Administración de Servidores

Arranque del Sistema

Personalización del sistema de arranque

Creado por

Iñaki Fernández de Viana y González

Huelva, octubre 2021

Sobre Nosotros

Iñaki Fernández de Viana y González

Despacho 128. Escuela Técnica Superior de Ingeniería

Dpto. De Tecnologías de la Información

i.fviana@dti.uhu.es

+34 959217378

Objetivos

- **Peso:** 3
- **Descripción:** Candidates should be able to query and modify the behaviour of system services at various targets / run levels. A thorough understanding of the systemd, SysV Init and the Linux boot process is required. This objective includes interacting with systemd targets and SysV init run levels.

Objetivos (II)

- **Áreas clave de conocimiento:** Systemd; SysV init; Linux Standard Base Specification (LSB)

Objetivos (III)

- **Términos y utilidades:** /usr/lib/systemd/ ; /etc/systemd/; /run/systemd/; systemctl ; systemd-delta ; /etc/inittab ; /etc/init.d/ ; /etc/rc.d/ ; chkconfig: update-rc.d; init and telinit

Índice

Introducción

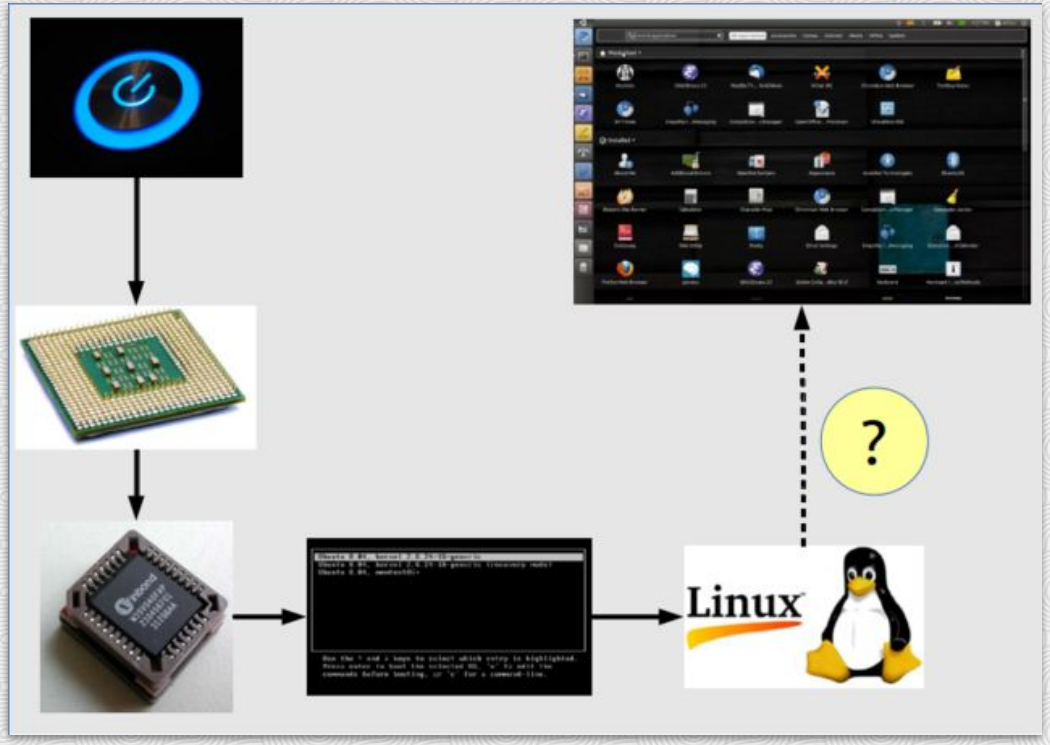
1

Introducción

- El proceso de arranque de un sistema operativo es un proceso complejo.
- Es una secuencia de eventos altamente personalizable.
- Es importante conocer esta secuencia para:
  - Poder solucionar problemas en el proceso de arranque
  - Modificar el proceso de arranque
  - Conocer las características y servicios que se inician durante el proceso de arranque.

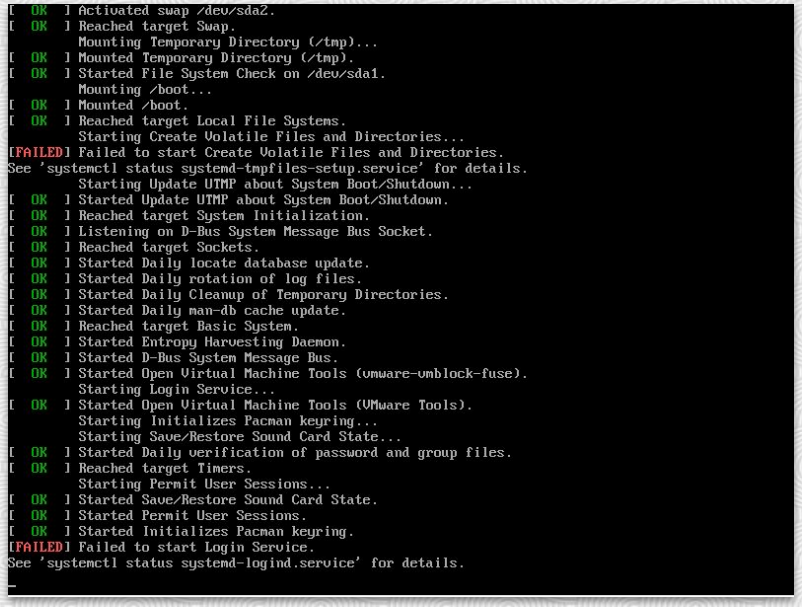
Introducción (II)

★



Introducción (III)

★ Una vez que se carga el núcleo Linux, comienzan a ejecutarse distintos servicios y demonios.



