaremos sobre lo que sucede desde el momento en que encendemos el equipo hasta que iniciamos sesión y comenzamos a utilizar la línea de comandos.

La secuencia de arranque

Al encender el equipo, el **BIOS** (Basic Input-Output System) o **UEFI** (Unified Extensible Firmware Interface) llevan a cabo una revisión del hardware conocida comúnmente como **POST** (Power-On Self Test). A continuación, se busca un gestor de arranque (por lo general, **GRUB**) en el **MBR** (Master Boot Record) o en la partición **EFI** de un dispositivo de almacenamiento.

A partir de ese momento, el control del proceso se pasa a manos de **GRUB**, que se encargará de cargar el kernel, el cual reconocerá y configurará los dispositivos de hardware presentes en el equipo y los preparará para su uso. Como próximo paso, el núcleo también será responsable por ejecutar el primer proceso, también conocido como **init**. Finalmente, este último utilizará el gestor del sistema (**systemd** o **Upstart**) o los scripts de **SysVinit** para continuar el inicio de los demás servicios. El proceso de arranque culmina al presentar una interfaz de inicio de sesión de modo texto o gráfico, según sea el caso.

*Los servicios o procesos son programas que corren en segundo plano (de manera no interactiva). Otro nombre por el que son conocidos es **daemons** (a veces traducido como **demonios** en castellano).*

A continuación haremos un repaso por los 3 métodos de gestión del sistema más utilizados a lo largo del tiempo.

SysVinit

Hasta no hace mucho tiempo, la mayoría de las distribuciones más utilizadas utilizaban como base de su funcionamiento el sistema de arranque y de administración de servicios conocido como **SystemV** o **SysVinit**. Este sistema, heredado de Unix, contempla 5 niveles útiles de funcionamiento (de ahí el nombre **SystemV**, correspondiente al número 5 en el sistema romano) numerados del 1 al 5. A

estos se les suma el nivel 0 (apagado del equipo) y el 6 (reinicio del sistema). Cada uno de ellos es lo que se conoce como *niveles de corrida* o *runlevels* en Linux.

Cada *runlevel* se encuentra asociado a un cierto número de servicios que por defecto deben iniciarse automáticamente cuando encendemos el equipo, y que deben detenerse al reiniciarlo o apagarlo. Por ejemplo, en el directorio **/etc** en un sistema Debian Wheezy podemos encontrar 7 directorios con nombre **rcN.d**, donde **N** es un número del 0 al 6. Es precisamente dentro de estos directorios que se encuentran una serie de enlaces simbólicos a los ejecutables que inician y detienen los servicios del sistema en cada *runlevel* respectivo, como vemos a continuación:

```
root@wheezy:~# ls -l /etc | grep rc[0-6].d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc0.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc1.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc2.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc3.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc4.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc5.d
drwxr-xr-x 2 root root 1024 Nov 1 01:15 rc6.d
root@wheezy:~# ls -l /etc/rc2.d
total 1
-rw-r--r-- 1 root root 677 Jul 14 2013 README
lrwxrwxrwx 1 root root 14 Nov 1 00:29 S01motd -> ../init.d/motd
lrwxrwxrwx 1 root root 17 Nov 1 01:15 S13rpcbind -> ../init.d/rpcbind
lrwxrwxrwx 1 root root 20 Nov 1 01:15 S14nfs-common -> ../init.d/nfs-common
lrwxrwxrwx 1 root root 17 Nov 1 01:15 S16rsyslog -> ../init.d/rsyslog
lrwxrwxrwx 1 root root 32 Nov 1 01:15 S16virtualbox-guest-utils -> ../init.d/virtualbox-guest-utils
```

Estos directorios contienen los enlaces simbólicos a los servicios que deben iniciarse en cada runlevel



*Por el momento, podemos asociar el concepto de enlaces simbólicos (también llamados **soft links**) con el de **acceso directo**, con el que quizás estemos más familiarizados. En síntesis, se trata de una representación de un recurso del sistema que apunta a dicho recurso pero que es independiente del mismo.*

