Administración de Servicios y Aplicaciones

Miguel Ortuño
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

Septiembre de 2022

Empaquetado de ficheros

Almacenar varios ficheros en uno solo, no necesariamente con compresión

Utilidad:

- Más cómodo de manejar (copiar, enviar por correo, etc)
- Conservar metainformación (permisos) o incluso mayúsculas/minúsculas, tildes, etc si los ficheros van a pasar por un sistema de ficheros diferente
  - ISO9660 (cdrom)
  - vfat (Windows, discos externos, pendrives)
  - ntfs (Windows)

gzip

Comprime o descomprime 1 fichero
Extensión: `fichero.z` `fichero.gz`

- Comprimir y descomprimir (borrando el original):
  `gzip fichero`
  `gunzip fichero.gz`

- Comprimir y descomprimir (manteniendo el original):

```
gzip -c fichero > fichero.gz
zcat fichero.gz > fichero
zcat fichero.gz  | less
```

tar + gzip

Comprime o descomprime varios ficheros, directorios
Extensión: `fichero.tar.gz` `fichero.tgz`

- Comprimir:
  `tar -cvzf fichero.tgz fichero1 fichero2`
- Descomprimir:
  `tar -xvzf fichero.tgz`
- Mostrar contenido:
  `tar -tzf fichero.tgz`

WinZip

- Por motivos de licencias, originalmente no había compresores para Linux. (Pero las aplicaciones Windows saben descomprimir descomprimir `.tgz`)
- Descomprimir: `unzip fichero.zip`

bz2

Formato que ofrece compresión más alta que .gz, (empleando más CPU y memoria)

- Comprimir y descomprimir 1 fichero, borrando el original

```
bzip2 fichero
bunzip2 fichero.bz2
```

- Comprimir y descomprimir 1 fichero, manteniendo el original

```
bzip2 -c fichero > fichero.bz2
bunzip2 -c fichero.bz2 >  fichero
```

- Comprimir y descomprimir varios ficheros, manteniendo el original

```
tar -c fichero1 fichero2  |  bzip2 > fichero.bz2
tar -xjf fichero.bz2
```

Fragmentación de ficheros

Si necesitas trocear una imagen de gran tamaño en ficheros que quepan en un *pendrive* o cdrom

- Empaquetar y comprimir un directorio:
  `tar -cvzf mi_imagen.tgz mi_directorio`
- Mostrar contenido:
  `tar -tzf mi_imagen.tgz`
- Trocear:
  `#  tamaño  fichero  prefijo`
  `split -b 500MB  mi_imagen.tgz  mi_imagen.tgz.` (Observa que el segundo parámetro es igual al primero, pero añadiendo un punto)
- Habremos generado
  `mi_imagen.tgz.aa  mi_imagen.tgz.ab  mi_imagen.tgz.ac`

En la máquina destino (no importa si en el *host* el S.O. es distinto)

- Unir los fragmentos
  `cat mi_imagen.tgz.* > mi_imagen.tgz`
  (En MS Windows para este paso podemos emplear HjSplit, Free File Splitter o cualquier otro programa similar)
- Descomprimir y desempaquetar:
  `tar -xvzf mi_imagen.tgz`
  (En MS Windows podemos usar 7-Zip o similares)

Instalación de paquetes

- Método clásico para instalar programas:
  Formato .tgz
  Descomprimir y seguir las instrucciones del fichero README
  Suele ser del estilo de

```
./configure
make compile
make install
```

- Sistema de gestión de paquetes
  Colección de herramientas que automatizan la instalación, actualización y eliminación de programas.

