

Introducción a:
Redireccionamientos
Runlevels
Ubuntu update-rc.d
Gentoo rc-update

Copyright (c) **Jonathan Hernández Velasco** – <http://jhernandez.gpltarragona.org>

versión: 0.3 – 7-nov-2006

Esta obra está bajo una licencia Reconocimiento-CompartirIgual de Creative Commons. Para ver una copia de esta licencia, visite

<http://creativecommons.org/licenses/by-sa/2.0/es/> o envíe una carta a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Redireccionamientos

Una vez hemos aprendido a utilizar algunos de los comandos del sistema, es muy probable que en algunos casos nos interese utilizarlos de forma simultánea para agilizar las acciones que queremos realizar. Una operación muy interesante consiste en poder coger la salida de un comando para que sirva de entrada a otro y procesarla adecuadamente. El sistema operativo utiliza un mecanismo de pipes (tuberías), que nos permite redirigir las salidas de cualquier comando o programa hacia donde queramos. Su funcionamiento es muy simple: se trata de poner el carácter `|` entre los comandos, de manera que la salida del primero sirve como entrada para el segundo.

Vamos a verlo con un ejemplo: al escribir el comando

```
$ echo campo1:campo2:campo3:campo4
```

lo único que conseguiríamos sería que por pantalla nos apareciera

```
$ campo1:campo2:campo3:campo4
```

Si de esta salida sólo quisiéramos coger el `campo3`, podríamos redirigirla con un pipe hacia el comando `cut`, para que seleccione únicamente el campo que nos interesa de la siguiente manera:

```
$ echo campo1:campo2:campo3:campo4 | cut -d: -f 3
```

Naturalmente, podemos conectar tantos pipes como necesitemos para realizar acciones más prácticas que la que acabamos de ver. Otro tipo de redireccionamientos muy prácticos son aquellos que están relacionados con los ficheros. Este tipo de redireccionamiento nos permite coger toda la salida de un comando o programa y guardarla en un fichero utilizando el carácter `>`, igual que hacíamos con `|`. Por ejemplo, si queremos guardar en un nuevo fichero todo lo que vayamos escribiendo hasta apretar `CTRL+C`, podríamos utilizar lo siguiente:

```
$ cat > prueba.txt
```

Con `>>` podemos hacer exactamente lo mismo, pero en lugar de crear siempre el nuevo fichero, si éste ya existiera, se añadiría la información al final del mismo. Con `<` el redireccionamiento se realiza en sentido contrario, de modo que el contenido del fichero que le indicamos se dirigirá hacia el comando o programa señalado. Un aspecto muy interesante que debemos conocer es que en sistemas tipo UNIX se separa la salida normal de un programa con la de los errores. Aunque por defecto las dos salidas están dirigidas a la consola donde se ha ejecutado el programa, podemos manipularlas para que se dirijan hacia donde queramos.

Para ver esto de manera práctica, probamos de borrar un fichero que no existe con la siguiente instrucción:

```
$ rm fichero > resultados
```

Aunque estamos redireccionando la salida del comando hacia el fichero de resultados, por pantalla nos aparecerá un mensaje de error indicando que no se ha encontrado el fichero. Esto es debido a que por defecto los redireccionamientos sólo aceptan la salida estándar del programa y no la de error, que por defecto también se muestra por pantalla.

Para redirigir la salida de error, deberíamos indicar, antes del carácter `>` el número `2`, que es la salida de error (la `1` es la normal). De esta manera, ejecutando

```
$ rm fichero 2> resultados
```

sí que conseguiríamos que la salida se dirigiera al archivo de resultados.