Introducción (IV)

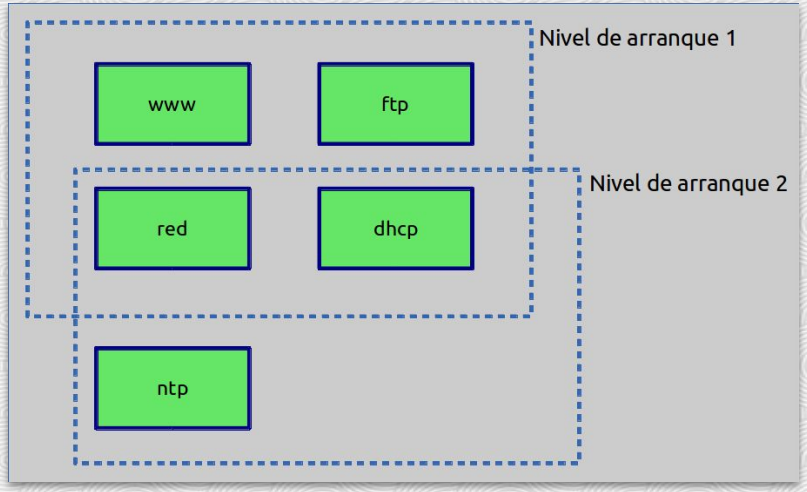
- ★ Qué y cómo arrancan éstos dependen de una serie de ficheros de configuración.
- ★ Existen tres tecnologías para gestionar este proceso de arranque:
  - SysV, *“el viejo”*.
  - Upstart, *“el olvidado”*.
  - Systemd, *“el controvertido”*

SysV-init

2

Niveles de arranque

- En un sistema Linux que use SysV-init como tecnología de arranque, el primero proceso (PID 1) que se ejecuta se denomina `init`.
- Este proceso es el encargado de arrancar una serie de servicios:



Niveles de arranque (II)

- ★ Un nivel de arranque (*runlevel*) es un estado funcional de un sistema operativo.
- ★ Se pueden establecer las características y servicios disponible en cada nivel de arranque
- ★ Hay tres conjuntos distintos de *runlevels*:
  - ○ Linux Standard Base Set
  - ○ Red Hat Set
  - ○ Debian Set

Niveles de arranque (III)

Nivel	LSB	Red Hat	Debian
0	Para el sistema	Para el sistema	Para el sistema
1,	Modo Monousuario	Modo Monousuario	Modo Monousuario
2	Multiusuario sin red ni GUI	Multiusuario sin NFS ni GUI	2-5 son multiusuario con GUI
3	Multiusuario sin GUI	Multiusuario sin GUI	
4	No definido	Igual que el 3	
5	Multiusuario con GUI	Multiusuario con GUI	
6	Reinicia el sistema	Reinicia el sistema	Reinicia el sistema

Niveles de arranque (IV)

- ★ Podemos definir los runlevels 7,8 y 9 aunque raramente se hace
- ★ Suele existir el nivel de arranque `single`, `s` o `S` que es similar o igual al nivel 1.
- ★ Muchas distribuciones personalizan los niveles de arranque

El fichero /etc/inittab

★ Para determinar el runlevel que se debe alcanzar, `init` consulta el fichero `/etc/inittab`.

```
id:2:initdefault:
si::sysinit:/etc/rc.sysinit
~:S:wait:/sbin/sulogin
l0:0:wait:/etc/rc 0
l1:1:wait:/etc/rc 1
l2:2:wait:/etc/rc 2
l3:3:wait:/etc/rc 3
l4:4:wait:/etc/rc 4
l5:5:wait:/etc/rc 5
l6:6:wait:/etc/rc 6
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
1:23:respawn:/sbin/mingetty tty1
```

El fichero /etc/inittab (II)

★ Cada entrada del fichero `/etc/inittab` tiene la siguiente sintaxis:

```
id:runlevels:action:process
```

★ `id`: Identificador de hasta 4 caracteres

★ `runlevels`: Lista de los niveles en los que se ejecutará

★ `action`: cómo se tiene que usar esta línea

★ `process`: qué comando se va a ejecutar

```
[...]
l2:2:wait:/etc/rc 2
l6:6:wait:/etc/rc 6
1:23:respawn:/sbin/mingetty tty1
[...]
```

El fichero /etc/inittab (III)

- En el fichero `/etc/inittab` podemos encontrar distintas palabras clave.
- `initdefault`: Identificador de hasta 4 caracteres
- `sysinit`: La línea se ejecutará al arrancar el sistema
- `wait`: init espera a que el comando se ejecute por completo
- `ctrlaltdel`: la linea se ejecutar cuando se pulsa la combinación de teclas.
- `powerfail`: la línea se ejecuta cuando se recibe señal de un SAI
- `powerokwait`: la línea se ejecuta cuando se recibe señal de un SA
- `respawn`: init volverá a ejecutar el comando si este finaliza.

Scripts de arranque

- El proceso init invoca a una serie de scipts para que el sistema alcance el nivel definido:
  - El `/etc/rc.d/rc.sysinit` se ejecuta para llevar al sistema a un estado base inicial (similar al nivel 1)
  - El `/etc/rc.d/rc` (o `/etc/init.d/rc`) se ejecuta cuando el sistema va a entrar en el nivel 1 a 6.

/etc/rc.d/rc

- El `/etc/rc.d/rc` se ejecuta cuando el sistema va a entrar en el nivel 1 a 6.
- Es invocado por el proceso `init` tal y como se indica en el `/etc/inittab`, se le pasa el nivel de arranque.