- Gestión de paquetes, Debian y derivados
  Paquetes en formato `.deb`
  Se pueden manejar directamente con `dpkg`, o con `apt-get`, `apt`, `aptitude`, `dselect`, o `synaptic`
- Gestión de paquetes, RedHat y derivados
  Paquetes en formato `.rpm`
  Se pueden manejar directamente con `rpm`, o con `up2date` o `yum`

El sistema de paquetes de Debian

Los paquetes mantienen *dependencias* entre sí, de forma que la instalación de un paquete puede:

- *depender* de que se instale también otro
- *recomendar* que se instale también otro
- *sugerir* que se instale también otro
- *entrar en conflicto* con otro actualmente instalado

dpkg

- Es la herramienta básica de gestión de paquetes, que es usada por las otras (dselect, apt-get, apt, aptitude, synaptic).
- Usos principales:
  - `dpkg -i paquete_VVV-RRR.deb`
    Instala un paquete
  - `dpkg -r paquete`
    Desinstala (*remove*) un paquete, elimina todo excepto los ficheros de configuración
  - `dpkg -P paquete`
    Purga un paquete, eliminando incluso los ficheros de configuración
- Tiene muchas opciones. Puede esquivarse el esquema de dependencias (peligroso) con las opciones que empiezan por `--force-...`

Versiones de Ubuntu:

```
nombre año.mes

Warty Warthog 4.10       Hoary Hedgehog 5.04
...                      ...
Saucy Salamander 13.10   Trusty Tahr 14.04 LTS
Utopic Unicorn 14.10     Vivid Vervet 15.04
Wily Werewolf 15.10      Xenial Xerus 16.04 LTS
Yakketiy Yak 16.10       Zesty Zapus 17.04
Artful Aardvark 17.10    Bionic Beaver 18.04 LTS
Cosmic Cuttlefish 18.10  Disco Dingo 19.04
Eoan Ermine 19.10        Focal Fossa 20.04 LTS
Groovy Gorilla 20.10     Hirsute Hippo 21.04
Impish Indri 21.10       Jammy Jellyfish 22.04 LTS
Kinetic Kudu 22.20
```

Versión estándar: Desde 13.04, soportada durante 9 meses (18 meses en las versiones anteriores)
LTS: Long Term Support: soportada durante 3 años en escritorio y 5 en servidor

Ubuntu Desktop / Ubuntu Server Edition / Ubuntu Server Edition JeOS
Variantes de Ubuntu: Kubuntu, Xubuntu, Gobuntu, Ubuntu Studio

apt

- La herramienta más sencilla de usar y más potente.
- Usa *repositorios*: sitios centralizados donde se almacenan paquetes
- Las direcciones de los repositorios se indican en el fichero `/etc/apt/sources.list`
- Los repositorios de ubuntu se dividen en 4 componentes
  1. *Main*. Soportado oficialmente por ubuntu. Libre
  2. *Restricted*. Soportado oficialmente. No libre
  3. *Universe*. No soportado oficialmente. Libre
  4. *Multiverse*. No soportado oficialmente. No libre

  Además, se pueden añadir componentes de terceros

```
# deb cdrom:[Ubuntu 6.06 _Dapper Drake_ - Release i386 (20060531)]/ dapper main
deb http://archive.ubuntu.com/ubuntu edgy main restricted
deb http://security.ubuntu.com/ubuntu edgy-security main restricted
deb http://archive.ubuntu.com/ubuntu edgy-updates main restricted

## All community supported packages, including security- and other updates
deb http://archive.ubuntu.com/ubuntu edgy universe multiverse
deb http://security.ubuntu.com/ubuntu edgy-security universe multiverse
deb http://archive.ubuntu.com/ubuntu edgy-updates universe multiverse

# Google Picasa for Linux repository
deb http://dl.google.com/linux/deb/ stable non-free
```

Uso básico de apt

Desde línea de comandos se puede usar `apt-get`

- `apt-get update`
  Actualizar lista de paquetes:
- `apt-get upgrade`
  Actualizar todos los paquetes instalados a la última versión disponible (sin cambiar de distribución)
- `apt-get install paquete`
  Instalar un paquete (resolviendo conflictos)

En 2014 aparece la herramienta `apt`, con la misma finalidad e interfaz de usuario, pero que resulta un poco más fácil de manejar porque unifica `apt-get` y `apt-cache`

```
apt update
apt upgrade
apt install paquete
```