*La **S** utilizada como prefijo en los nombres de los enlaces simbólicos indica que, en este nivel de de corrida en particular, se debe iniciar el servicio asociado. Por otro lado, la letra **K** indica lo contrario (el servicio debe apagarse o permanecer detenido al ingresar al **runlevel** en cuestión).*

Es importante tener en claro que lo primero que hace **init** es leer el archivo **/etc/inittab** para identificar los próximos pasos a seguir. Por brevedad, nos referiremos solamente a la detección del *runlevel* por defecto, lo que se indica en la siguiente línea (**2** en este ejemplo):

```
# The default runlevel.
id:2:initdefault:
```

A continuación, **init** se dirige a **/etc/rc2.d** en este caso. En este directorio podemos apreciar una serie de enlaces simbólicos cuyos nombres comienzan con la letra **S**, tal

como mencionamos antes, y un número de dos dígitos. Cada uno de ellos apunta a un servicio que se debe levantar en el *runlevel* 2 por orden numérico y alfabético). Por ejemplo, de la imagen anterior podemos sacar la conclusión de que **S01motd** recibirá atención en primer lugar, seguido por **S13rpcbind**, y así sucesivamente. Si hay dos o más que compartan el mismo prefijo **SXX**, el orden de inicio se decide por orden alfabético. De esta manera, **S16rsyslog** será llamado primero que **S16virtualbox-guest-utils** (este último no se muestra en la imagen).

Con el correr del tiempo, las siguientes limitaciones se hicieron notorias en este esquema:

- El proceso de inicio de servicios es estrictamente síncrono (se espera a que un servicio esté corriendo antes de iniciar el próximo). Esto impacta en el tiempo total de inicio del sistema.
- Necesidad de chequear dependencias antes de iniciar un servicio. Por ejemplo, que el servicio de red esté disponible antes de iniciar un servidor web.
- Cualquier evento posterior al inicio del equipo (identificación y montaje de dispositivos extraíbles, por ejemplo) necesitan intervención manual del usuario.

Fueron precisamente estas desventajas que condujeron al desarrollo de otros sistemas de arranque y administración de servicios como **Upstart** primero y **systemd** después.

Upstart

Entre **SysVinit** y la adopción final de **systemd** por la mayoría de las distribuciones principales GNU/Linux, surgió una alternativa conocida como **Upstart**. Fue desarrollada por la empresa **Canonical** (desarrolladora de Ubuntu) e integrada por primera vez con **Ubuntu 6.10 Edgy** a fines de 2006. Posteriormente, Fedora la adoptó y utilizó hasta la versión 14 inclusive. Hoy en día podemos encontrarla en RHEL 6.7 y similares, los cuales gozan de soporte hasta fines de 2018.

Upstart se pensó como un reemplazo basado en eventos para **SysVinit**. Como tal, supervisa tareas mientras el sistema está funcionando y responde a eventos tales como la conexión o desconexión de dispositivos extraíbles. Además, también gestiona los servicios durante el inicio y el apagado. Es importante notar que es 100% compatible con los scripts clásicos de **SysVinit**. De esta manera, aquellos servicios que provean un script para **init** pueden funcionar sin problemas bajo **Upstart**.

Por otro lado, **Upstart** también trabaja con archivos **.conf** dentro del directorio **/etc/init**. En los mismos se define el funcionamiento de servicios, y tienen la siguiente estructura:

- Descripción del proceso
- Niveles de corrida en los cuales el proceso debe ejecutarse o eventos que deben iniciarlo.