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

/etc/rc.d/rc (II)

★ Parte del código de `/etc/rc.d/rc` es:

```
for i in /etc/rc$runlevel.d/K* ; do
    [...]
        Action $"Stopping $subsys: " $i stop
    [...]

For i in /etc/rc$runlevel.d/S* ; do
    [...]
    Action $"Starting $subsys: " $i start
    [...]
```

★ La variable `$runlevel` indica el nivel pasado como parámetro

★ Ejecuta los ficheros `K*` y `S*` localizados en `/etc/rc?.d/`

Los directorios /etc/rc?.d/

- Los directorios `/etc/rc?.d/` (donde `?` es un nivel de arranque) contienen una serie de scripts que serán invocados (en orden) por `/etc/rc.d/rc`
- Los que empiezan por `S` recibirán como parámetro `start` y los que empiezan por `K` el parámetro `stop`:

```
/etc/rc2.d$ ls
K16sslh     K20vboxdrv  S20vboxbaoonctrl-service   S20hddtemp
K50rsync    K50saned    K70dns-clean               K95preload
S16sslh     S20vboxdrv  S20vboxbaoonctrl-service   S20hddtemp
S50rsync    S50saned    S70dns-clean               S95preload
```

Los directorios /etc/rc?.d/ (II)

★ Realmente estos scripts son enlaces simbólicos a ficheros contenidos en `/etc/init.d`

```
/etc/rc2.d$ ls -l
lrwxrwxrwx 1 root root  14 mar 19  2013 S16sslh  --> /init.d/sslh
```

★ Eliminando o añadiendo enlaces podemos establecer lo que se ejecutar en cada nivel de arranque.

El directorio /etc/init.d/

★ Contiene los scripts de arranque los distintos servicios.

```
#!/bin/bash
#
# chkconfig: 35 90 12
# description: Foo server
#
# Get function from functions library
. /etc/init.d/functions
# Start the service FOO
start() {
        initlog -c "echo -n Starting FOO server: "
        /path/to/FOO & && touch /var/lock/subsys/FOO
        success $"FOO server startup"
        echo
}
```

El directorio /etc/init.d/ (II)

```
status() {
        initlog -c "echo -n Starting FOO server: "
        /path/to/FOO &
        touch /var/lock/subsys/FOO
        success $"FOO server startup"
        echo
}
stop() {
        initlog -c "echo -n Stopping FOO server: "
        killproc FOO
        ### Now, delete the lock file ###
        rm -f /var/lock/subsys/FOO
        echo
}
```

El directorio /etc/init.d/ (III)

```
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status FOO
        ;;
  restart|reload|condrestart)
        stop && start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|reload|status}" && exit 1
esac
```

Invocando a los scripts

★ Podemos invocar a los scripts contenidos en `/etc/init.d` indicando la ruta completa y pasándole un parámetro:

```
[root@localhost:/]$ /etc/init.d/apache2 start
```

★ Lo ideal es usar el comando service tanto para arrancar un servicio:

```
[root@localhost:/]$ service apache2 start
```

★ Parar un servicio

```
[root@localhost:/]$ service apache2 stop
```

★ Comprobar el estado de un servicio

```
[root@localhost:/]$ service apache2 status
```

Modificando runlevels

- Si queremos modificar los servicios que se ejecutan en un runlevel, basta añadir los enlaces oportunos `S*` y `K*` en el `/etc/rcX.d` oportuno.
- Antes de eso nos tenemos que asgurar que el script al que apuntan existe en `/etc/init.d`
- Las principales distribuciones, automatizan este proceso gracias a las utilidades:
  - chkconfig
  - update-rc

chkconfig

- La utilidad chkconfig, permite modificar la configuración de los distintos runlevel para distribuciones Red Hat.
- Si queremos consultar los servicios y los niveles en los que están activos usamos la opción `--list`:

```
[root@localhost:/]$ chkconfig --list
pcmcia          0:off 1:off 2:on   3:on     4:on 5:on 6:off
Nfs-common      0:off 1:off 2:off  3:on     4:on 5:on 6:off
Xprint          0:off 1:off 2:off  3:on     4:on 5:on 6:off
Setserial       0:off 1:off 2:off  3:off    4:off 5:off 6:off
[root@localhost:/]$ chkconfig --list nfs-common
nfs-common      0:off 1:off 2:off  3:on     4:on 5:on 6:off
```

chkconfig (II)

★ Para poder añadir o quitar un servicio a un nivel usamos las opciones `--level` (para indicar los niveles) y `on` un `off`:

```
[root@localhost:/]$ chkconfig --list nfs-common
nfs-common    0:off 1:off 2:off  3:on     4:on 5:on 6:off
[root@localhost:/]$ chkconfig --levels 23 nfs-common on
[root@localhost:/]$ chkconfig --list nfs-common
nfs-common    0:off 1:off 2:on   3:on     4:on 5:on 6:off
[root@localhost:/]$ chkconfig --levels 23 nfs-common off
[root@localhost:/]$ chkconfig --list nfs-common
nfs-common    0:off 1:off 2:off  3:off    4:on 5:on 6:off
```

★ Para que un script pueda ser gestionado por chkconfig antes hay que indicárselo usado `--add (chkconfig --add nfs-common)`

update-rc.d

- La utilidad update-rc.d, permite modificar la configuración de los distintos runlevel para distribuciones Debian.
- Si queremos añadir/quitar un servicio usamos las opciones `remove` o no indicamos opción:

```
[root@localhost:/]$ update-rc.d -f apache2 remove
[root@localhost:/]$ update-rc.d apache2 defaults
```

- Si queremos indicar el orden de los scripts de arranque y parada:

```
[root@localhost:/]$ update-rc.d apache2 defaults 90 90
[root@localhost:/]$ update-rc.d mysql defaults 10 10
```

update-rc.d (II)

★ Si queremos añadir un servicio a un determinado nivel usamos la opción `start` o `stop` seguida de los niveles de arranque:

```
[root@localhost:/]$ update-rc.d apache2 start 2345
[root@localhost:/]$ update-rc.d apache2 stop 016
```

★ También podemos indicar la prioridad:

```
[root@localhost:/]$ update-rc.d apache2 start 90 2345 . stop  50 016 .
```

Cambiando de runlevel

★ En SysV, el nivel por defecto lo determina la directiva `initdefault` del fichero `/etc/inittab`:

```
[root@localhost:/]$ grep :initdefault: /etc/inittab
id:5:initdefault:
```

★ Podemos consultar el nivel actual mediante la orden `runlevel`:

```
[root@localhost:/]$ runlevel
N2
```

★ El número de la izquierda indica el nivel anterior, la letra N de la derecha indica que no se ha cambiado de nivel desde el arranque.