Aunque indiquemos a nuestro sistema de paquetería que instale la última versión de un paquete, tal vez no sea posible. Se dice que el paquete está *retenido* (*hold*)

- El paquete depende de otro no incluido en la distribución actual
- El administrador lo ha retenido *a mano* (no le gusta, da problemas...)

```
sudo install feta
sudo feta hold nombre_del paquete
sudo feta unhold nombre_del paquete
```

- `apt remove paquete`
  Desinstala un paquete
- `apt purge paquete`
  Desinstala un paquete y borra su configuración
- `apt full-upgrade`
  Actualiza *agresivamente* todos los paquetes instalados, lo que puede incluir el paso a la versión más reciente de la distribución

Otros mandatos interesantes

En los repositorios hay muchos paquetes ¿Cómo saber cuál necesito?

- `apt search cadena`
  Buscar una cadena en el nombre o descripción de un paquete. Indica el estado del paquete (instalado, no instalado, borrado...)
- `apt show paquete`
  Muestra descripción del paquete
- `dpkg-reconfigure paquete`
  Reconfigurar un paquete

Sistemas de paquetes en macOS

Apple no tiene previsto el uso de estos sistemas para usuarios *normales*. Pero sí son muy útiles para usuarios con perfil de desarrollador o administrador. Se usan prácticamente igual que apt-get

Actualmente podemos optar por tres sistemas

1. fink
   El más antiguo. Basado en apt-get. Poco usado hoy
2. macports
   Basado en los ports de FreeBSD. Muy completo. Muy independiente de Apple
3. homebrew
   El más moderno y mejor integrado con Apple. Tal vez el más popular hoy

Recapitulación: ficheros tar

Como hemos visto, la forma clásica para distribuir aplicaciones, desde finales de los años 70 es el uso de ficheros .tar o tar.gz con el código fuente y tal vez un *Makefile*

- Exigen mucha intervención manual por parte del administrador de la máquina
- El número de dependencias es muy alto, incluyendo ¡todas las herramientas de compilación!

Recapitulación: paquetes tradicionales

Desde finales de los años 90, sistemas de gestión de paquetes (deb, rpm, brew, etc)

- Los responsables de cada distribución (Debian, Ubuntu, Fedora, Suse, etc) toman la versión original de cada aplicación (denominada *upstream*) y la preparan para que funcione en una versión concreta de una distribución concreta, con todas las dependencias necesarios.
- Ya no requiere trabajo por pate del administrador local, pero sí de los responsables de la distribución
- Para instalar un paquete, es necesario
  - Disponer de la versión exacta para la distribución exacta
  - Que ese paquete esté correctamente preparado
  - Que el estado de mis paquetes sea consistente (a veces hay conflictos, paquetes mal instalados, etc)

Paquetes Snap

En 2014 Canonical desarrolla los paquetes Snap, que están ganando popularidad recientemente

- Un único paquete Snap incluye todo lo necesario para que funcione la aplicación, con todas sus dependencias en cualquier distribución Linux, sin necesidad de adaptar nada
- Un paquete Snap incluye un fichero que se monta en el sistema de ficheros, que se descomprime sobre la marcha
- Cada aplicación se actualiza automáticamente
- La aplicación se ejecuta de forma aislada dentro de un *sandbox*, con acceso limitado al *host*

Inconvenientes:

- La integración con el resto del sistema puede ser peor
- El tiempo necesario para comenzar a ejecutar la aplicación puede ser peor

Uso de Snap

Podemos consultar todos los *snaps* disponibles en la *Snap Store*
https://snapcraft.io
El manejo es muy sencillo

```
snap find <palabras clave>
sudo snap install <paquete>
snap list
sudo snap remove <paquete>
```

Más información:
https://snapcraft.io/docs/getting-started

Localizar ficheros

- `find` Busca un fichero
  `find . | grep fichero` Filtra la búsqueda
- `locate` Busca un fichero (en una base de datos)
- `updatedb` Actualiza la base de datos