También podemos guardar la salida normal y la de errores en dos ficheros diferentes:

```
$ rm fichero 1> resultados 2> errores
```

Si por el contrario quisiéramos que todas las salidas se dirigieran hacia un mismo archivo, podríamos utilizar `>&`.

```
$ rm fichero &> todo
```

Además, con el carácter `&` podemos encaminar salidas de un tipo hacia otras; por ejemplo, si quisiéramos encaminar la salida de errores hacia la normal, podríamos indicarlo del siguiente modo: `2>&1`

Es importante tener en cuenta que el orden de los redireccionamiento es significativo: siempre se ejecutan de izquierda a derecha.

Runlevels

Los daemons que tengamos ejecutándose en un determinado momento nos marcan los servicios que el sistema operativo está ofreciendo y/o recibiendo. El hecho de que podamos tener tantos daemons diferentes hace que tengamos que plantear su organización de forma adecuada.

Entenderemos un runlevel (o nivel de ejecución) como la ejecución de unos determinados daemons que a su vez proporcionan unos servicios concretos. En la instalación de un servidor es habitual diseñar una configuración para que en determinados momentos se puedan ofrecer determinados servicios y en otros no. Para permitir este tipo de funcionamiento, el sistema operativo nos proporciona diferentes niveles de ejecución que podremos adaptar a nuestras necesidades. Si bien podemos configurar el número de niveles de ejecución que queremos y la funcionalidad de cada uno de ellos, generalmente los sistemas tipo UNIX nos proporcionan 6 diferentes con las siguientes propiedades:

Nivel 0: El nivel de ejecución 0 está configurado para parar el sistema.

Nivel 1: Este nivel es denominado como single user, ya que sólo permite la entrada al sistema al root del mismo. Se arrancan los daemons mínimos y sirve para tareas de mantenimiento.

Nivel 2-5: Los niveles del 2 al 5 están destinados para ser configurados según las necesidades de cada instalación. Al instalar el sistema, por defecto todos son iguales. Estos niveles también se llaman multiusuario, ya que, por defecto, permiten que más de un usuario trabaje en el sistema.

Nivel 6: El último nivel está preparado para reiniciar el sistema. Es muy parecido al 0 pero se añade una función de reinicio.

El comando necesario para cambiar de nivel de ejecución es "init" (le pasamos como parámetro el nivel de ejecución que queramos) y para ver en cuál estamos, "runlevel".

Los comandos halt, reboot, shutdown o poweroff lo único que hacen es llamar al nivel de ejecución 0 o 6 realizando, antes, alguna operación concreta (ver su manual para más información).

En el fichero `/etc/inittab` tenemos definida toda la configuración de los runlevels: el nivel de ejecución por defecto, el número de consolas disponibles en cada uno de ellos, etc.

Cada línea del fichero es una directiva con la sintaxis: `<id> :<runlevels> : <action> : <process>` . El primer campo es el identificador de la directiva, seguidamente encontramos en qué niveles de ejecución es válida esta directiva, la acción a realizar y el proceso a lanzar. En el siguiente ejemplo explicamos cómo configurar algunas de estas directivas:

```
# El nivel de ejecución por defecto (en este caso, el 2)
id:2:initdefault:
# Scripts a ejecutar al arrancar el sistema (antes
# de entrar en el nivel de ejecución por defecto)
si::sysinit:/sbin/rc sysinit

# Inicialización del sistema, ejecuta todos los demonios
# que hay en el runlevel boot (/etc/runlevels/boot)
rc::bootwait:/sbin/rc boot

# Los runlevels disponibles.
l0:0:wait:/sbin/rc shutdown
l1:S1:wait:/sbin/rc single
l2:2:wait:/sbin/rc nonetwork
l3:3:wait:/sbin/rc default
l4:4:wait:/sbin/rc default
l5:5:wait:/sbin/rc default
l6:6:wait:/sbin/rc reboot

# Definición de las consolas abiertas en cada
# nivel de ejecución (la acción respawn indica
# que al terminar la ejecución del proceso
# getty se lance otra vez)
c1:12345:respawn:/sbin/agetty 38400 tty1 linux
c2:12345:respawn:/sbin/agetty 38400 tty2 linux
c3:12345:respawn:/sbin/agetty 38400 tty3 linux
c4:12345:respawn:/sbin/agetty 38400 tty4 linux
c5:12345:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux

# Comando a ejecutar al apretar CTRL+ALT+DEL
ca:12345:ctrlaltdel:/sbin/shutdown -r now
```