Cambiando de runlevel (II)

- Para cambiar de runlevel tenemos las siguientes opciones:
  - Init o telinit
  - Shutdown
  - Halt, reboot o power off
- `shutdown`, `halt`, `reboot` y `poweroff` nos permiten llevar el sistema a los niveles 0 (parada) o 7 (reinicio)

Init y telinit

★ El comando `init` se puede usar para cambiar el nivel de arranque actual:

```
[root@localhost:/]$ init 1
N2
ijfviana@valagume:~$ runlevel
S21
ijfviana@valagume:~$ init 6
```

★ El comando `telinit` es similar a `init`. Su opción `-Q` o -q provoca que se relean el fichero /etc/inittab

★ En casi todas las distribuciones modernas telinit e init son el mismo comando.

Upstart

3

Introducción

- ★ upstart está desarrollado por ubuntu y tiene como objetivos
  - Equivalente y es compatible con SysV
  - Gestiona mejor el hardware plug-ing
- ★ La última versión es la 1.13 liberada el 11 de julio del 2014.

Diferencias cons SysV

- ★ No existe el fichero `/etc/inittab`
- ★ En el directorio `/etc/init` se encuentran los ficheros de arranque de los servicios
- ★ Estos scripts los ejecuta `/lib/init/upstart-job`

```
$ ls
acpid.conf            cups.conf          module-init-tools.conf
alsa-restore.conf     dbus.conf          mountall.conf
alsa-store.conf       dmesg.conf         mountall-net.conf
anacron.conf          failsafe.conf      mountall-reboot.conf
```

Diferencias cons SysV (II)

★ Los scripts de arranque son de las forma siguiente:

```
description     "CUPS printing spooler/server"
author          "Michael Sweet <msweet@apple.com>"

start on (filesystem
          and (started dbus or runlevel [2345]))
stop on runlevel [016]

respawn
respawn limit 3 12

pre-start script
    [ -x /usr/sbin/cupsd ]
```

Diferencias cons SysV (III)

```
    if [ -r /etc/default/cups ]; then
    . /etc/default/cups
    fi
    if [ "$LOAD_LP_MODULE" = "yes" -a -f
/usr/lib/cups/backend/parallel \
     -a -f /proc/modules -a -x /sbin/modprobe ]; then
    modprobe -q -b lp || true
    modprobe -q -b ppdev || true
    modprobe -q -b parport_pc || true
    fi
    mkdir -p /var/run/cups/certs
    if [ -x /lib/init/apparmor-profile-load ]; then
    /lib/init/apparmor-profile-load usr.sbin.cupsd
    fi
end script
```

Diferencias cons SysV (IV)

```
exec /usr/sbin/cupsd -F

post-start script
    timeout=6
    while [ ! -e /var/run/cups/cups.sock ]; do
        sleep 0.5
    timeout=$((timeout-1))
    if [ "$timeout" -eq 0 ]; then
        echo "cupsd failed to create /var/run/cups/cups.sock,
skipping automatic printer configuration" >&2
        exit 0
    fi
    done
    if ! /lib/udev/udev-configure-printer enumerate 2>/dev/null;
then
```

Diferencias cons SysV (IV)

- ★ Hay que ejecutar `initctl` tras modificar cualquier script
- ★ upstart no está del todo extendido por lo que tiene un modo de compatibilidad con SysV ejecutando los scipts de SysV

```
:/etc$ ls -ld rc* init.d/
drwxr-xr-x 2 root root 4096 oct 20 22:52 init.d/
drwxr-xr-x 2 root root 4096 sep 19 10:23 rc0.d
drwxr-xr-x 2 root root 4096 jul  7 21:15 rc1.d
drwxr-xr-x 2 root root 4096 jul  7 21:15 rc2.d
drwxr-xr-x 2 root root 4096 jul  7 21:15 rc3.d
drwxr-xr-x 2 root root 4096 jul  7 21:15 rc4.d
drwxr-xr-x 2 root root 4096 jul  7 21:15 rc5.d
drwxr-xr-x 2 root root 4096 sep 19 10:23 rc6.d
-rwxr-xr-x 1 root root 3033 mar 25  2013 rc.local
drwxr-xr-x 2 root root 4096 sep 19 10:23 rcS.d
```