Hora. Parada del sistema

- `shutdown -P now` ≡ `poweroff`
  Apaga el sistema
- `shutdown -r now` ≡ `reboot`
  Reinicia el sistema
- `sleep n`
  Duerme la shell segundos
- `sleep 28800 ; halt`
  Detiene la máquina al cabo de 8 horas

- Poner fecha y hora:
  - Automáticamente: Demonio *ntpd*, cliente de *Network Time Protocol*
  - Manualmente

    `date -s AAAA-MM-DD`

    `date -s HH:MM`

Casi siempre hay varias soluciones para una tarea. Generales o particulares

- `find . |grep cadena`
  `find . -name cadena\*`
- `sleep 60 | shutdown -h now`
  `shutdown -h 1`
- etc, etc

Todas sirven. ¿No? ¿Cuál es *mejor*?

- Cuando somos novatos en un sistema, con una solución general sabremos resolver ese problema y otros parecidos
- Cuando conocemos mejor un sistema y dominamos las soluciones generales, las soluciones particulares suelen ser más eficientes

Copias de seguridad

- `tar` o similares
  Problema: Siempre se duplican los directorios enteros
- `rsync`
  Mirror unidireccional. Permite mantener una réplica de un directorio. Solo se actualizan las novedades. No permite modificar la réplica
- `FreeFileSync, Synkron`
  Herramientas libres para sincronización bidireccional (Windows, Linux, OS X).
  Sincronizan dos (o más) directorios: Cualquiera de los dos directorios puede modificarse

- Sistemas de almacenamiento permanente
  `Time Machine` (OS X)
  `dumpfs` (bsd)
  `pdumpfs` (Linux, Windows)
  `TimeVault, FlyBack` (Linux)
  `venti` (Plan 9)
  - Se registran los cambios en los ficheros, sin borrar nunca nada
  - Mantienen una *foto* del estado diario del sistema de ficheros, en un directorio con formato yyyy/mm/dd
  - Parece mucho, pero hoy el almacenamiento es muy barato. P.e. si generamos 10 Mb diarios, necesitamos unos 4Gb anuales

Administración de los demonios

Los demonios son programas relativamente *normales*, con algunas particularidades

- Ofrecen servicios (impresión, red, ejecución periódica de tareas, logs, etc)
- Suelen estar creados por el proceso de arranque *init* (ppid=1)
- Sus nombres suelen acaban en *d*
- Se ejecutan en *background*
- No están asociados a un usuario en una terminal
- El grueso de su configuración suele hacerse desde un único fichero
  En el caso de debian, `/etc/midemonio.conf`
- Se inician y se detienen de manera uniforme

Unix System V

Versión de Unix comercializada en 1983 por AT&T

- La mayoría de los Unix, incluyendo Linux, son derivados de System V
- Otros Unix son derivados del Unix BSD de aquella época: esto incluye los BSD actuales y OS X (Apple)

System V introduce una forma de organizar los demonios basada en

- *Niveles de ejecución*
- Scripts en `/etc/init.d`
- Ordenación lineal de sucesos:
  Las tareas se ordenan secuencialmente en orden preestablecido, solo cuando una está completamente acabada empieza la siguiente

Systemd

- El sistema de arranque tradicional de Linux (System V) no es adecuado para las máquinas actuales
  - Son externos: aparecen y desaparecen
  - Están en red
  - Ahorran energía
  - ...
- *Systemd* es un sistema de arranque basado en eventos (pueden suceder en cualquier orden, puede haber tareas en paralelo)
- Mantiene una capa adicional de software para que las órdenes al estilo System V sigan funcionando

- El desarrollo de Systemd lo compienza Red Hat en el año 2010
- En Ubuntu solamente se utiliza desde la versión 15.04 (año 2015). Anteriormente empleaba *upstart*, un sistema similar, también compatible con System V

Ficheros de configuración

Los ficheros donde se configura un demonio son:

- Fichero principal de configuración
  `/etc/midemonio.conf`
- Configuración de puesta en marcha y parada
  - System V
    `/etc/init.d/midemonio`
  - Systemd
    `/etc/init/midemonio.conf`
- Configuración del administrador local
  `/etc/default/midemonio`
  Solo existe en Debian y derivados (también en Ubuntu con Upstart). No lo usan todos los paquetes

Directorio /etc/default

- La inmensa mayoría de los parámetros de `/etc/midemonio.conf` y de `/etc/init.d/midemonio` o de `/etc/init/midemonio.conf` los ha escrito el desarrollador del demonio o el empaquetador de la distribución, es normal que el administrador local de cada máquina concreta solo modifique unos pocos
- En algún momento habrá que actualizar el demonio a una versión nueva, que frecuentemente incluirá cambios en sus ficheros de configuración, escritos por el desarrollador o el empaquetador
  - ¿Instalamos los ficheros nuevos y *machacamos* los viejos? Problema: se pierde la configuración que ha personalizado el administrador local
  - ¿Mantenemos los viejos y descartamos los nuevos? Problema: se pierden los cambios de la versión actual, el fichero de configuración (antiguo) podría incluso se incompatible con el demonio (actual)

Solución: fichero `/etc/default/midemonio`

- Es un fichero muy corto, con muy pocos parámetros, muy importantes, que se sabe que serán modificados por el administrador local
- Cuando se instalan versiones nuevas del demonio, este fichero se mantiene
- Los cambios introducidos por las nuevas versiones de los demonios estarán en `/etc/midemonio.conf` o en `/etc/init.d/midemonio` o `/etc/init/midemonio.conf`

Administración estilo System V

El código de un demonio puede estar en cualquier lugar del sistema de ficheros. Pero siempre se coloca en `/etc/init.d/midemonio` un script para manejarlo

- `/etc/init.d/midemonio start`
  Inicia el servicio
- `/etc/init.d/midemonio stop`
  Detiene el servicio
- `/etc/init.d/midemonio restart`
  Detiene e inicia el servicio. Suele ser **necesario para releer los ficheros de configuración** si se han modificado (`/etc/midemonio.conf`)

Con frecuencia también está disponible

- `/etc/init.d/midemonio reload`
  Lee el fichero de configuración sin detener el servicio

Niveles de ejecución

¿Qué demonios se ponen en marcha cuando se inicia el sistema?
Un Nivel de ejecución (*runlevel*) es una configuración de arranque.
Para cada nivel, se define un conjunto de demonios que deben ejecutarse

- Este es un concepto de System V, en *Systemd* se usan *targets*, que son equivalentes

Supongamos una fábrica. Diferentes niveles (estados), no secuenciales. Al entrar en un nivel se apagan ciertos sistemas y se encienden otros

```
Nivel 1 - Noche
    Al entrar en este nivel apagar
        01 motores
        02 luces principales
    Al entrar en este nivel encender
        01 alarma
        02 luces_auxiliares
Nivel 2 - Producción normal
    Al entrar en este nivel apagar
        01 alarma
        02 luces auxiliares
    Al entrar en este nivel encender
        ....
Nivel 3-  Mantenimiento
    Al entrar en este nivel apagar
        01 motores
        ....
```

El responsable de conectar y desconectarlo todo será el vigilante de seguridad, así que hay que dejarle unas instrucciones muy claras:

`ordinal` `[encender|apagar]` `nombre_del_sistema`

Código	**Significado**	**Mandato**
S10motor-ppal	1º encender motor principal	`/etc/init.d/motor-ppal start`
S20motor-aux	2º encender motor auxiliar	`/etc/init.d/motor-aux start`
K10alarma	1º apagar alarma	`/etc/init.d/alarma stop`

Dentro de cada nivel, las tareas se ordenan desde 00 hasta 99 (con un cero a la izquierda para los valores del 0 al 9)

El *vigilante de seguridad* es el proceso `init`.