- Niveles de corrida en los cuales el proceso no debe correr o eventos que deben detenerlo.
- Opciones
- Comando a utilizar para lanzar el proceso

Consideremos el archivo de ejemplo **backup.conf** que mostramos a continuación. Los comentarios presentes en el mismo (en líneas que comienzan con #) nos sirven como guía para entender el funcionamiento del servicio que utiliza este archivo de configuración):

```
# Servicio de prueba para Upstart
# Stanzas
# Stanzas define when and how a process is started and stopped
# See a list of stanzas here: http://upstart.ubuntu.com/wiki/Stanzas#respawn

# Runlevels para iniciar el servicio
start on runlevel [2345]
# Cuando detenerlo
stop on runlevel [016]
# Reiniciarlo en caso de falla
respawn
# Especificar directorio de trabajo
chdir /home/alumno/misarchivos
# Indicar el comando, junto con cualquier argumento necesario (opcional)
exec bash backup.sh arg1 arg2
```

A pesar de la aceptación que este sistema tuvo en la comunidad, con el tiempo cedió lugar al más robusto **systemd**. Incluso Canonical decidió adoptarlo para Ubuntu luego de que el proyecto Debian hiciera lo mismo a partir de la versión 8 (Debian Jessie).

Systemd

Es importante aclarar que **systemd** no surgió como un reemplazo de **SysVinit** porque este último fuera *defectuoso* o porque hubiera usuarios y administradores que estuvieran descontentos con el mismo. Más bien, comenzó como un proyecto para desarrollar un sistema que fuera más eficiente al **1)** iniciar menos servicios durante el arranque (solamente aquellos que fueran estrictamente necesarios de acuerdo al uso esperado y al hardware disponible), y **2)** hacerlo en paralelo, en vez de manera secuencial. En otras palabras, se buscó un sistema de inicio y de administración de servicios que pudiera reaccionar *dinámicamente* ante cambios en el software y en el hardware.

Si lo pensamos bien, esta es una característica distintiva de los sistemas de cómputo actuales. Por ejemplo, no hay necesidad de mantener el servicio de impresión corriendo si no hay ninguna impresora conectada

al equipo o disponible a través de la red. Por otra parte, queremos que se inicie de manera transparente si esa condición cambia.

A pesar de las bondades mencionadas en el párrafo anterior, muchas personas se opusieron tenazmente a la adopción de **systemd** por parte de sus distribuciones favoritas. Sin embargo, las reservas de la comunidad eventualmente cedieron, resultando en la adopción de **systemd** en Fedora -lo cual derivó en Red Hat Enterprise Linux 7 y eventualmente en CentOS 7-, ArchLinux, Debian, Ubuntu, y derivados de estos últimos.

Targets en systemd

Bajo **systemd**, el concepto de *runlevel* o *nivel de corrida* solamente se mantiene por compatibilidad con **SysVinit**. El componente básico de systemd se denomina *unidad* (**unit**). Existen varias categorías de unidades: las más conocidas se denominan *servicios* (**services**) y *objetivos* (**targets**). Las primeras son las que ejecutan los *daemons* y sus dependencias en el orden apropiado. Por otro lado, las segundas se utilizan para agrupar varias unidades. Estas últimas pueden compararse en cierta forma a los clásicos runlevels.

Dentro de **/lib/systemd/system** podemos encontrar las definiciones de las distintas unidades. Para empezar, cabe aclarar que los archivos de configuración de los *targets* llevan el sufijo **.target**. Por ejemplo, podemos encontrar **basic.target** y **multi-user.target**. Este último es el objetivo donde se encuentran agrupados la mayoría de los *daemons*. Para funcionar correctamente, se necesita que **basic.target** esté activado. Dicho de otra forma, antes de entrar en **multi-user.target**, todos los servicios agrupados en **basic.target** deben haberse iniciado. A su vez, **basic.target** requiere **sysinit.target**.

*El objetivo **basic.target** en esencia cubre todo lo relacionado con el inicio del sistema. Luego, podemos decir lo siguiente en general: Si en **A.target** se indica **Requires=B.target**, hablamos que el objetivo **B** es una dependencia del **A**.*

Además, en **systemd** hablamos de *objetivo por defecto* para describir el estado que alcanza el sistema al finalizar el proceso de arranque. Para ver cuál es el objetivo por defecto en nuestro sistema, podemos utilizar el comando

```
systemctl get-default
```