Como vemos, en este fichero se configura todo lo referente a los niveles de ejecución de forma muy flexible pudiendo cambiar lo que nos interese para adaptarlo mejor a nuestras necesidades. Fijémonos que, aunque aquí definamos el nivel de ejecución por defecto, también lo podríamos especificar al arrancar el sistema con el Lilo o Grub. Esto es muy útil, por ejemplo, cuando tenemos problemas graves en el sistema que no nos permiten arreglarlos adecuadamente; si arrancamos con el primer nivel (pasando `1` o `single` al Lilo o Grub), sólo se iniciarán las funciones más necesarias y podremos entrar para arreglar lo que haga falta.

Administración de los runlevels con update-rc.d

Sistemas que se inician a la System V como Debian tienen diferentes *runlevels* que permiten poner el sistema en un estado definido. Por ejemplo "2" es el runlevel por omisión de Debian, "0" apaga el sistema y "1" es el modo monousuario que permite ejecutar trabajos de mantenimiento.

Al entrar a un runlevel se ejecutan diferentes guiones de inicio en el directorio `/etc/init.d`. Estos guiones disponen de las funciones "start", "stop", "reload" y "restart" para iniciar, terminar, recargar una nueva configuración o reiniciar el programa correspondiente.

Por ejemplo con

```
/etc/init.d/inetd stop
```

se puede parar el **inetd**.

A cada runlevel pertenece una colección de enlaces a los guiones de inicio. Estos enlaces se encuentran en los directorios `/etc/rc?.d` donde "?" es el correspondiente runlevel. Los enlaces tienen nombres especiales. Si empiezan con "S" inician, si empiezan con "K" terminan el correspondiente programa. Después de la letra sigue un número de dos dígitos que determina el orden de ejecución y terminan en el nombre del guión. Un ejemplo: S20samba es un enlace al guión de inicio de **samba**. Inicia samba después de la ejecución de los guiones S19... (o menor) y antes de los guiones con números mayores.

Para deshabilitar el inicio automático de un programa por ejemplo para realizar pruebas hay que borrar todos los enlaces al guión `/etc/init.d/samba` en los directorios `/etc/rc?.d` y más tarde - para restaurar el estado anterior - reponerlos.

Este trabajo se simplifica mucho con el programa **update-rc.d** de Debian. Un ejemplo simple es:

```
update-rc.d -f samba remove
```

Con este comando se borran todos los enlaces en los directorios `rc?.d` al guión `/etc/init.d/samba`. La opción "-f" le dice hacerlo a pesar de que existe todavía el guión samba. Todavía es posible arrancar Samba manualmente con `/etc/init.d/samba start`.

Otro ejemplo:

```
update-rc.d samba defaults
```

Este comando crea enlaces que inician samba en los runlevels 2345 y enlaces que lo terminan en los runlevels 016 con la prioridad 20. Equivale a

```
update-rc.d samba start 20 2 3 4 5 . stop 20 0 1 6 .
```

que pone los argumentos explícitamente.

Información más detallada se encuentra en las páginas del manual `init(8)`, `inittab(5)` y `update-rc.d(8)`

Thomas Bliesener <bli@melix.com.mx>

Página creada en: 2004-11-04 11:40:58 +0000

© Copyright 2001, 2002, 2003, 2004, La Espiral, debian-laespiral@lists.debian.org

Permitida la copia y distribución textual, integral, siempre y cuando se mantenga este aviso.

Gentoo rc-update

En Gentoo existen por defecto los runlevels “boot”, “default” y “nonetwork”.

“boot”: Contiene demonios de inicialización del sistema no básicos, como carga de módulos del kernel, configuración de los caracteres, de la red... etc;

“default”: Contiene todos los demonios restantes.