Hay un directorio por nivel:
Debian y derivados

```
/etc/rc0.d
/etc/rc1.d
...
```

Red Hat y derivados

```
/etc/rc.d/rc0.d
/etc/rc.d/rc1.d
...
```

Hay otro directorio cuyos servicios se activan siempre, en cualquier nivel

```
/etc/rcS.d
```

Dentro de los directorios hay enlaces simbólicos

- Apuntan al script en `/etc/init.d` que controla el demonio
- Cada nombre del enlace indica conexión/desconexión, ordinal y script a manejar

Cuando entra en el nivel N, el proceso init se encarga de

- Ejecutar por orden todos los scripts en `/etc/rcN.d` que empiezen por K (de Kill). Les pasa el parámetro *stop*
- A continuación, ejecuta por orden todos los scripts en `/etc/rcN.d` que empiezen por S (de Start). Les pasa el parámetro *start*

```
who -r
```

Indica el nivel de ejecución actual

```
0   Halt (Parada del sistema)
1   Modo monousuario, usuario root, sin red
2-4 Diversos modos multiusuario, sin gráficos
5   Modo multiusuario completo, con X Window
6   Reboot (Reiniciar el sistema)
```

Ejemplo del contenido de /etc/rc2.d/

```
S10acpid                S18hplip          S20postfix          S89atd
S10powernowd.early      S19cupsys         S20powernowd        S89cron
S10sysklogd             S20apmd           S20rsync            S90binfmt-support
S10wacom-tools          S20festival       S20ssh              S98usplash
```

Ejemplo del contenido de /etc/rc6.d/

```
K19cupsys          K25mdadm              S15wpa-ifupdown     S50lvm
K19setserial       K25nfs-user-server    S20sendsigs         S50mdadm-raid
K20dbus            K30etc-setserial      S30urandom          S60umountroot
K20laptop-mode     K50alsa-utils         S31umountnfs.sh     S90halt
```

Resumiendo, para ejecutar automáticamente un demonio manejamos 3 ficheros

- El fichero con el ejecutable del demonio

  p.e.

  `/usr/sbin/sshd`

  - Si se trata de un servicio que no es estándar en la distribución, su sitio es el directorio `/usr/local`
- El script que maneja el demonio
  - Acepta los parámetros `start`, `stop`, `reload`, `...` y llama el demonio en consecuencia

  p.e.

  `/etc/init.d/ssh`
- El enlace, dentro del directorio correspondiente al nivel, que apunta al script

  p.e.

  `/etc/rc5.d/S02ssh`

  apuntando a

  `/etc/init.d/ssh`

Los demonios no muestran información ni en la consola ni en ninguna aplicación gráfica

- Los demonios usan el demonio *syslogd* o *sysklogd* para notificar y almacenar información relevante: inicio, parada, estado, peticiones, respuestas, errores, etc
  A partir del año 2009 es más habitual emplear *rsyslogd*, muy similar a *syslogd*
- Todo esto se escribe en diversos ficheros de texto, siendo los más interesantes
  - `/var/log/syslog`
    Información general del sistema
  - `/var/log/auth.log`
    Información sobre autenticación de usuarios

- Podemos ver un fichero cualquiera, p.e. un log, con *cat*

  `cat /var/log/syslog`
  (Muestra un fichero entero)
- Es más práctico usar *tail*

  `tail -20 /var/log/syslog`
  (Muestra las últimas 20 línea)