systemd

4

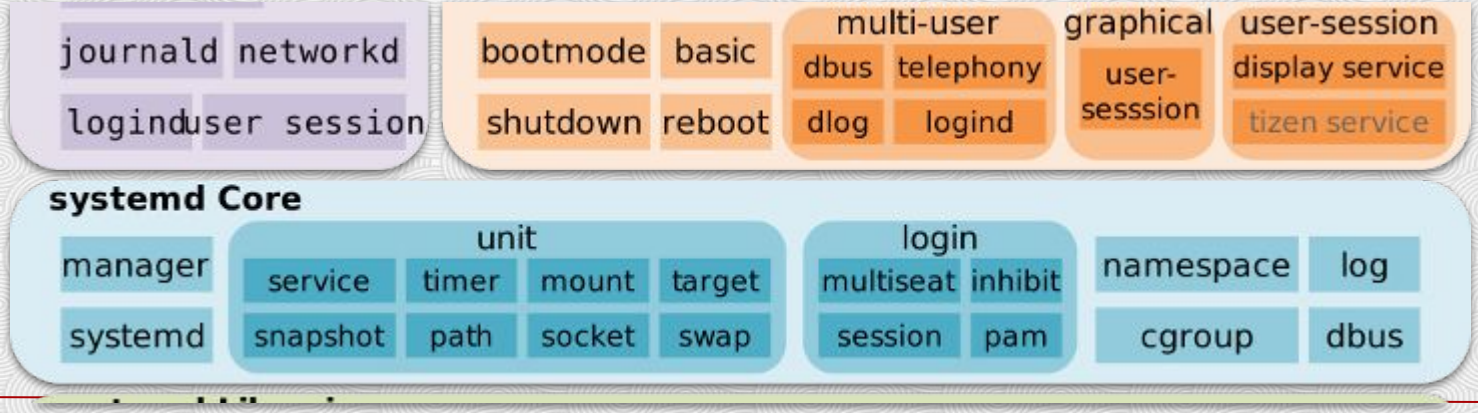
Introducción

- ★ `systemd` proporciona un gestor del sistema y servicios que se ejecuta como PID 1 y arranca el resto del sistema.
- ★ Se caracteriza por:
  - Es altamente paralelizable.
  - Usa *sockets* y activación `D-Bus` para arrancar los servicios.
  - Ofrece ejecución de demonios bajo demanda.
  - Seguimiento a los procesos usando Linux control groups.
  - Mantiene los puntos de montaje y automontaje.
  - Aplica una elaborada lógica de control de servicios basada en la dependencia de las transacciones.

Introducción

★ (continuación) Se caracteriza por:

- Soporta los *scripts* de inicio de SysV y LSB
- Log de demonios
- Utilidades básicas de gestión (hostname, date, locale, usuarios logeados, cuentas del sistema...),



Introducción

- ★ El comando principal para interaccionar, gestionar y controlar el estado de un sistema `systemd` es `systemctl`.
- ★ Con este comando podemos, entro otras cosas, consultar el estado global de sistema, de una **unidad** en particular, interaccionar con las unidades, gestionar los **target**….
- ★ Otra utilidad importante es `journalctl` que nos permite acceder a los registros de los log del sistema.

Targets

- ★ `systemd` utiliza **targets** («objetivos») que sirven a un propósito similar a los runlevels («niveles de ejecución»),
- ★ Cada **target** se nomina, en lugar de numerarse, y está destinado a servir a un propósito específico con la posibilidad de realizar más de una acción al mismo tiempo.
- ★ Algunos **targets** son activados heredando todos los servicios de otro target e implementando servicios adicionales.
- ★ Hay distintos tipos de **targets**: los que imitan los runlevels de SystemVinit, global de sistemas, de una unidad en particular o interaccionar con las unidades.

Gestión (I)

★ Para conocer los distintos targets disponibles ejecutamos `systemctl list-units --type=target`

```
$ systemctl --failed
  UNIT                   LOAD   ACTIVE SUB  DESCRIPTION
  basic.target           loaded active active Basic System
  bluetooth.target       loaded active active Bluetooth Support
  cryptsetup.target      loaded active active Local Encrypted Volumes
  getty.target           loaded active active Login Prompts
  graphical.target       loaded active active Graphical Interface
  local-fs-pre.target    loaded active active Preparation for Local File Systems
  local-fs.target        loaded active active Local File Systems
  multi-user.target      loaded active active Multi-User System
  network-online.target  loaded active active Network is Online
  network-pre.target     loaded active active Preparation for Network
```

Gestión (II)

- Podemos saber el *target* en el que estamos si ejecutamos `systemctl get-default`

```
$ systemctl get-default
graphical.target
```

- Podemos cambiar target si ejecutamos `systemctl isolate xxxx.target`

```
$ systemctl isolate graphical.target
```

Gestión (III)

★ Si queremos cambiar el target con el que arrancará por defecto el sistema `systemctl set-default multi-user.target`:

```
# systemctl set-default multi-user.target
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target ->
/usr/lib/systemd/system/graphical.target.
```

★ A la hora de elegir el `default.target`, `systemd` sigue la secuencia:

a. El parámetro que le pasemos al kernel
b. Enlace simbólico `/etc/systemd/system/default.target`
c. Enlace simbólico `/usr/lib/systemd/system/default.target`

Crear un target personalizado (I)

★ Para crear un nuevo target personalizado, tenemos que crear el fichero oportuno en `/ets/systemd/system/`:

```
$ su vi /etc/systemd/system/foo.target
[Unit]
Description=Foobar boot target
Requires=multi-user.target
Wants=foobar.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target
AllowIsolate=yes
```

Crear un target personalizado (II)

★ Se debe crear un directorio en `/etc/systemd/system/<nombre_del_targer>.wants` que contiene los enlaces a las unidades que se desean arrancar:

```
$ sudo mkdir /etc/systemd/system/foo.target.wants
$ cd /etc/systemd/system/foo.target.wants
$ ln -s /etc/systemd/system/boo.service
```

Relación con SystemVinit

★

Runlevels

Comments	SysVinit	Systemd
System halt	0	runlevel0.target, poweroff.target
Single user mode	1, s, single	runlevel1.target, rescue.target
Multi user	2	runlevel2.target, multi-user.target
Multi user with Network	3	runlevel3.target, multi-user.target
Experimental	4	runlevel4.target, multi-user.target
Multi user, with network, graphical mode	5	runlevel5.target, graphical.target
Reboot	6	runlevel6.target, reboot.target
Emergency Shell	emergency	emergency.target
Change to multi user runlevel/target	telinit 3	systemctl isolate multi-user.target (OR systemctl isolate runlevel3.target)
Set multi-user target on next boot	sed s/^id:.*:initdefault:/id:3:initdefault:/	ln -sf /lib/systemd/system/multi-user.target /etc/systemd/system/default.target
Check current runlevel	runlevel	systemctl get-default
Change default runlevel	sed s/^id:.*:initdefault:/id:3:initdefault:/	systemctl set-default multi-user.target

