Tema 2: Introducción a los sistemas Linux/Unix

Administración de Sistemas e Redes

Tomás Fernández Pena
tf.pena@usc.es

Índice

1. Introducción a Unix y Linux

Características de UNIX:

- Sistema operativo potente, flexible y versátil.
- Características: portabilidad, adaptabilidad y simplicidad, naturaleza **multiusuario** y **multitarea**, adecuación a redes.
- Disponibilidad de código fuente (algunas versiones)
- Implementado casi íntegramente en C (lenguaje de alto nivel).

GNU/Linux:

- Sistema operativo libre, de código abierto, similar a Unix
- Código fuente con licencia GPL
- Disponible para un gran número y variedad de sistemas: supercomputadores, servidores, sobremesas, portátiles, PDAs, móviles, sistemas empotrados,...

1.1. Historia de Unix

- Multics: proyecto de Bell Labs (AT&T), General Electrics y el MIT (1969) para el sistema GE 645
  - demasiado ambicioso para la época (pobre rendimiento)
- Thompson y Ritchie (Bell) migran un juego (*Space Travel*) en Multics de GE 645 a PDP-7.
- Empiezan del desarrollo de un SO para el PDP-7 $\longrightarrow$ Surge UNIX
- En 1970, UNIX se instala en una PDP-11
- En 1971 se edita el primer *UNIX Programmer's Manual.*
- En 1973 UNIX se reprograma en C (Ritchie)
- En 1974/75 UNIX v6 se difunde fuera de los laboratorios Bell y llega a las universidades
  - Los investigadores tienen acceso al código fuente del UNIX de AT&T
- En 1977 la Universidad de Berkeley licencia UNIX BSD
- AT&T limita la distribución del código de UNIX a partir de la v7
  - se dificulta el acceso al código fuente
  - System III: primera versión comercial de UNIX (1982)
- Dos líneas principales: System V y BSD

AT&T System V

- A partir de UNIX Versión 6 y 7, AT&T lanza, en 1982, la primer versión de la linea comercial de UNIX: System III
- SysIII carecía de innovaciones como `vi` y `csh`
- En 1983 surge System V. Incluía algunas características de los sistemas BSD (p.e. `vi`, `curses`,...)
- En 1984 surge la SysV Release 2 y en 1987 la SVR3
- Finalmente, SysV Release 4 aparece en 1988
- SVR4 combina SVR3, 4.3BSD, XENIX (Microsoft), SunOS (Sun Microsystems) y agrega nuevas utilidades

Berkeley System Distribution

- Thompson, Bill Joy (co-fundador de Sun) y Chuck Haley (1975).
- Second Berkeley Software Distribution (2BSD), 1978, incorpora el editor vi (versión visual de ex) y el C shell.
- En 1979, 3BSD, combina 2BSD con UNIX v7.
- DARPA (*Defense Advanced Research Projects Agency*) colabora con las nuevas versiones 4BSD: 4.1BSD, 4.2BSD y en 1986 4.3BSD (implementación de TCP/IP).
- Conflicto con AT&T por el uso de código propietario.
- Su última versión es 4.4BSD-Lite Rel. 2 (1995), sin código propietario AT&T. En ella se basan muchas variantes:
    - FreeBSD, OpenBSD, NetBSD, Darwin (base de OS X e iOS), etc.

Otras versiones

La mayoría de los UNIX históricos y actuales derivan de System V o BSD, o son una mezcla de los dos

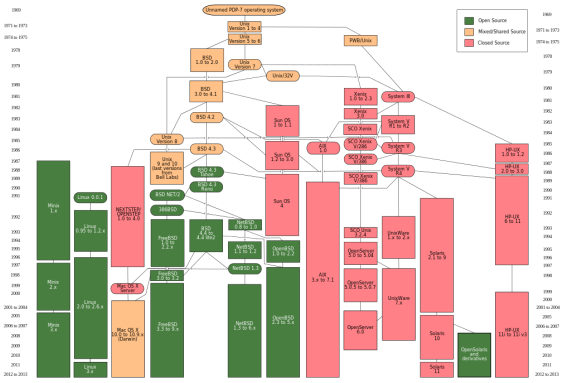
- XENIX: desarrollada por Microsoft en 1980 para uso en microprocesadores, derivada del AT&T UNIX v7
- SCO OpenServer (antes SCO UNIX): derivada de XENIX y desarrollada por *Santa Cruz Operation*, hoy propiedad de Xinuos
- UnixWare: desarrollado por Novell a partir de System V, ahora propiedad de Xinuos
- SunOS: desarrollado por Sun Microsystems (ahora Oracle), en 1982, basado en BSD
- Mach: microkernel desarrollado en la Carnegie-Mellon University, basado en 4.3BSD
- XNU: desarrollado por Apple, kernel basado en Match, que forma parte de Darwin
- OSF/1 (Open Software Foundation): DEC, IBM y HP desarrollan un UNIX para competir con System V y SunOS:

    - Basado en el kernel Mach
    - Llamado después Digital UNIX y Tru64
- GNU Hurd: conjunto de servicios que corren encima de GNU Mach formando el kernel del SO de GNU
- Minix: escrito por Andrew S. Tanenbaum de la Vrije Universiteit, para correr en los IBM PCs
- Linux: kernel desarrollado por Linus Torvals, primera versión en 1991
- Android: basado en el kernel Linux, desarrollado por Google para móviles y tablets

Versiones comerciales

- Oracle: Oracle Solaris (evolución de SunOS versión 5 y SVR4), versiones para Sparc y x86, última versión Solaris 11 (versiones *open source* OpenSolaris (discontinuada), illumos, OpenIndiana)
- IBM: AIX (*Advanced Interactive eXecutive*) para servidores IBM, basado en OSF/1 y SVR4, última versión AIX 7.1
- HP: HP-UX, versiones para PA-RISC e Itanium, variante System V con características de OSF/1, última versión 11i
- SGI: IRIX basado en System V con extensiones BSD, para sistemas MIPS; última versión 6.5 (2006)
- Xinuos: OpenServer X (basado en FreeBSD), SCO OpenServer 6 y UnixWare 7
- Apple: Mac OS X, con dos partes Darwin + Aqua (GUI); Darwin basado en Mach y BSD

Evolución de UNIX



Más detalles en http://www.levenez.com/unix/

1.2. Sistemas GNU/Linux

Linux:

1. En agosto de 1991, el estudiante finlandés Linus Torvals, presenta en Internet la versión 0.01 del kernel de un nuevo SO, inspirado en MINIX (aunque sin código de MINIX)
    - Esta primera versión tenía poco más de 10.000 líneas de código
2. En 1992, Linux se libera bajo licencia GPL
3. A través de Internet, muchos programadores se unieron al proyecto
4. En 1994 Linux alcanzó la versión 1.0
5. En 2003, llegamos a la versión 2.6, con casi 6 millones de líneas de código
6. En 2011, versión 3.0, en 2015 versión 4.0 (última 4.2)

GNU:

- El proyecto GNU (*GNU's Not Unix*) fue iniciado en 1983 por Richard Stallman bajo los auspicios de la Free Software Foundation (ver noticia)
    - Objetivo: crear un sistema operativo completo basado en *software libre*, incluyendo herramientas de desarrollo de software y aplicaciones
- En el momento de la liberación, GNU no tenía listo su kernel

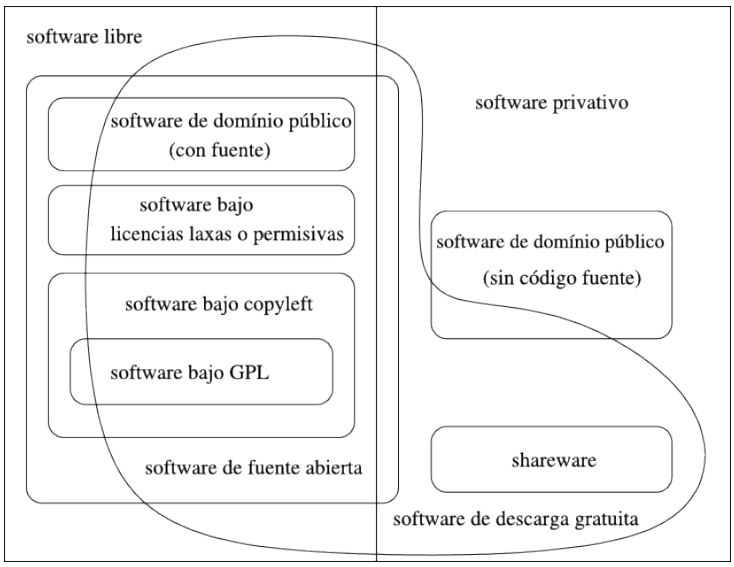
- Linux fue adaptado para trabajar con las aplicaciones de GNU: Sistema GNU/Linux
  1. Kernel Linux +
  2. Aplicaciones GNU: compilador (gcc), librería C (glibc) y depurador (gdb), shell bash, GNU Emacs, GNOME, Gimp,...
- GNU tiene ahora su propio kernel: GNU Hurd