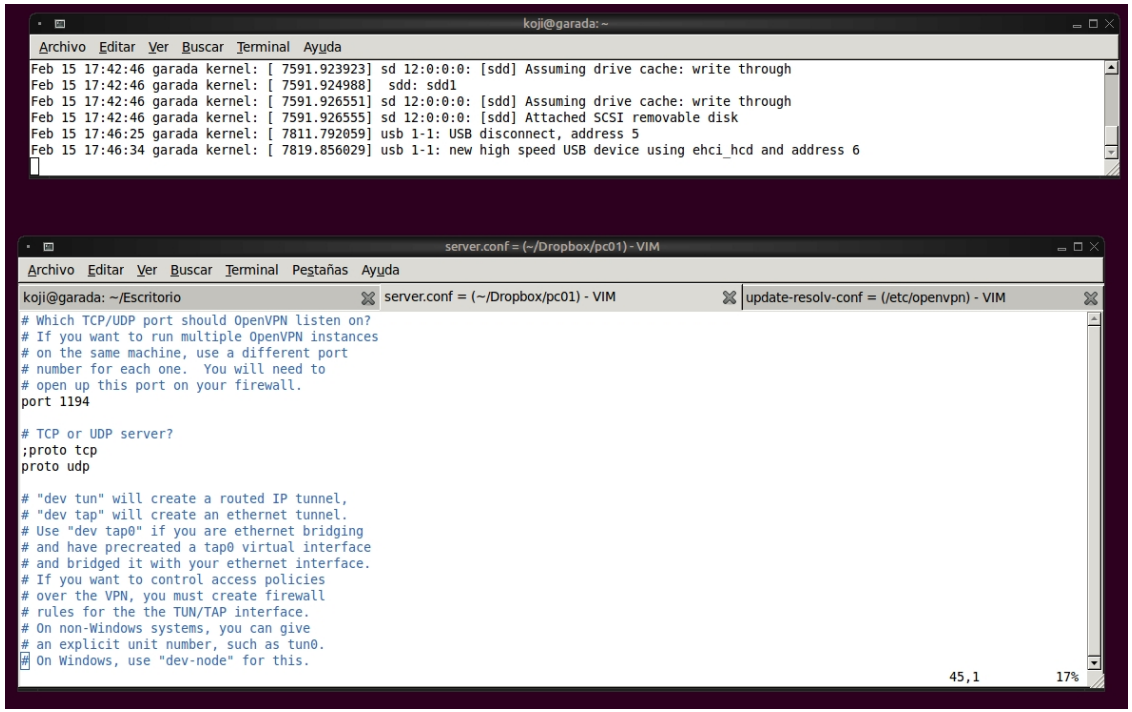
  `tail /var/log/syslog`
  (Muestra las últimas 10 línea)

O mejor aún, si queremos monitorizar continuamente un log, abrimos un terminal exclusivamente para esto y ejecutamos

- `tail -f /var/log/syslog`

(Muestra las últimas líneas, se queda esperando a que haya novedades en el fichero, y cuando las hay, las muestra también)

Si estamos depurando un servicio y tenemos una sesión gráfica, puede ser útil esta disposición del escritorio: dejar un terminal siempre visible, ejecutando `tail -f`, y trabajar en otro terminal con varias pestañas



Si no tenemos gráficos, podemos pulsar Alt F2, Alt F3, etc, abrir una sesión y dedicarla a los logs (también dentro de VirtualBox)

Tareas periódicas

- Automatizan la gestión del sistema
- Fiabilidad. Protegen frente a olvidos
- Se ejecutan en el momento preciso (día y hora)
- Ayudan o detectan situaciones de error
- Facilitan el control del sistema
- Programas:
  - `cron`
  - `anacron`. Permite ejecutar algo programado para un momento en que el sistema estaba apagado
  - `at`. Ejecuta una tarea a la hora indicada por *stdin*

Usos de las tareas periódicas

- Generación de informes periódicos (fin de mes, etc.)
- Estado de las comunicaciones
- Borrado de ficheros temporales (`/tmp`, `/var/tmp`)
- Tareas de respaldo de información
- Control de los procesos presentes en el sistema
- Parada del sistema según horarios de trabajo
- Recordatorios
- Descarga de *software* en horarios de poco tráfico