Unidades

- En `systemd`, una **unidad** es un objeto que representa cualquier recurso que el sistema conoce cómo gestionarlo.
- Estos recursos se definen mediante ficheros de configuración denominados **unit files**.
- Estos ficheros tienen distintas extensiones (`.service`, `.mount`, `.device`, `.socket`) dependiendo del recurso que representen.
- Se guardan, y consultan, en `/usr/lib/systemd/system/` `/run/systemd/system/` y `/etc/systemd/system/`
- Las unidades son similares a los servicios o trabajos existentes en otros sistemas de arranque.

Tipos

- ★ **.service:** Describe cómo gestionar un servicio o aplicación .
- ★ **.socket:** Describes a network or IPC socket, or a FIFO buffer that systemd uses for socket-based activation. These always have an associated .service file that will be started when activity is seen on the socket that this unit defines.
- ★ **.device:** Describes a device that has been designated as needing systemd management by udev or the sysfs filesystem. Not all devices will have .device files. Some scenarios where .device units may be necessary are for ordering, mounting, and accessing the devices.

Tipos

- **.mount:** Defines a mountpoint on the system to be managed by systemd. These are named after the mount path, with slashes changed to dashes. Entries within /etc/fstab can have units created automatically.
- **.automount**: Configures a mountpoint that will be automatically mounted. These must be named after the mount point they refer to and must have a matching .mount unit to define the specifics of the mount.
- **.swap**: Describes swap space on the system. The name of these units must reflect the device or file path of the space.

Tipos

- ★ **.target**: Is used to provide synchronization points for other units when booting up or changing states. They also can be used to bring the system to a new state. Other units specify their relation to targets to become tied to the target's operations.
- ★ **.path**: Defines a path that can be used for path-based activation. By default, a .service unit of the same base name will be started when the path reaches the specified state..
- ★ **.timer**: Defines a timer that will be managed by `systemd`, similar to a cron job for delayed or scheduled activation. A matching unit will be started when the timer is reached.

Tipos

- ★ **.snapshot**: Is created automatically by the systemctl snapshot command. It allows you to reconstruct the current state of the system after making changes. Snapshots do not survive across sessions.
- ★ **.slice**: Is associated with Linux Control Group nodes, allowing resources to be restricted or assigned to any processes associated with the slice. The name reflects its hierarchical position within the cgroup tree. Units are placed in certain slices by default depending on their type.
- ★ **.scope**: Are created automatically by `systemd` from information received from its bus interfaces. These are used to manage sets of system processes that are created externally.

Analizar el estado del sistema

- ★ Si queremos consultar el estado del sistema tras su arranque, podemos ejecutar `systemctl status`.
- ★ Muestra el tiempo de ejecución, el estado global de los trabajos y/o unidades así como la relación de procesos lanzados.

```
$ systemctl status
● host
    State: running
    Jobs: 0 queued
  Failed: 0 units
    Since: Thu 2021-10-28 10:50:09 CEST; 5 days ago
  CGroup: /
          ├─user.slice
          │ └─user-1000.slice
```

Analizar el estado del sistema

★ Si queremos conocer el estado de las unidades activas ejecutamos `systemctl` o `systemct list-units`.

```
$ systemctl
UNIT                      LOAD   ACTIVE SUB     DESCRIPTION
accounts-daemon.service   loaded active running  Accounts Service
alsa-restore.service      loaded active exited   Save/Restore Sound Card State
anydesk.service           loaded active running   AnyDesk
bluetooth.service         loaded active running   Bluetooth service
colord.service            loaded active running   Manage, Install and Generate Color
Profiles
```

Analizar el estado del sistema

★ Si sólo nos interesan las unidades que han fallado indicaremos `systemctl --failed`.

```
$ systemctl --failed
UNIT                    LOAD   ACTIVE SUB      DESCRIPTION
0 loaded units listed.
```

★ Si nos interesa el listado de todas las unidades instaladas `systemctl list-unit-files`.

```
$ systemctl list-unit-files
UNIT FILE                   STATE      VENDOR PRESET
debug-shell.service         disabled      disabled
dhclient@.service           disabled      disabled
```

Gestión

- Para gestionar las unidades hay que indicar el nombre completo (incluir la extensión) pero:
  - Si no se especifica extensión se asumirá `.service`
  - Los puntos de montaje se traducen automáticamente `.mount`
  - Los dispositivos se traducen automáticamente a `.device`
- Para activar una unidad ejecutamos `systemctl start unidad`:

```
$ systemctl start httpd.service
```

- Para detener una unidad `systemctl stop unidad`:

```
$ systemctl stop httpd.service
```

Gestión

★ Para reiniciar la unidad `systemctl restart unidad`:

```
$ systemctl restart httpd.service
```

★ Para volver a leer la configuración `systemctl reload unidad`:

```
$ systemctl reload httpd.service
```

★ Para consultar el estado de una unidad: `systemctl status unidad`:

```
$ systemctl status httpd.service
```

★ Para que la unidad se inicie en el arranque `systemctl enable unidad`:

```
$ systemctl enable httpd.service
```

Gestión

- Para que la unidad no se inicie en el arranque `systemctl disable unidad`:

```
$ systemctl disable htttpd
```