¿De dónde proviene este dato? Se trata de un enlace simbólico llamado **default.target** dentro de **/lib/systemd/system** que apunta a la definición del *target* indicado. Por ejemplo, en la Fig. 1 podemos ver que **/lib/systemd/system/default.target** es un *soft link* hacia **/lib/systemd/system/graphical.target**:

```

root@stretch:~# ls -l /lib/systemd/system/default.target
lrwxrwxrwx 1 root root 16 jul  5 2017 /lib/systemd/system/default.target -> graphical.target
root@stretch:~# cat /lib/systemd/system/default.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
root@stretch:~#

```

Definición del objetivo
por defecto

Por supuesto, es posible cambiar el *target* por defecto de acuerdo a nuestras necesidades. Más adelante en el curso aprenderemos cómo hacerlo.

El proceso de inicio de sesión

Cuando ingresamos un usuario y su correspondiente contraseña para iniciar sesión en el sistema, esta información se compara con la disponible en los archivos **/etc/passwd** y **/etc/shadow**. En el primero de ellos encontramos información de cada cuenta de usuario y en el segundo las contraseñas *hasheadas* de las cuentas que tienen permitido el inicio de sesión. Por lo general, en distribuciones modernas las contraseñas se protegen utilizando el algoritmo [SHA-512](#). Este método dificulta muchísimo el adivinar una contraseña a partir de la cadena de texto que la representa.

Si la contraseña que ingresamos ha sido aceptada por el sistema nos aparece el contenido del archivo **/etc/motd** que suele estar vacío pero que le sirve al administrador para dejar mensajes dirigidos a los usuarios con información pertinente, como por ejemplo:

Hoy a las 20:00 horas se efectuará un backup. Les agradezco dejar todas las aplicaciones cerradas.

A continuación, el sistema operativo nos devuelve el *prompt*, o la línea de comandos preparada para recibir nuestras órdenes. En el caso de Debian el *prompt* inicial está compuesto por dos partes:

- El nombre del usuario actual, seguido del símbolo **@**.
- El nombre del equipo.
- El directorio inicial de trabajo, más conocido como el *home* o el directorio personal del usuario en cuestión. El símbolo **~** se utiliza para representar este directorio.
- Si estamos logueados como root, veremos el símbolo **#** a continuación. De otra manera (usuario común), se mostrará el signo **\$**.

Además de su directorio personal, cada usuario tiene asignado un *shell* o *intérprete de comandos*. Se trata de un programa que *recibe* los comandos que escribimos y que los *envía* al sistema operativo para ser ejecutados.

El *intérprete* utilizado en la mayoría de las distribuciones actuales (de las cuales Debian no es la excepción) se llama **Bash** (**B**ourne-**A**gain **S**hell). **Bash** tiene la posibilidad de ejecutar comandos en tiempo real pero además tiene un poderoso lenguaje de programación de scripts. Permite generar programas con funciones, control de flujo, creación de archivos, seguimiento de procesos, entre otros.

El modo texto

El modo texto es la interfaz más interesante para comenzar nuestro aprendizaje de Linux. Sin desmerecer las muy bien acabadas interfaces gráficas actuales, la primera nos permite interactuar con el sistema operativo desde el principio y muy fácilmente.

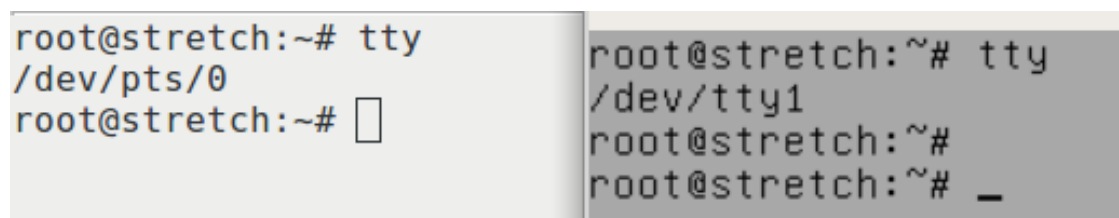
Para empezar, veamos a qué le llamamos *consola* en Linux. En palabras simples, una consola es una terminal física conectada directamente a una máquina. Por otro lado, una *terminal* (también llamada **tty**) es un recurso del sistema donde se pueden escribir comandos y ver el resultado de los mismos. Hoy en día asociamos las terminales con tty1, tty2, hasta tty6, que están disponibles para ser utilizadas por distintos usuarios al presionar la combinación de teclas Ctrl+Alt+F1, Ctrl+Alt+F2, y así sucesivamente (¡recordemos que Linux es un sistema operativo multiusuario y multitarea!).