cron

- Es uno de los demonios esenciales de un sistema, siempre está arrancado (`/usr/sbin/cron`)
- Se encarga de ejecutar tareas programadas para un determinado momento, bajo la identidad del usuario que lo programó y con precisión de 1 minuto
- Se controla a través del uso de determinados ficheros de configuración (solo para el superusuario) y mediante el uso de la orden "`crontab`" (para todos los usuarios).

```
SHELL=/bin/bash
MAILTO=koji
PATH=/usr/local/bin:/usr/bin:/bin
#   m     h    dayofmonth  month  dow    command
   16     *        *         *     *     ping 193.147.71.119  -c 1
   0      9        4         8     *     echo "regar plantas"
   0    15,18      *         *    1-5    echo "hora de salir" | wall
```

m: Minuto. De 0 a 59
h: Hora. De 0 a 23
dayofmonth: de 0 a 31
month: de 1 a 12
dayofweek: de 0 a 7. 0=7=domingo, 1=lunes, 2=martes...
Cada línea es una tarea

- Se pueden poner comentarios con # pero no en cualquier posición, solo siguiendo el patrón *principio de línea, 0 o más espacios, almohadilla*
- En las asignaciones `variable=valor`, el valor no se expande. Todo lo que hay a la derecha del igual se toma literalmente. Por tanto, no pueden hacerse cosas como p.e. `PATH=$HOME/bin:$PATH`
- Se puede usar la variable de entorno `$PATH` en un comando. Ejemplo:

```
0   18        *        *     *   $HOME/blabla/bla.py
```

- Es necesario dejar una linea en blanco al final de la tabla

```
*          -> todos
1-4        -> 1,2,3 y 4
1,4        -> 1 y 4
*/3        -> cada 3
1-15/3     -> los primeros 15, cada 3
```

Ejemplos y contraejemplos:

```
#   m     h    dayofmonth  month  dow    command
    *   14-15      *         *      *     echo "OJO: de 14 a 15:59"
    *   23-7       *         *      *     echo "RANGO ILEGAL, 23>7"
```

- `crontab -e`
  Edita la tabla de cron del usuario. Usa el editor por omisión (normalmente vi). Podemos usar otro cambiando la variable de entorno EDITOR
- `crontab -l`
  Muestra tabla de cron
- `crontab mi_tabla`
  El fichero *mi_tabla* pasa a ser nueva tabla de cron

Ambigüedades en la especificación del momento de ejecución

- El día en el que se ejecuta cada orden se puede indicar de 2 maneras:
  - día del mes (3^{er} campo)
  - día de la semana (5^{o} campo)

  En caso de aparecer los dos campos (esto es, que ninguno es "*"), la interpretación que hace `cron` es que la orden debe ejecutarse cuando se cumpla *cualquiera* de ellos
  Ejemplo:
  `0,30 * 13 * 5 echo 'Viernes 13!' | wall`
  (ejecuta la orden cada media hora, todos los viernes y además todos los días 13 de cada mes)

Momentos "especiales" (solo Linux)

En lugar de especificar los 5 primeros campos, se puede usar una cadena de las siguientes:

- `@reboot`: Se ejecuta al iniciarse la máquina.
- `@yearly`: Se ejecuta una vez al año.
- `@monthly`: Se ejecuta una vez al mes.
- `@weekly`: Se ejecuta una vez por semana.
- `@daily`: Se ejecuta una vez al día.
- `@hourly`: Se ejecuta una vez por hora.

Entorno de ejecución de las tareas

- Cada tarea de cron se ejecuta por una *shell* `/bin/sh`. (a menos que definamos otra cosa en SHELL)
- Causa de **errores frecuentes:** El PATH con el que cron busca el mandato no es el del usuario, sino `/usr/bin:/bin`.
  Soluciones:
  - Indicar PATH en la tabla
  - Especificar el path abosoluto del mandato (p.e. `/usr/local/bin/mimandato`)
- Quien ejecuta las tareas no es el dueño de la tabla, sino cron. Aunque emplea algunas variables de entorno del dueño de la tabla, como `LOGNAME` y `HOME`.
- La entrada estándar de cada tarea se redirige de `/dev/null`, la salida estándar y la de error se envían por correo electrónico al propietario de la tarea (si hay servidor de correo)