Mascotas

Características de Linux

1. Sistema operativo de código abierto, multitarea y multiusuario
2. Portable (corre en arquitecturas Intel x86 y IA64, Sparc, MIPS, PowerPC, Alpha, PARisc,...)
3. Soporte para multiprocesador
4. Soporte para múltiples sistemas de ficheros
5. Kernel de tipo monolítico con módulos cargables dinámicamente

Software Libre y Open Source



Software libre (*free software*):

- Movimiento que parte de las ideas de Richard Stallman
- El software, una vez obtenido puede ser usado, copiado, estudiado, modificado y redistribuido
- La distribución no tiene que ser necesariamente gratuita

Open Source (o software de código abierto):

- Posibilidad de acceder al código fuente, y modificarlo y distribuirlo dentro de una determinada licencia de código abierto (ver www.opensource.org/licenses)

- La Open Source Initiative fue fundada en febrero de 1998 por Bruce Perens y Eric S. Raymond para la certificación de software Open Source

FLOSS Free/Libre/Open-Source Software

- Software libre y open software

Diferencia entre ellos principalmente filosóficas

- Código abierto: es una metodología de programación
- Software libre: asociado a la libertad del usuario

Ejemplo de la diferencia: dispositivos *tiranos* o *tivoized*
Más información: www.gnu.org/philosophy/

Licencia GPL

La licencia GPL (GNU General Public License) :

1. Bajo GPL el software puede ser copiado y modificado
2. Las modificaciones deben hacerse públicas bajo GPL (copyleft)
3. Se impide que el código se mezcle con código propietario

La licencia LGPL (GNU Lesser General Public License) permite integrar el software con software propietario

- Pensado para librerías que pueden ser usadas en el desarrollo de software propietario

Más información sobre licencias:

- Introducción a las licencias
- Varias licencias y comentarios

Distribuciones de GNU/Linux

Colección de software que forma un S.O. basado en el kernel Linux; normalmente incluye:

1. El kernel Linux
2. Las aplicaciones GNU (o parte de ellas)

3. Software de terceros, libre o propietario: X Windows, servidores, utilidades,...

Las distribuciones difieren en el empaquetado de los programas (RPM, deb, tgz), el programa de instalación y herramientas específicas

- Lista de distribuciones en wikipedia: en.wikipedia.org/wiki/List_of_Linux_distributions
- Timeline de distribuciones
- Información interesante en http://www.distrowatch.com

Algunas de las más populares son Debian, Red Hat (Fedora), Mandriva (Mageia), Slackware, SuSE, Gentoo, Ubuntu...

Debian

- Distribución totalmente libre, sin fines comerciales
- Tres ramas en la distribución:
  1. *Stable*: destinada a entornos de producción (desde abril 2015, versión 8.0 *jessie*)
  2. *Testing*: software más nuevo, en fase de prueba (actualmente *stretch*)
  3. *Unstable*: en fase de desarrollo (siempre *sid*)
- Versiones anteriores:
  - 7.0 wheezy, mayo 2013
  - 6.0 squeeze, febrero 2011
  - 5.0 lenny, febrero 2009
  - 4.0 etch, abril 2007
  - 3.1 sarge, junio 2005
  - 3.0 woody, julio 2002
  - 2.2 potato, agosto 2000
  - 2.1 slink, marzo 1999
  - 2.0 hamm, julio 1998
  - 1.3 bo, junio 1997
  - 1.2 rex, diciembre 1996
  - 1.1 buzz, junio 1996
- Algunas características
  1. Gran número de aplicaciones disponibles
  2. Potente formato de empaquetado: paquetes DEB y herramienta APT
  3. Instalación y cambio de versiones a través de red

Ubuntu

- Distribución enfocada a ordenadores de escritorio (*Desktop Computers*), aunque existe la versión para servidores
- Basada en Debian, Ubuntu concentra su objetivo en la usabilidad, lanzamientos regulares y facilidad en la instalación
- Patrocinado por Canonical Ltd., una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth
- Última versión: Ubuntu 15.04 (*Vivid Vervet*), fue lanzada el 23 de abril de 2015
- Próxima versión: Ubuntu 15.10 (*Wily Werewolf*) prevista para el 22 de octubre de 2015
- Última versión con soporte a largo plazo: Ubuntu 14.04 LTS (*Trusty Tahr*)
- Proyectos relacionados: kubuntu, edubuntu, xubuntu

Red Hat

- Una de las principales firmas comerciales del mundo GNU/Linux
- Fundada por Marc Ewing y Bob Young en 1994
- Inicialmente, proporcionaba distribuciones para el usuario individual (versiones personal y profesional), y orientadas a empresas (versión Enterprise)
- Introduce el formato de empaquetado RPM (*RedHat Package Manager*)
- Desde 2002, orientado en exclusiva al mercado corporativo
  - Cede la última distribución personal (RH 9) a la comunidad $\longrightarrow$ aparece el proyecto Fedora
- Última versión: Red Hat Enterprise Linux 7 (*Maipo*) desde junio de 2014

- Distribuciones libres que clonan RHEL: CentOS, Scientific Linux, ClearOS, etc.

Fedora

- Objetivo: construir un SO completo, de propósito general basado exclusivamente en código abierto
- Parte de la versión Red Hat 9
- Mantiene el sistema de paquetes RPM
- Última versión: Fedora 22, 26 de mayo de 2015

Slackware

- Una de las primeras distribuciones: creada en 1993 Patrick Volkerding
- Orientada hacia usuarios avanzados:
- Ultima versión: Slackware 14.1 (4 de noviembre de 2013)

SuSE Linux

- Compañía alemana fundada en 1992, subsidiaria de *Micro Focus International*
- Originalmente basada en Slackware
- Herramienta de configuración gráfica: YaST (*Yet Another Setup Tool*)
- Principales versiones: SUSE Linux Enterprise Server y SUSE Linux Enterprise Desktop
- Versión open source: openSUSE, última revisión 13.2 (4 de noviembre de 2014)

Gentoo Linux

- Distribución orientada a permitir la máxima adaptabilidad y rendimiento

- puede ser optimizada y configurada automáticamente para el uso en un sistema concreto

- Portage: Sistema de distribución, compilación e instalación de software

Arch Linux

- Distro ligera y flexible centrada en la elegancia, corrección del código, minimalismo, y simplicidad (KISS)

- Gestor de paquetes Pacman

Otras distribuciones

- Existen cientos de distribuciones diferentes de Linux

  - Adaptadas a diferentes necesidades: seguridad, multimedia, sistemas viejos, análisis forense, clusters...
  - Suelen estar basadas en las principales distribuciones

- Ejemplos (ver distrowatch.com):

  1. Sistemas basados en Debian/Ubuntu: LinuxMint, Knoppix y derivados (BAcktrack, Damn Small...), Trisquel, Minino, Guadalinex, ...
  2. Sistemas basados en RedHat/Fedora: Mageia, PCLinuxOS, Oracle Linux, Springdale, Berry Linux, Kororaa, Tinyme, Rocks...
  3. Sistemas basados en Slackware: SLAX, Zenwalk, Vectorlinux, Porteus, Absolute...
  4. Sistemas basados en Gentoo: Funtoo, Sabayon, Pentoo, Toorox...
  5. Sistemas basados en Arch: Parabola, Manjaro, Archbang, Chakra...

2. Instalación del sistema y de software

A la hora de instalar un sistema, tenemos que tener en cuenta el tipo de funciones que va a desempeñar.
Podemos distinguir:

1. Sistema de escritorio: usado en tareas rutinarias (ofimática, acceso a Internet, etc.)

2. Estación de trabajo (*workstation*): sistema de alto rendimiento, generalmente orientado a una tarea específica
    - estación dedicada al cálculo (p.e. aplicaciones científicas)
    - estaciones gráficas (p.e. diseño 3D)
3. Servidores: ofrecen servicios a otras máquinas de la red
    - servicios de disco, impresión, acceso a Internet, filtrado, etc.