- Para evitar que la unidad se inicie en cualquier caso `systemctl mask unidad`:

```
$ systemctl mask httpd
```

systemd vs SysVinit

Service Related Commands

Comments	SysVinit	Systemd
Start a service	service dummy start	systemctl start dummy.service
Stop a service	service dummy stop	systemctl stop dummy.service
Restart a service	service dummy restart	systemctl restart dummy.service
Reload a service	service dummy reload	systemctl reload dummy.service
Service status	service dummy status	systemctl status dummy.service
Restart a service if already running	service dummy condrestart	systemctl condrestart dummy.service
Enable service at startup	chkconfig dummy on	systemctl enable dummy.service
Disable service at startup	chkconfig dummy off	systemctl disable dummy.service
Check if a service is enabled at startup	chkconfig dummy	systemctl is-enabled dummy.service
Create a new service file or modify configuration	chkconfig dummy --add	systemctl daemon-reload

Creación de unidades

- Los ficheros de unidades se pueden almacenar en distintos sitios (`systemctl show --property=UnitPath`), principalmente `/usr/lib/systemd/system/` y `/etc/systemd/system/`
- Un ejemplo sencillo de fichero (+info):

```
[Unit]
Description=Foo
[Service]
Type=simple
ExecStart=/usr/sbin/foo-daemon
[Install]
WantedBy=multi-user.target
```

Taxonomía

- ★ The internal structure of unit files are organized with sections.
- ★ Sections are denoted by a pair of square brackets “[” and “]” with the section name enclosed within.
- ★ Each section extends until the beginning of the subsequent section or until the end of the file.
- ★ Within these sections, unit behavior and metadata is defined through the use of simple directives using a key-value format with assignment indicated by an equal sign,

Directivas de la sección Unit

- ★ `Description=`: This directive can be used to describe the name and basic functionality of the unit. It is returned by various systemd tools, so it is good to set this to something short, specific, and informative.
- ★ `Documentation=`: This directive provides a location for a list of URIs for documentation. These can be either internally available man pages or web accessible URLs. The systemctl status command will expose this information, allowing for easy discoverability.
- ★ `Requires=`: This directive lists any units upon which this unit essentially depends. These units are started in parallel with the current unit by default.

Directivas de la sección Unit

- ★ `Wants=`: This directive is similar to Requires=, but less strict. Systemd will attempt to start any units listed here when this unit is activated. If these units are not found or fail to start, the current unit will continue to function. This is the recommended way to configure most dependency relationships. Again, this implies a parallel activation unless modified by other directives.
- ★ `BindsTo=`: This directive is similar to Requires=, but also causes the current unit to stop when the associated unit terminates.ective.

Directivas de la sección Unit

- ★ `Before=`: The units listed in this directive will not be started until the current unit is marked as started if they are activated at the same time. This does not imply a dependency relationship and must be used in conjunction with one of the above directives if this is desired.
- ★ `After=`: The units listed in this directive will be started before starting the current unit. This does not imply a dependency relationship and one must be established through the above directives if this is required.
- ★ `Conflicts=`: This can be used to list units that cannot be run at the same time as the current unit. Starting a unit with this relationship will cause the other units to be stopped.

Directivas de la sección Unit

- ★ `Condition...=`: There are a number of directives that start with Condition which allow the administrator to test certain conditions prior to starting the unit. This can be used to provide a generic unit file that will only be run when on appropriate systems. If the condition is not met, the unit is gracefully skipped.
- ★ `Assert...=`: Similar to the directives that start with Condition, these directives check for different aspects of the running environment to decide whether the unit should activate. However, unlike the Condition directives, a negative result causes a failure with this directive.

Directivas de la sección Install

- ★ WantedBy=: is the most common way to specify how a unit should be enabled. This directive allows you to specify a dependency relationship in a similar way to the Wants= directive does in the [Unit] section. The difference is that this directive is included in the ancillary unit allowing the primary unit listed to remain relatively clean. When a unit with this directive is enabled, a directory will be created within /etc/systemd/system named after the specified unit with .wants appended to the end. Within this, a symbolic link to the current unit will be created, creating the dependency. For instance, if the current unit has WantedBy=multi-user.target, a directory called multi-user.target.wants will be created within /etc/systemd/system (if not already available) and a symbolic link to the current unit will be placed within. Disabling this unit

Directivas de la sección Install

- `RequiredBy=`: This directive is very similar to the WantedBy= directive, but instead specifies a required dependency that will cause the activation to fail if not met. When enabled, a unit with this directive will create a directory ending with .requires.
- `Alias=`: This directive allows the unit to be enabled under another name as well. Among other uses, this allows multiple providers of a function to be available, so that related units can look for any provider of the common aliased name.

Directivas de la sección Install

- ★ `Also=`: This directive allows units to be enabled or disabled as a set. Supporting units that should always be available when this unit is active can be listed here. They will be managed as a group for installation tasks.
- ★ `DefaultInstance=`: For template units (covered later) which can produce unit instances with unpredictable names, this can be used as a fallback value for the name if an appropriate name is not provided.

Directivas de la sección Service

- ★ `Type=`: can be one of the following:
  - simple: The main process of the service is specified in the start line.
  - forking: This service type is used when the service forks a child process, exiting the parent process almost immediately.
  - oneshot: This type indicates that the process will be short-lived and that systemd should wait for the process to exit.
  - dbus: This indicates that unit will take a name on the D-Bus bus. When this happens, systemd will continue to process the next unit.
  - notify:
  - idle:

Directivas de la sección Service

- ★ `RemainAfterExit=`: This directive is commonly used with the oneshot type. It indicates that the service should be considered active even after the process exits.
- ★ `PIDFile=`: If the service type is marked as “forking”, this directive is used to set the path of the file that should contain the process ID number of the main child that should be monitored.
- ★ `BusName=`: This directive should be set to the D-Bus bus name that the service will attempt to acquire when using the “dbus” service type.