Si en vez de trabajar en el modo texto exclusivamente también hemos instalado una distribución con un entorno gráfico, dispondremos de una aplicación llamada **Terminal**. Este programa nos permite acceder a la línea de comandos mediante una interfaz gráfica, por lo que recibe el nombre de *pseudo terminal* o **pts**.

En la siguiente imagen utilizamos el comando

tty

para observar dos sesiones abiertas del usuario root. Una de ellas está utilizando la **tty1** y la otra **pts0**.



```
root@stretch:~# tty
/dev/pts/0
root@stretch:~# ☐

root@stretch:~# tty
/dev/tty1
root@stretch:~#
root@stretch:~# _
```

Los primeros comandos

Después de la introducción anterior, ya es hora de comenzar a utilizar la línea de comandos. Con el correr de las clases veremos que se convertirá en nuestra mejor amiga como *sysadmins*.

Antes de finalizar, veamos una lista de tres comandos esenciales para navegar el sistema:

- `pwd` para mostrar el nombre del directorio actual de trabajo (el lugar dentro de la estructura de directorios donde estemos posicionados en un momento dado).
- `ls` para mostrar los contenidos de un directorio.
- `cd` para cambiar de directorio.

Para empezar, ejecutemos los dos primeros y veamos el resultado:

```
alumno@stretch:~$ pwd
/home/alumno
alumno@stretch:~$ ls
adm.txt          archivo2          dato
alumnosconectados.txt  archivo3          ejem
archivo1         claseloperador.txt  for-
```

Como podemos apreciar, `pwd` devolvió **/home/alumno** como resultado. Esto significa que en este momento nos hallamos posicionados dentro de ese directorio.

Recordemos del capítulo 2 que **alumno** es un subdirectorio de **/home**, que a su vez se encuentra dentro de **/**. Para verificarlo, podemos cambiar de directorio a **/** y al listar sus contenidos, veremos allí al directorio **home**. Si a continuación nos movemos a este último encontraremos el subdirectorio **alumno** como era de esperarse:

```
alumno@stretch:~$ cd /
alumno@stretch:/$ ls
archivos  boot      curso  etc      general  ini
bin       comunes  dev    finanzas home     ini
alumno@stretch:/$ cd home
alumno@stretch:/home$ ls
alumno  fulano  gabriel  lost+found  pruebasamba
```

Como vemos en los ejemplos anteriores, para cambiar de directorio utilizamos el comando `cd` seguido de la ubicación a la que deseamos movernos. Lo mismo aplica a `ls`, que no solamente nos permite ver los contenidos de la ubicación actual sino también la de cualquier directorio del sistema (si es que contamos con los permisos para poder acceder al mismo, tema que trataremos más adelante). Por último, sin importar el directorio en el que estemos parados, podemos utilizar `cd a secas` para regresar a nuestro *home* o `cd -` para dirigirnos al último directorio en el que

estuvimos. Este truco es útil para evitar tener que escribir una ruta completa para movernos a la misma.

En todos los comandos que requieren el uso de una ubicación dentro de la estructura de directorios, esta puede especificarse de dos maneras:

- *Ruta absoluta* es la que comienza en / y va hasta el directorio o archivo en cuestión. Por ejemplo, para hacer referencia al archivo **adm.txt** dentro del *home* del usuario **alumno** utilizamos **/home/alumno/adm.txt**.
- *Ruta relativa* es la que parte de la ubicación actual. Por ejemplo, si estamos parados en **/home** podemos movernos al *home* de alumno con el comando `cd alumno` en vez de `cd /home/alumno` (que sería el equivalente utilizando la ruta absoluta).