2.1. Tipos de servicios

Un sistema servidor ofrece servicios al resto de sistemas de la red:

1. Aplicaciones
    - servicios de terminales, conexión remota (telnet, ssh), aplicaciones gráficas a través de X Window, aplicaciones web, etc.
2. Ficheros
    - acceso a ficheros a través de FTP,
    - servicio transparente a través de NFS o Samba
3. Impresión
    - servir impresoras locales o remotas a otros sistemas UNIX o Windows
4. Servicios de información de red, por ejemplo, NIS, NIS+ o LDAP
    - permiten centralizar la información de las máquinas, usuarios y recursos
5. Servicios de configuración dinámica de máquinas
    - DHCP (*Dynamic Host Configuration Protocol*): permite configurar dinámicamente la red de los clientes
6. Correo electrónico
    - agentes MTA (*Mail Transfer Agent*) para recuperar y retransmitir correo, o servicios de POP o IMAP
7. Servidor Web (p.e. Apache)

8. Servicio de nombres (DNS)
9. Servicio de base de datos
10. Servicios de acceso a Internet: NAT, proxy
11. Servicios de filtrado (firewall)

2.2. Virtualización

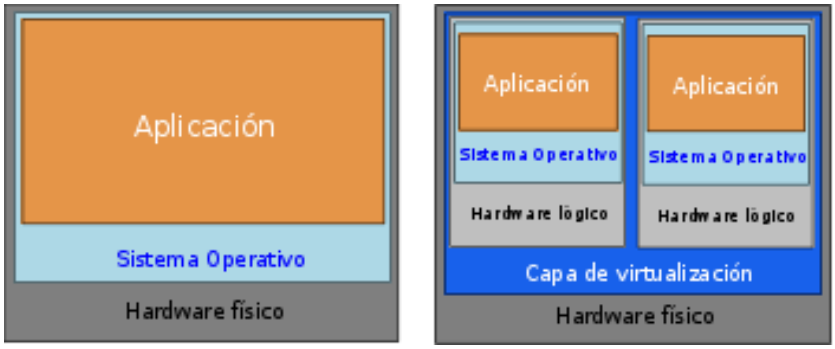
Abstracción de un conjunto de recursos computacionales para que puedan ser utilizados de forma más conveniente

- Memoria virtual
- Sistemas RAID o LVM
- Virtualización de servidores

Virtualización de servidores

- Máquina virtual
  - Entorno virtual entre el sistema real y el usuario final que permite que este ejecute un software determinado
  - Normalmente usado para ejecutar varios sistemas operativos simultaneamente sobre el mismo hardware
- Usos de la virtualización
  - Consolidación de servidores
  - Execución de aplicacións non-fiables
  - Recuperación de desastres
  - Pruebas y desarrollo de software
  - Computación elástica (cloud computing)

Conceptos:



- Sistema anfitrión (*host*): SO ejecutado sobre la máquina real
- Sistema huesped (*guest*): SO ejecutado sobre la máquina virtual

Algunas herramientas de virtualización:

- VirtualBox desarrollado originalmente por la empresa alemana Innotek, ahora propiedad de Oracle; version Open Source (VBox OSE) y propietaria
- QEMU emulador/virtualizador de código abierto desarrollado por Fabrice Bellard
- KVM virtualización asistida por hardware, utiliza una versión modificada de QEMU como front-end.
- Xen desarrollado inicialmente en la universidad de Cambridge, versiones comerciales Citrix XenServer, Oracle VM,...
- VMWare Workstation programa propietario de VMware Inc.; es uno de los más conocidos (versiones para Windows y Linux)
- Hyper-V herramienta de Microsoft Windows

Una comparativa en wikipedia
Tipos de virtualización:

- Emulación (o recompilación dinámica): la máquina virtual simula el hardware completo
  - Permite ejecutar SOs para sistemas diferentes del anfitrión
  - Normalmente es lenta
  - Ejemplos: Bochs, PearPC, QEMU sin aceleración,...

- Paravirtualización: la máquina virtual no simula todo el hardware, sino que ofrece una API especial
  - Requiere modificacións en el SO huesped
  - Velocidad nativa
  - Ejemplos: Xen
- Virtualización completa: la máquina virtual sólo simula el hardware necesario para permitir que un SO huesped se pueda ejecutar
  - El SO huesped debe ser para el tipo de arquitectura del host
  - Velocidad cerca de la nativa
  - Ejemplos: VMWare, QEMU con aceleración, Parallels Desktop for Mac, etc.
- Virtualización asistida por hardware
  - El hardware del anfitrión proporciona soporte para mejorar la virtualización: x86 virtualization, (Intel VT o AMD-V)
  - Velocidad similar a la paravirtualización sin necesidad de modificar el huesped
  - Ejemplos: Xen, VirtualBox, KVM, VMWare, Parallels Workstation, etc.
- Virtualización a nivel de SO: aisla varios servidores sobre el SO anfitrión
  - También llamados Contenedores Software
  - Los SO huespedes son los mismos que el anfitrión, ya que usan el mismo kernel
  - Ejemplos: User-mode Linux, FreeBSD Jail, Linux-VServer, Docker,...

2.3. Instalación de Linux Debian

Para detalles de instalación ver Guía de instalación de Debian

- Descargaremos la imagen de CD pequeño (fichero `debian-8.2.0-i386-netinst.iso`)

- `Enter` para iniciar con opciones por defecto, `Advances options` para opciones de instalación avanzadas, `Help` para ayuda

Siguientes pasos en la instalación[1]

- Selección de idioma, localización y teclado
- Configuración de la red
  - Por defecto, intenta configurarla por DHCP
  - Si no lo consigue, pasa a configuración manual (indicar IP, máscara, pasarela y DNSs)
- Poner un nombre a la máquina e indicar el dominio (si alguno)
- Fijar el password del superusuario (root) y crear un usuario no privilegiado

Cuenta del superusuario

- El superusuario es un usuario especial que actúa como administrador del sistema

[1] En cualquier momento de la instalación tenemos acceso a una consola pulsando `Alt-F2`; usar `Alt-F1` para volver a la instalación

    - Tiene acceso a todos los archivos y directorios del sistema
    - Tiene capacidad para crear nuevos usuarios o eliminar usuarios
    - Tiene capacidad de instalar y borrar software del sistema o aplicaciones
    - Puede detener cualquier proceso que se está ejecutando en el sistema
    - Tiene capacidad de detener y reiniciar el sistema
- El login del superusuario es `root` (aunque puede cambiarse)
- No es conveniente acceder al sistema directamente como root:
    - acceder como un usuario sin privilegios, y
    - obtener los permisos de root haciendo `su` (necesitamos la contraseña de root)

Elección de contraseña

- Tener una contraseña de root adecuada es básico para la seguridad de un sistema
- Las contraseñas de usuario también deberían ser adecuadas
- Recomendaciones para elegir una contraseña:
    - No usar el nombre de usuario (login) ni variantes de este (p.e. login: pepe, passwd: pepe98)
    - No usar el nombre real del usuario ni los apellidos
    - No usar palabras contenidas en diccionarios, o palabras de uso común
    - Usar más de 6 caracteres para la contraseña
    - Mezclar caracteres en mayúsculas y minúsculas, con caracteres no alfabéticos (números, signos de puntuación, etc.)
    - Usar contraseñas fáciles de recordar, para evitar tener que apuntarlas
    - Cambiar la contraseña con frecuencia (p.e. una vez al mes)
- La contraseña se cambia con el comando `passwd`
    - `passwd`: cambia la contraseña (password) del usuario