Directivas de la sección Service

- ★ `NotifyAccess=`: This specifies access to the socket that should be used to listen for notifications when the “notify” service type is selected This can be “none”, “main”, or “all. The default, "none”, ignores all status messages. The “main” option will listen to messages from the main process and the “all” option will cause all members of the service’s control group to be processed.
- ★ `ExecStart=`: This specifies the full path and the arguments of the command to be executed to start the process. This may only be specified once (except for “oneshot” services). If the path to the command is preceded by a dash “-” character, non-zero exit statuses will be accepted without marking the unit activation as failed.

Directivas de la sección Service

- ★ `ExecStartPre=`: This can be used to provide additional commands that should be executed before the main process is started. This can be used multiple times. Again, commands must specify a full path and they can be preceded by “-” to indicate that the failure of the command will be tolerated.
- ★ `ExecStartPost=`: This has the same exact qualities as ExecStartPre= except that it specifies commands that will be run after the main process is started.
- ★ `ExecReload=`: This optional directive indicates the command necessary to reload the configuration of the service if available.

Directivas de la sección Service

- ★ `ExecStop=`: This indicates the command needed to stop the service. If this is not given, the process will be killed immediately when the service is stopped.
- ★ `ExecStopPost=`: This can be used to specify commands to execute following the stop command.
- ★ `RestartSec=`: If automatically restarting the service is enabled, this specifies the amount of time to wait before attempting to restart the service.

Directivas de la sección Service

- ★ `Restart=`: This indicates the circumstances under which systemd will attempt to automatically restart the service. This can be set to values like “always”, “on-success”, “on-failure”, “on-abnormal”, “on-abort”, or “on-watchdog”. These will trigger a restart according to the way that the service was stopped.
- ★ `TimeoutSec=`: This configures the amount of time that systemd will wait when stopping or stopping the service before marking it as failed or forcefully killing it. You can set separate timeouts with TimeoutStartSec= and TimeoutStopSec= as well.

Modificando unidades

- ★ Útil cuando queremos modificar “levemente” una unidad existente.
- ★ Dependiendo de la cantidad de cambios que queremos hacer podemos:
  - Crear un directorio con ficheros de configuración adicionales localizados en `/etc/systemd/system/unit.d/`. Es el método recomendado si introducimos muchos cambios
  - Crear una copia del fichero de la unidad contenido en `/usr/lib/systemd/system/` en `/etc/systemd/system/` y hacer los cambios oportunos.

Extender la definición de una unidad

- First create a configuration directory in `/etc/systemd/system/`
- Que se llamará como el servicio a modificar seguido de la extensión .d:

```
Mkdir /etc/systemd/system/httpd.service.d/
```

- En el directorio creamos un fichero acabado en .conf que contiene las modificaciones introducidas en la nueva definición de la unidad:

```
vi /etc/systemd/system/name.service.d/config_name.conf
[Service]
ExecStartPost=/usr/local/bin/custom.sh
```

Overriding the Default Unit Configuration

★ First copy the file to the /etc/systemd/system/

```
cp /usr/lib/systemd/system/name.service /etc/systemd/system/name.service
```

★ Modificamos el fichero creado:

```
vi /etc/systemd/system/name.service
[[]...]
[Service]
ExecStartPost=/usr/local/bin/custom.sh
[[]...]
```

★ Cualquier cambio en los ficheros de unidades se puede consulta usando `systemd-delta`.

Gestión de energía

Miscellaneous Commands

Comments	SysVinit	Systemd
System halt	halt	systemctl halt
Power off the system	poweroff	systemctl poweroff
Restart the system	reboot	systemctl reboot
Suspend the system	pm-suspend	systemctl suspend
Hibernate	pm-hibernate	systemctl hibernate
Follow the system log file	tail -f /var/log/messages or tail -f /var/log/syslog	journalctl -f

systemd vs SysVinit

Systemd New Commands

Comments	Systemd
Execute a systemd command on remote host	systemctl dummy.service start -H user@host
Check boot time	systemd-analyze or systemd-analyze time
Kill all processes related to a service	systemctl kill dummy
Get logs for events for today	Journalctl –since=today
Hostname and other host related information	hostnamectl
Date and time of system with timezone and other information	timedatectl

Analizando los procesos de inicio

- ★ Systemd permite analizar el orden y tiempo de todos los procesos implicados en el proceso de arranque.
- ★ Para ver la cantidad de tiempo que hay en el espacio del núcleo y el espacio de usuario en el arranque, basta con utilizar:

```
> systemd-analyze
Startup finished in 4.473s (firmware) + 3.188s (loader) + 1.305s (kernel) + 2.074s
(initrd) + 16.388s (userspace) = 27.430s
graphical.target reached after 16.345s in userspace
```

- ★ Si queremos consultar el detalle de las unidades que se han ejecutado y lo que tardo cada una:

```
>  systemd-analyze blame
```

Analizando los procesos de inicio

★ (continuación) Si queremos consultar el detalle de las unidades que se han ejecutado y lo que tardó cada una:

```
>  systemd-analyze blame
6.320s docker.service
5.647s dkms.service
4.877s plymouth-quit-wait.service
4.625s NetworkManager-wait-online.service
1.958s firewalld.service
1.804s systemd-udev-settle.service
1.461s fwupd.service
```

Analizando los procesos de inicio

★ Si nos interesa conocer el orden en el que se han ejecutado las unidades y las dependencias entre ellas (algunas tienes que esperar a que otras acaben):

```
>  systemd-analyze critical-chain
The time when unit became active or started is printed after the "@" character.
The time the unit took to start is printed after the "+" character.

graphical.target @16.345s
└─multi-user.target @16.344s
  └─docker.service @10.022s +6.320s
```

★ También podemos obtener esta información gráficamente ejecutando:

```
>  systemd-analyze plot > plot.svg
```