“nonetwork”: Por defecto, sólo contiene el script “local”, que por defecto esta vacío y sirve para añadir nuestros propios scripts.

Con la herramienta “rc-update” podemos controlar todo lo relacionado con los runlevels. Así:

```
# rc-update add demonio nivel
```

añadiría el demonio “demonio” al runlevel “nivel”. Y:

```
# rc-update del demonio nivel
```

lo quitaría de ese nivel.

Además también tenemos:

```
# rc-update show
```

Que lo que hace es mostrar todos nuestros demonios y los runlevels a los que pertenece. Éste es un ejemplo:

```
valinor ~ # rc-update show
  alsasound | boot
  apache2   |
  bootmisc  | boot
  checkfs   | boot
  checkroot | boot
  clock     | boot
  consolefont | boot
  crypto-loop |
  cupsd     | default
  dcron     | default
  distccd   | default
  domainname |
  dotnet    | default
  esound    | default
  exim      |
  famd      | default
  gpm       | default
  hdparm    | boot
  hostname  | boot
  hotplug   |
  iptables  |
  keymaps   | boot
  local     | default nonetwork
  localmount | boot
  modules   | boot
  mysql     |
```

```

net.eth0 | boot
net.lo   | boot
net.ppp0 |
netmount |         default
nfs      |
nfsmount |
nscd     |
numlock  |
portmap  |
rmnologin | boot
rsyncd   |
samba    |
serial   | boot
sshd     |         default
syslog-ng |         default
urandom  | boot
xdm      |
xfs      |         default
xinetd   |         default

```

Otra forma de mirarlo es ir al directorio de runlevels (/etc/runlevels) y listar dentro de cada directorio los archivos que tiene. Cada archivo dentro del directorio no es un demonio en sí, sino un link simbólico al demonio. Recordad que todos los demonios se guardan en el directorio "/etc/init.d".

Después con el comando:

```
$ rc-status
```

podemos ver qué demonios de nuestro runlevel tenemos arrancados. Ejemplo:

```

valinor ~ # rc-status
Runlevel: default
  cupsd           [ started ]
  dotnet          [ started ]
  dcron           [ started ]
  famd            [ started ]
  esound          [ started ]
  gpm             [ started ]
  xfs             [ started ]
  netmount        [ started ]
  sshd            [ started ]
  xinetd          [ started ]
  syslog-ng       [ started ]
  local           [ started ]
  distccd         [ started ]
valinor ~ #

```

Finalmente, si queremos retornar al estado inicial del runlevel, es decir, si queremos tener únicamente en funcionamiento los demonios que tenemos predefinidos en el runlevel que estamos ejecutando en ese momento, podemos hacerlo ejecutando el comando:

```
# rc
```

Si tenemos demonios de más ejecutándose, los parará. Y si tenemos de menos, pondrá en funcionamiento los que faltan. Por ejemplo:

```
valinor ~ # rc
* Stopping apache2... [ ok ]
* Stopping DDClient... [ ok ]
* Stopping USB and PCI hotplugging... [ ok ]
* Stopping mysqld... [ ok ]
* Stopping portmap... [ ok ]
* Starting distccd... [ ok ]
* Starting portmap... [ ok ]
* Starting famd... [ ok ]
valinor ~ #
```

En el ejemplo tenía ejecutando los demonios “apache2”, “ddclient”, “hotplug”, “mysqld”, “portmap”, que no estaban en el runlevel default. Al ejecutar “rc” me lo para. También me añade lo del runlevel “default” y que no estaba ejecutando. Es el caso de “distccd” y “famd”. También me ejecuta “portmap” porque detecta que es una dependencia de “famd”.

Bibliografía.

Este documento ha sido hecho por **Jonathan Hernández Velasco** y alguna información ha sido recopilada de:

“Apuntes Master en Software Libre” de la UOC.