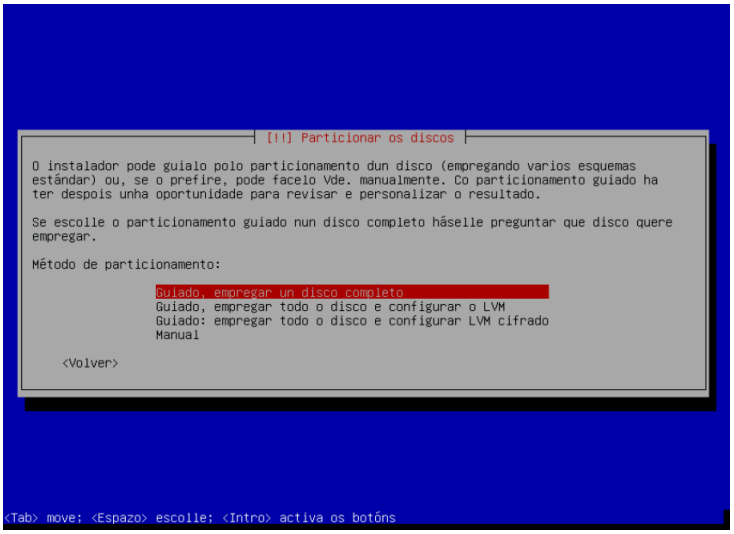
- Ejemplo: usuario pepe

```
# passwd
Changing password for pepe
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Continuación de la instalación

En una instalación por red los paquetes se traen de un repositorio remoto a través de http o ftp

- Seleccionar el huso horario
- Realizar el particionado del disco (modo guiado o manual)



Particionado del disco

Podemos optar por instalar todo el sistema en una sola partición, aunque no es nada recomendable

- preferible instalar diferentes directorios del sistema en diferentes particiones
- la estructura de directorios UNIX sigue el estándar FHS (*Filesystem Hierarchy Standard*)

Filesystem Hierarchy Standard

Localización estándar de los ficheros

- `/bin/` (***bin****aries*) - ejecutables esenciales (`ls`, `cat`, `bash`, etc.)
- `/sbin/` - (***su****peruser* ***bin****aries*) - ejecutables esenciales para el superusuario (`init`, `ifconfig`, etc.)
- `/lib/` - Librerías esenciales para los ejecutables en `/bin/` y `/sbin/`
- `/usr/` (***U****nix* ***s****ystem* ***r****esources*) - aplicaciones y código fuente usados por los usuarios y el superusuario
  - `/usr/bin/` - más aplicaciones de usuario
  - `/usr/sbin/` - más aplicaciones para el superusuario
  - `/usr/lib/` - librerías esenciales para los ejecutables en `/usr/bin/` y `/usr/sbin/`
  - `/usr/share/` - datos, independientes de la arquitectura, necesarios para las aplicaciones y páginas de manual (`/usr/share/man`, `/usr/share/info`)
  - `/usr/include/` - ficheros de cabecera (`.h`) estándar
  - `/usr/src/` (opcional) - código fuente (del kernel u otras aplicaciones)
  - `/usr/X11R6/` (opcional) - sistema X Window, versión 11 release 6
  - `/usr/local/` - aplicaciones que no son parte del sistema operativo
- `/etc/` - contiene muchos de los scripts y ficheros de configuración del sistema
  - `/etc/X11/` (opcional) - configuración de X Window
  - `/etc/skel/` (opcional) - ficheros de configuración para los usuarios
- `/var/` - ficheros **var**iables (logs, bases de datos, etc.)
  - `/var/log/` - ficheros de log
  - `/var/spool/` - ficheros temporales de impresión, e-mail y otros
- `/tmp/` - ficheros temporales
- `/opt/` - otras aplicaciones software (estáticas)

- `/srv/` - datos de servicios proporcionados por el sistema (páginas web, ftp, cvs, etc.)
- `/boot/` - ficheros usados por el gestor de arranque, incluyendo el kernel

Otros directorios del sistema

- `/` - directorio raíz del sistema
- `/home/` (opcional) - directorio de usuarios (directorio inicial o *home*)
- `/root/` (opcional) - directorio *home* del superusuario
- `/dev/` - ficheros de acceso a periféricos
- `/proc/` - directorio virtual conteniendo información del sistema
- `/sys/` - similar a `/proc`, contiene información de dispositivos (sólo kernel 2.6)
- `/media/` - punto de montaje para medios removibles
- `/mnt/` - punto de montaje para sistemas temporales

Para más información ver www.pathname.com/fhs/

Esquemas de particionamiento

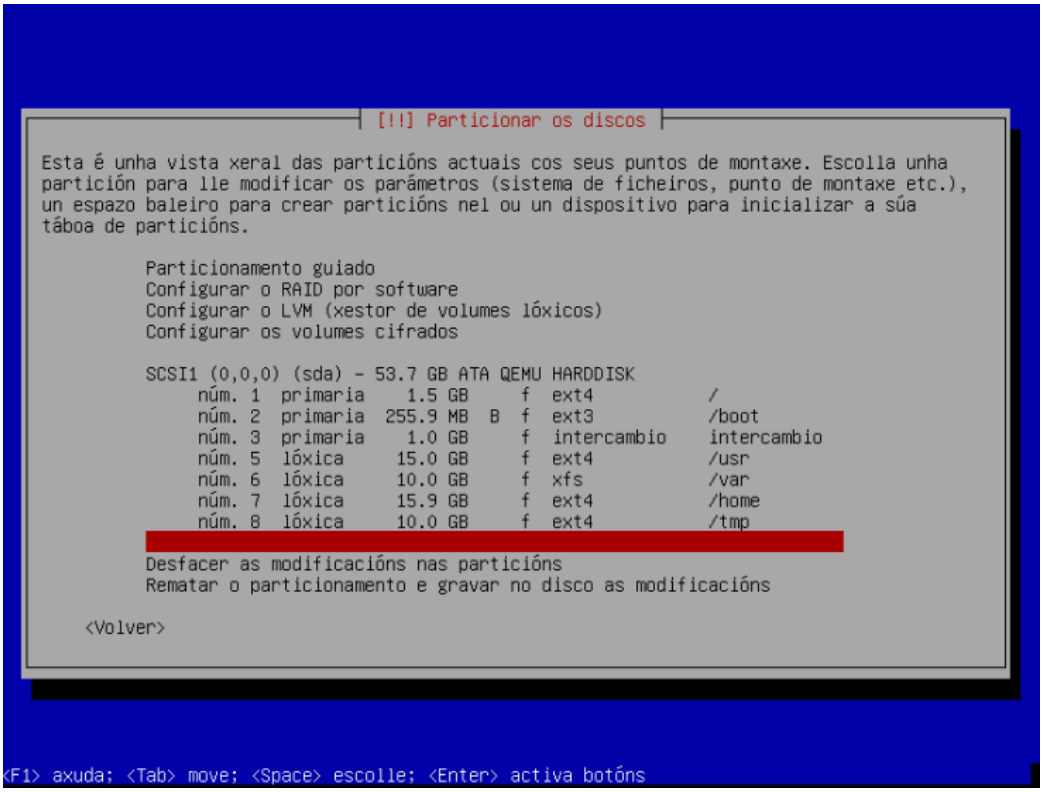
Dependiendo del tipo de sistema podemos escoger diferentes esquemas de particionamiento, algunos ejemplos:

- Maquina de escritorio (un sólo usuario), tres particiones
  - `swap` - área de intercambio; siempre necesaria, tamaño función del tamaño de la RAM y del tipo de aplicaciones que se ejecuten (como orientación, tomar al menos el doble de la RAM)
  - `/home/` - disco de los usuarios, tamaño en función de las necesidades del usuario
  - `/` - resto del disco
- Sistema multiusuario, además de las particiones anteriores crear particiones separadas para `/usr`, `/var` y `/tmp`
  - `/usr` podría montarse en modo sólo-lectura después de que todo el sistema esté instalado (dificulta la introducción de Troyanos)

- tener `/var` y `/tmp` en su partición evita que un usuario llene todo el disco

- Particiones adicionales:
  - `/boot` - en versiones antiguas de Linux se necesitaba que el directorio `/boot/` estuviese por debajo del cilindro 1024
  - `/chroot` - para aplicaciones en un entorno *enjaulado* (p.e. DNS, Apache, etc.)
  - `/var/lib` - partición para gestionar ficheros del servidor de bases de datos o del proxy (MySQL, squid) (limitar la posibilidad de un ataque por denegación de servicio)

Ejemplo de partición (disco de 50 G):



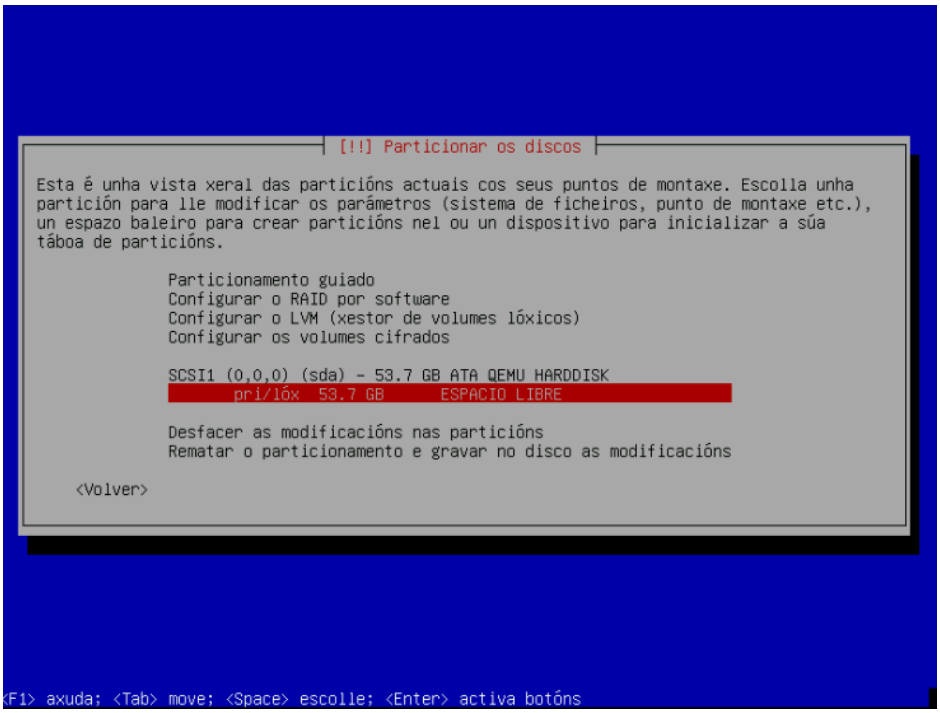
Particionamiento durante la instalación

Dos opciones:

- Particionamiento guiado (con o sin LVM)
  - Selecciona el tamaño de las particiones de manera automática
- Particionamiento manual
  - Particionamiento manual
    - control total del número y tamaño de las particiones

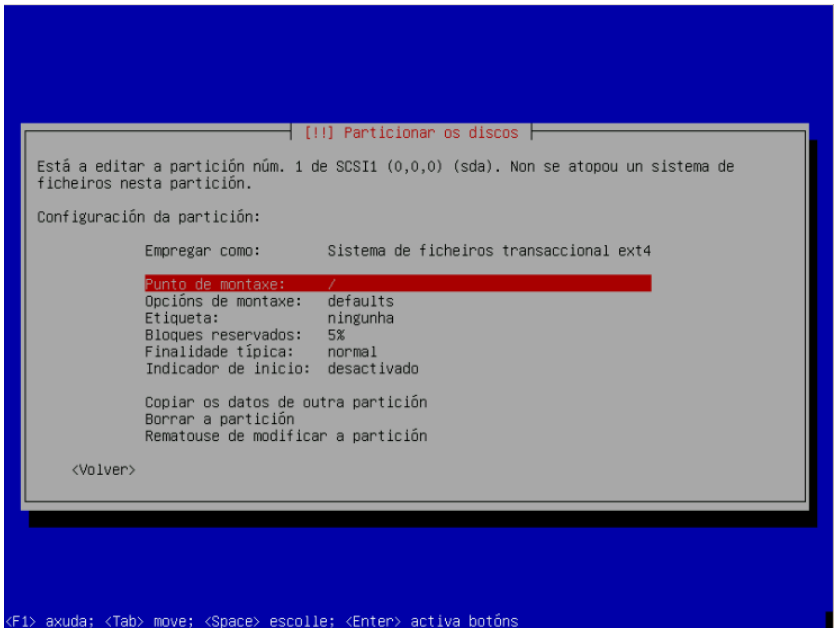
Particionamiento manual

1. Seleccionamos el disco a particionar y crear nueva tabla de particiones:



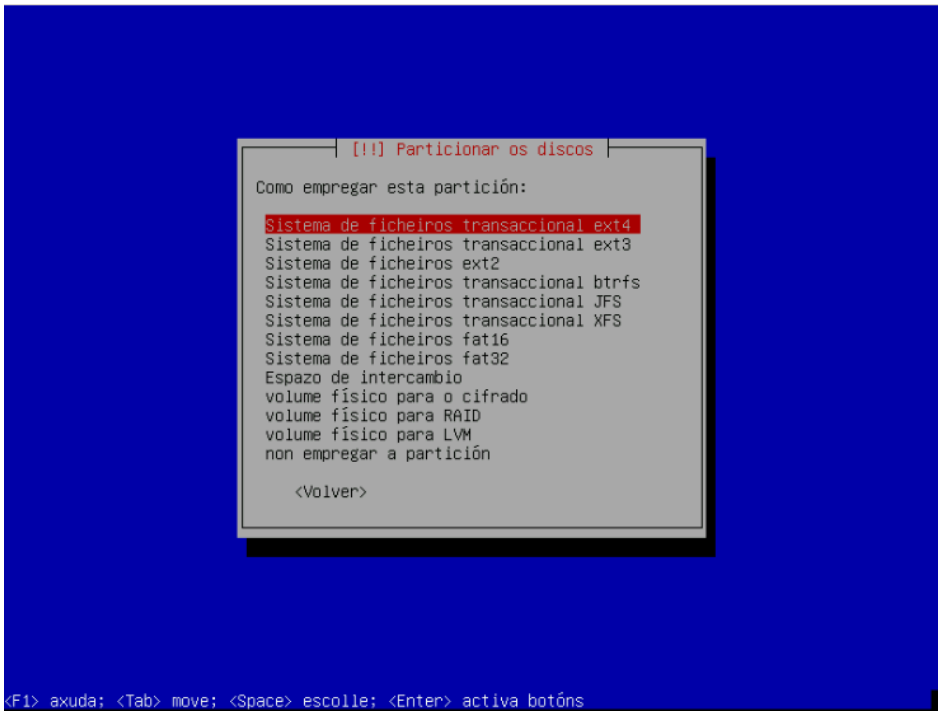
2. Creamos una nueva partición indicándole el tamaño, el tipo (primaria o lógica) y la localización (comienzo o final)
   - puede haber 4 primarias o 3 primarias y una extendida, que se puede dividir en varias lógicas

Ejemplo de partición para /



Sistemas de ficheros

Linux soporta múltiples sistemas de ficheros
Para cada partición podemos seleccionar los siguientes:



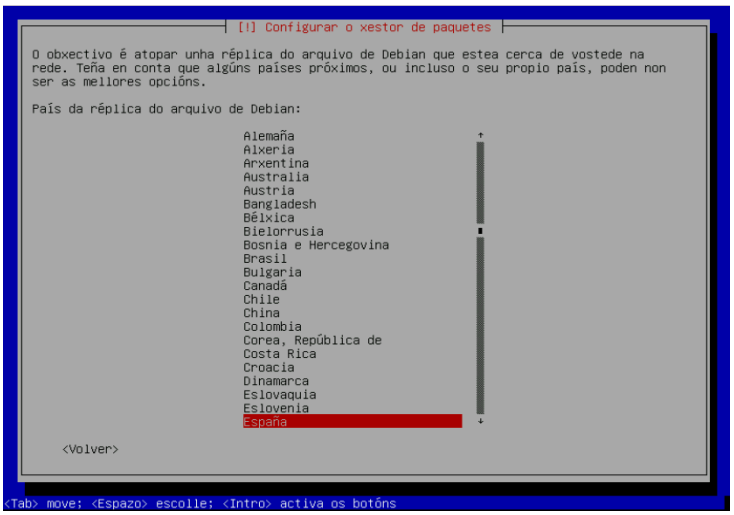
- `ext2` - *Second EXTended filesystem*, sistema estándar Linux
- `ext3` - *Third EXTended filesystem*, versión con *journal* de ext2, que evita corrupción (opción por defecto)
  - es posible convertir ext2 en ext3 con el comando *tune2fs -j*
  - muy robusto, aunque no escala muy bien (no ideal para filesystems muy grandes, ficheros muy grandes o un número de ficheros en un directorio muy alto)
- `ext4` - *Fourth EXTended filesystem*, última versión, disponible desde el kernel 2.6.28, mejoras en velocidad y otros aspectos
- `ReiserFS`, `JFS`, `XFS` - otros tipos de sistemas transaccionales (con *journal*) usados en diferentes sistemas
  - `ReiserFS` - por defecto en algunas distribuciones Linux (p.e. Slackware)
    - mayor rendimiento que `ext2` y `ext3`, principalmente con ficheros pequeños (menos de 4k) y buena escalabilidad
    - Sucesor: Reiser4
  - `XFS` - usado en sistemas SGI Irix

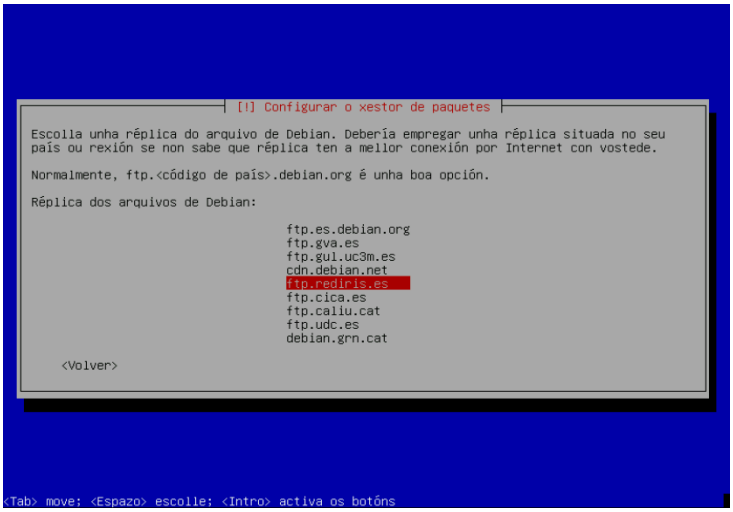
    - optimizado para escalabilidad
    - recomendado en grandes sistemas SCSI o *fiber channel* con fuente de alimentación ininterrumpida (utiliza caché de forma agresiva $\longrightarrow$ perdida de datos si el sistema se apaga)
  - `JFS` - usado en máquinas de IBM
- `fat16`, `fat32` - usados en MS-DOS y Windows 95/98/Me

Comparativa en wikipedia

Últimos pasos en la instalación

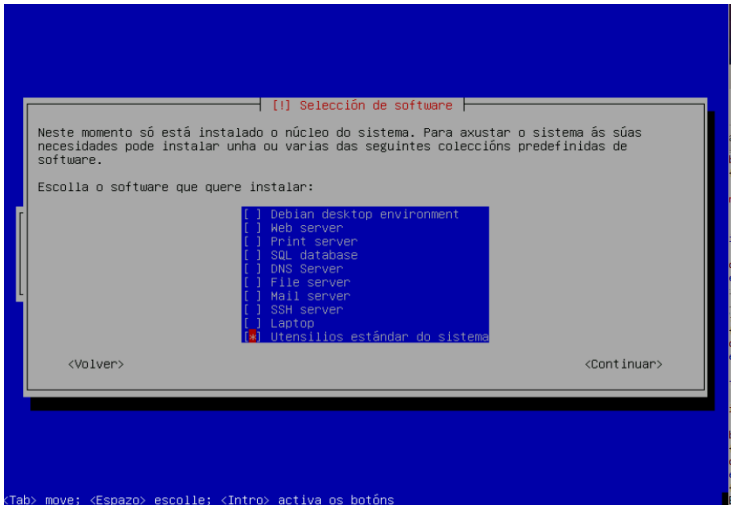
- Debemos seleccionar el *mirror* desde el que descargar el software
  - Existen varios repositorios de paquetes Debian $\longrightarrow$ elegir el más cercano
  - Introducir la información del proxy, en caso de ser necesario





- Seleccionar los paquetes software a instalar
- Instalar del gestor de arranque

Selección de paquetes



- Elegir los paquetes a instalar:
  - aunque optemos por no instalar nada, se instalarán todos los paquetes con prioridad “estándar”, “importante” o “requerido” que aún no estén instalados en el sistema
- Podemos repetir este paso con el sistema instalado usando el comando `tasksel`

Instalación del gestor de arranque

Gestor de arranque: permite seleccionar el SO a arrancar
Existían 2 posibilidades en Linux

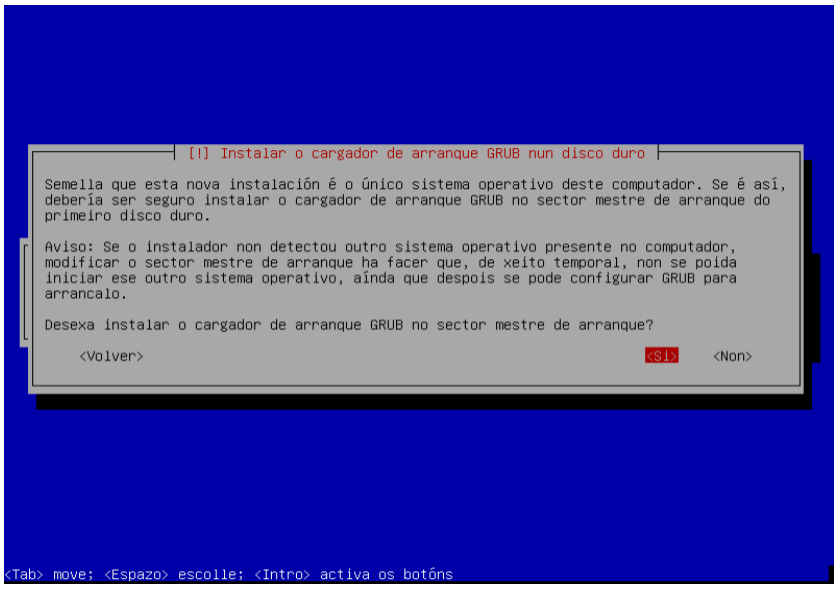
- LILO (*LInux Loader*), cargador clásico en Linux (obsoleto)
- GRUB (*GRand Unified Bootloader*), cargador del proyecto GNU

La gran mayoría de las distribuciones usan GRUB (las más actuales la versión 2)
El gestor de arranque se carga normalmente en el MBR del primer disco

- MBR (*Master Boot Record*) está localizado en el primer sector del disco
- en el MBR se encuentra información sobre las particiones (*Master Partition Table*) y un pequeño código (*Master Boot Code*)
- cuando el sistema se inicia, la BIOS carga el *Master Boot Code*, que permite seleccionar el sistema a cargar, y transfiere el control al programa de arranque del SO (localizado en `/boot`)

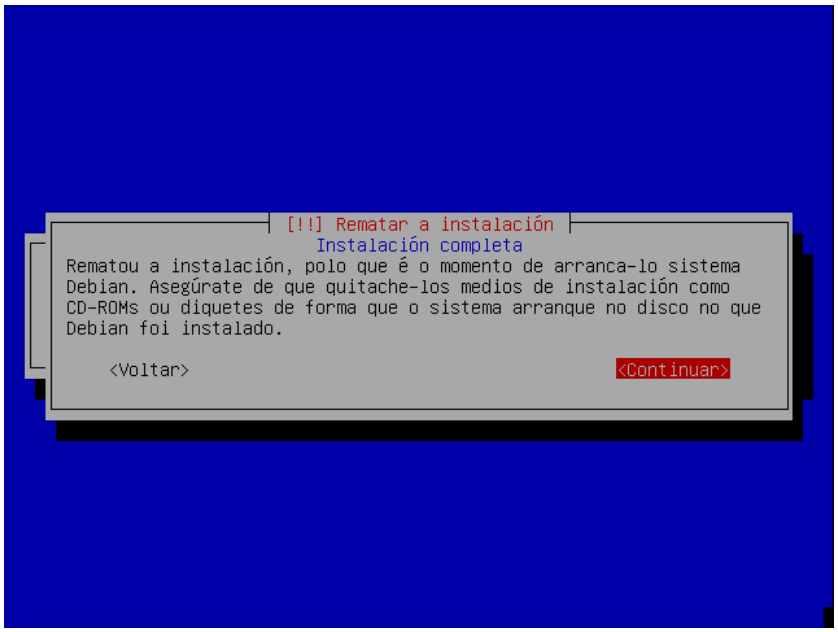
El gestor de arranque puede también cargarse en el primer sector de la partición root (por si tenemos otro bootloader en el MBR)

Instalación de GRUB en Debian



Finalización de la instalación

Debian: la instalación termina aquí



Debemos reiniciar el sistema para continuar

Logical Volume Management (LVM)

Proporciona una visión de alto nivel de los discos

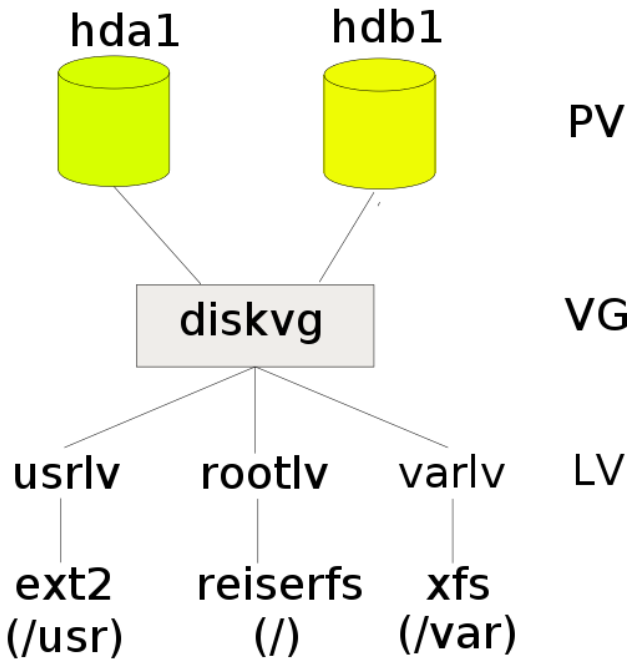
- permite ver varios discos como un único volumen lógico
- permite hacer cambios en las particiones sin necesidad de reiniciar el sistema
- permite gestionar los volúmenes en grupos definidos por el administrador

Conceptos (para más información LVM HOWTO):

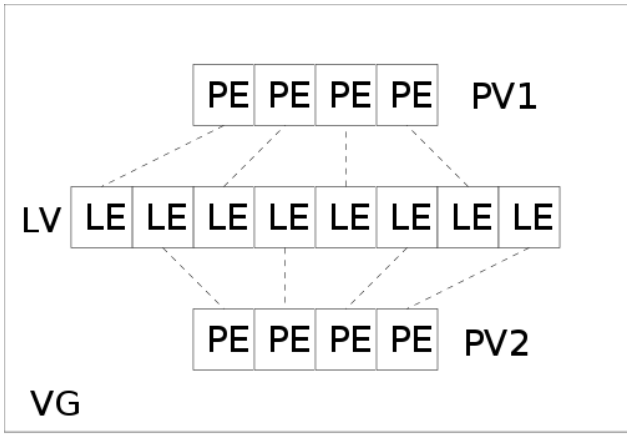
- **Volumen físico (PV)**: discos duros, particiones de los discos u otro dispositivo similar (p.e. RAID)
- **Volumen lógico (LV)**: *particiones* lógicas sobre las que se montan los sistemas de ficheros
- **Grupo de volúmenes (VG)**: agrupación de LV, que forman una unidad administrativa

- **Extensión física (PE)**: unidades básicas en las que se divide cada PV; el tamaño de cada PE es el mismo para todas los PV pertenecientes al mismo VG
- **Extensión lógica (LE)**: unidades básicas en las que se divide cada LV; para un VG el tamaño de las LE es el mismo que el de las PEs
- **Área de descripción del VG (DAVG)**: área donde se almacena la información (meta-data) sobre los LV y VG; sería el equivalente a la tabla de particiones de un sistema no-LVM

Estructura de LVM



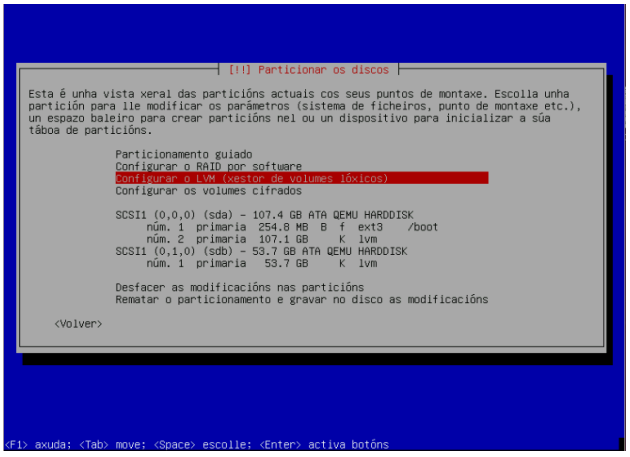
Hay una relación 1:1 entre cada LE y PE en un VG



- Se pueden elegir dos estrategias para mapear extensiones lógicas en extensiones físicas:
  - Mapeado Lineal: asigna un rango de PEs a un área de un LV en orden, por ejemplo LE 1-99 se mapean a PV1, y los LE 100-199 se mapean a PV2
  - Stripping: se reparten los LEs entre los distintos PVs
    - 1 LE → PV1[1], 2 LE → PV2[1], 3 LE → PV3[1], ...

Pasos para crear un sistema LVM
Suponemos un sistema con dos discos (`sda` y `sdb`)

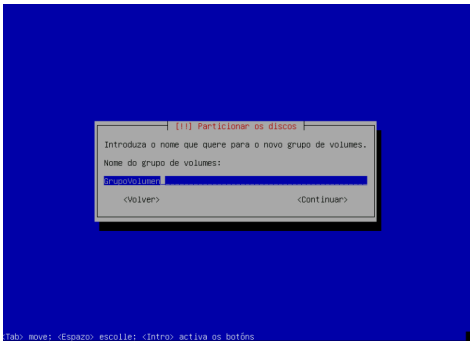
1. Crear los PV

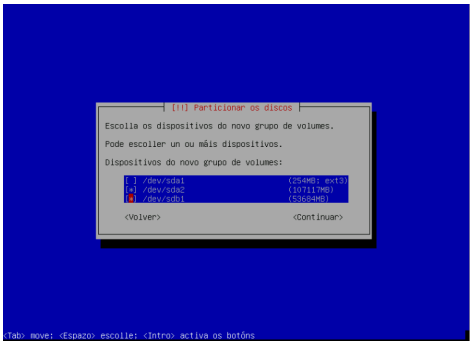


   - particionamos `sda` para reservar un espacio para `/boot` (dejamos `/boot` fuera de LVM para evitar problemas con el arranque, aunque en las últimas versiones no es necesario)

- definimos 2 volúmenes físicos
  - el primero incluye todo `sda` menos `/boot` (`sda2`)
  - el segundo incluye todo `sdb` (`sdb1`)

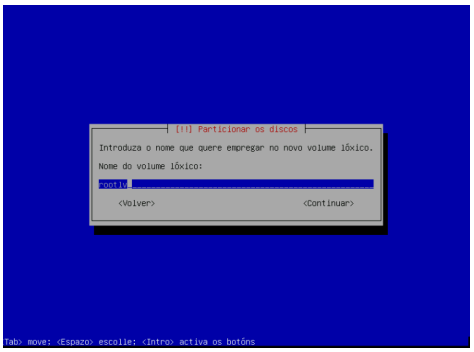
2. Crear un grupo de volumen que incluya los PVs

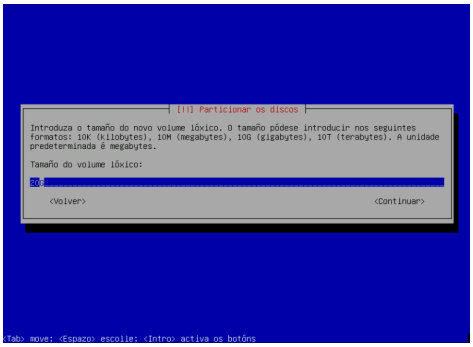




- podemos ponerle un nombre al grupo de volumen
- hacemos que incluya los dos volúmenes físicos que hemos definido en el punto anterior

3. Crear los volúmenes lógicos

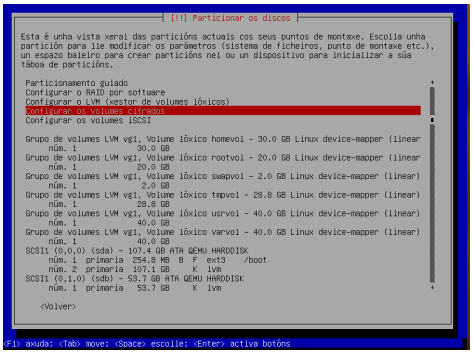


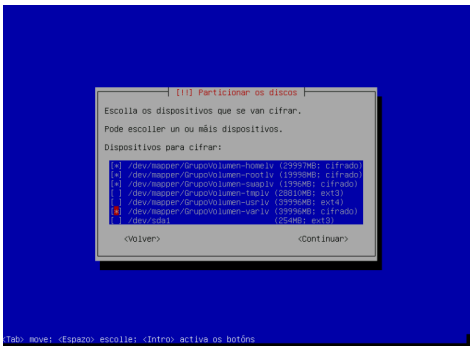


- creamos un volumen lógico por cada partición
- los LV pueden llevar un nombre identificativo

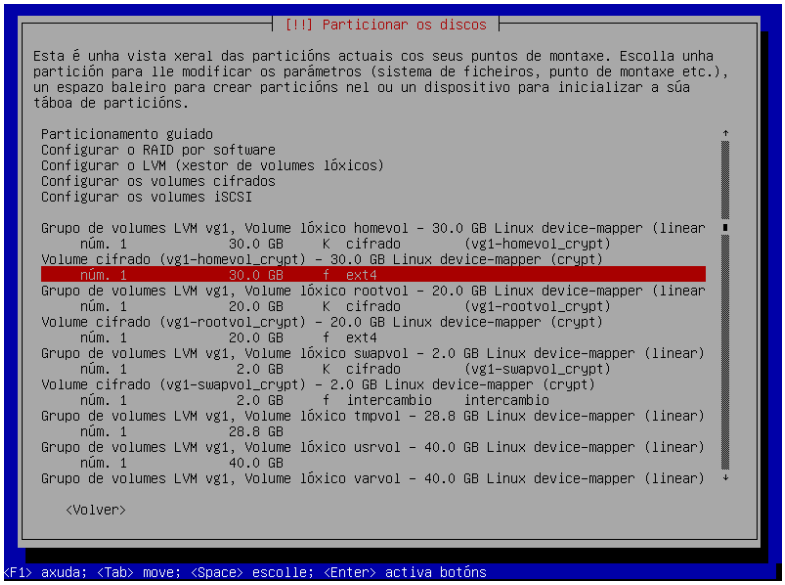
4. Cifrar sistemas de ficheros

- podemos usar algún LV como "volumen físico para cifrado"
- permite cifrar la información: contraseña para acceder a la misma





5. Asignar sistemas de ficheros a los volúmenes (cifrados o no)



Configuración del gestor de arranque

Debemos configurar GRUB para evitar que sea modificado el menu de arranque

- debemos usar una contraseña para limitar:
  - la modificación de los parámetros iniciales
  - el acceso a determinadas imágenes
  - el acceso a opciones avanzadas

2.4. Verificación de la instalación

- Las últimas distribuciones de Linux soportan la mayoría del hardware actual.
- Hay soporte Linux para múltiples arquitecturas: Intel, Alpha, MIPS, PowerPC, SPARC, etc.
- En el proceso de instalación se configura automáticamente casi todo el hardware
- Más información en Linux Hardware Compatibility HOWTO (anticuado) o páginas relacionadas

Verificación del hardware

Para verificar los dispositivos PCI de nuestro sistema se puede usar `lspci`

- `lspci`: lista dispositivos PCI; algunas opciones (para más opciones `man lspci`):
  - `-v`: salida descriptiva
  - `-vv`: salida más descriptiva
  - `-t`: salida con estructura de árbol
- Ejemplo: sistema con discos IDE, tarjeta VGA y dos tarjetas de red:

```
sarge1:~# lspci
0000:00:00.0 Host bridge: Intel Corp. 440FX - 82441FX PMC [Natoma] (rev 02)
0000:00:01.0 ISA bridge: Intel Corp. 82371SB PIIX3 ISA [Natoma/Triton II]
0000:00:01.1 IDE interface: Intel Corp. 82371SB PIIX3 IDE [Natoma/Triton II]
0000:00:02.0 VGA compatible controller: Cirrus Logic GD 5446
0000:00:03.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8029(AS)
0000:00:04.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8029(AS)
```

- Ejemplo: sistema con PCI Express, discos SATA y varios hubs USB conectados

```
jumilla:~# lspci
0000:00:00.0 Host bridge: Intel Corp. 915G/P/GV Processor to I/O Controller (rev 04)
0000:00:01.0 PCI bridge: Intel Corp. 915G/P/GV PCI Express Root Port (rev 04)
0000:00:02.0 VGA compatible controller: Intel Corp. 82915G Express Chipset Family Graphics
Controller (rev 04)
0000:00:02.1 Display controller: Intel Corp. 82915G Express Chipset Family Graphics
Controller (rev 04)
0000:00:1c.0 PCI bridge: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI Express Port 1
(rev 03)
0000:00:1c.1 PCI bridge: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI Express Port 2
(rev 03)
```

```
0000:00:1d.0 USB Controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #1
(rev 03)
0000:00:1d.1 USB Controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #2
(rev 03)
0000:00:1d.2 USB Controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #3
(rev 03)
0000:00:1d.3 USB Controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI #4
(rev 03)
0000:00:1d.7 USB Controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB2 EHCI
Controller (rev 03)
0000:00:1e.0 PCI bridge: Intel Corp. 82801 PCI Bridge (rev d3)
0000:00:1e.2 Multimedia audio controller: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family)
AC'97 Audio Controller (rev 03)
0000:00:1f.0 ISA bridge: Intel Corp. 82801FB/FR (ICH6/ICH6R) LPC Interface Bridge (rev 03)
0000:00:1f.1 IDE interface: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) IDE Controller
(rev 03)
0000:00:1f.2 IDE interface: Intel Corp. 82801FB/FW (ICH6/ICH6W) SATA Controller (rev 03)
0000:00:1f.3 SMBus: Intel Corp. 82801FB/FBM/FR/FW/FRW (ICH6 Family) SMBus Controller (rev
03)
0000:02:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet
PCI Express (rev 01)
```

- Algunas definiciones:
  - UHCI: *Universal Host Controller Interface*, estándar de Intel para controladores USB (define como el controlador USB habla al ordenador y a su sistema operativo); otro estándar similar es OHCI (*Open Host Controller Interface*), desarrollado por Compaq, Microsoft y National Semiconductor Corp.
  - EHCI: *Enhaced Host Controller Interface*, versión extendida para USB 2
  - ICH6: *Intel I/O Controller Hub 6*: controlador para interfaz con el bus PCI
  - SATA: Serial ATA
  - SMBus: *System Management Bus*, bus sencillo para conectar dispositivos de bajo ancho de banda, usado para gestión de energía (p.e. control de batería en portátiles, sensores de temperatura, etc.)

Otro comando: `lsusb`

- `lsusb`: lista dispositivos USB; algunas opciones (para más opciones `man lsusb`):
  - `-v`: salida descriptiva
  - `-t`: salida con estructura de árbol
- Ejemplo: sistema con teclado, ratón, hubs USB y dos pendrive:

```
jumilla:~# lsusb
Bus 005 Device 019: ID 0c76:0005 JMTek, LLC. USBdisk
Bus 005 Device 015: ID 0424:a700 Standard Microsystems Corp.
Bus 005 Device 001: ID 0000:0000
Bus 004 Device 001: ID 0000:0000
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 009: ID 413c:3010 Dell Computer Corp.
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 011: ID 0ea0:2168 Ours Technology, Inc. Transcend JetFlash 2.0 / Astone USB
Drive
Bus 001 Device 007: ID 413c:2002 Dell Computer Corp.
Bus 001 Device 005: ID 413c:1002 Dell Computer Corp. Keyboard Hub
Bus 001 Device 001: ID 0000:0000
```

Para verificar los recursos usados por el hardware podemos analizar los ficheros `interrupts`, `ioports` y `dma` del directorio `/proc`

- `/proc/interrupts`: muestra el número de interrupciones por IRQ (para x86)
- Ejemplo: sistema con una sola CPU

```
# cat /proc/interrupts
           CPU0
  0:   80448940          XT-PIC  timer
  1:     174412          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  8:          1          XT-PIC  rtc
 10:     410964          XT-PIC  eth0
 12:      60330          XT-PIC  PS/2 Mouse
 14:    1314121          XT-PIC  ide0
 15:    5195422          XT-PIC  ide1
NMI:          0
ERR:          0
```

- la primera columna muestra el número de IRQ, la segunda el numero de interrupciones por IRQ, la tercera el tipo de interrupción y la cuarta el dispositivo localizado en esa IRQ
- Definiciones
  - XT-PIC: *XT-Programmable Interrupt Controller*, controlador de interrupciones de la arquitectura AT
  - rtc: *Real Time Clock*
  - cascade: para conectar dos PICs (8259A y 8259B)
  - eth0: tarjeta Ethernet

- NMI (*Nonmaskable Interrup*), interrupción no-enmascarable

- Ejemplo: sistema con 2 CPUs (o 1 con hyperthreading)

```
# cat /proc/interrupts
           CPU0       CPU1
  0:   15126924          0    IO-APIC-edge  timer
  7:          2          0    IO-APIC-edge  parport0
  8:          0          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 14:     135534          1    IO-APIC-edge  ide0
169:      57807          0   IO-APIC-level  libata
177:     630479          0   IO-APIC-level  eth0
185:    1807688          0   IO-APIC-level  uhci_hcd, ehci_hcd
193:     154227          0   IO-APIC-level  uhci_hcd
201:          0          0   IO-APIC-level  uhci_hcd
209:    2153331          0   IO-APIC-level  uhci_hcd, Intel ICH
NMI:          0          0
ERR:          0
```

- Definiciones

  - IO-APIC (*I/O Advanced Programmable Interrupt Controller*): arquitectura de Intel para manejo de interrupciones en entorno multiprocesador (basado en el chip Intel 82093AA)
  - acpi (*Advanced Configuration and Power Interface*): interfaz estándar para configuración y manejo de energía gestionadas por el sistema operativo

- `/proc/ioports`: lista los puertos de entrada salida usados en el sistema

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
```

```
01f0-01f7 : ide0
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
```

- `/proc/dma`: lista los canales ISA DMA registrados en uso

```
# cat /proc/dma
2: floppy
4: cascade
```

Discos duros

En arquitectura Intel nos vamos a encontrar normalmente con alguno de los siguientes tipos de discos:

1. SCSI
   - usuales en servidores de altas prestaciones (PCs, SPARC, etc.)
   - identificados en Linux como: `/dev/sda`, `/dev/sdb`,...
2. Serial ATA
   - Los más comunes
   - Linux los trata de forma similar a SCSI (`/dev/sda`,...)
     - soportados en el kernel 2.4.27 o superior (controlador libata)
3. IDE o Parallel ATA
   - Practicamente no se usan en la actualidad
   - Identificados en Linux como: `/dev/hda`, `/dev/hdb`, `/dev/hdc` y `/dev/hdd`
     - `hda`, `hdb` controlador IDE primario maestro y esclavo, respectivamente
     - `hdc`, `hdd` controlador IDE secundario maestro y esclavo, respectivamente

- Particiones: en Linux, las particiones en un disco se identifican con un número después del nombre del dispositivo:

- podemos ver las particiones con el comando `fdisk -l` (sólo si superusuario):

```
# fdisk -l
Disco /dev/sda: 250.1 GB, 250059350016 bytes
255 cabezas, 63 sectores/pista, 30401 cilindros, 488397168 sectores en total
Unidades = sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico / físico): 512 bytes / 512 bytes
Tamaño E/S (mínimo/óptimo): 512 bytes / 512 bytes
Identificador del disco: 0x259d4594
Dispositivo Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sda1              63       80324       40131   de  Utilidad Dell
/dev/sda2         4179966   488396799   242108417    5  Extendida
/dev/sda5         4179968    64178175    29999104   83  Linux
/dev/sda6        64180224    68177919     1998848   82  Linux swap / Solaris
/dev/sda8        72179712   488396799   208108544   83  Linux
```

- podemos ver las particiones montadas con el comando `df`:

```
# df
Sist. Fich                  1K-bloques     Usado   Dispoñib Uso% Montado en
/dev/sda5                     29528148  20649776    7378420  74% /
udev                           1908232         4    1908228   1% /dev
tmpfs                           768136      1032     767104   1% /run
none                              5120         8       5112   1% /run/lock
none                           1920332      2756    1917576   1% /run/shm
cgroup                         1920332         0    1920332   0% /sys/fs/cg
/dev/sda8                    204842776 147789824   46647528  77% /home
```

- Algunas opciones (para más opciones `man df`):
  - `-h`: muestra valores más fáciles de leer
  - `-i`: muestra información sobre inodos
  - `-T`: imprime el tipo de sistema de ficheros
  - `-l`: sólo muestra sistemas de ficheros locales

Dispositivos SCSI

Muy usados en sistemas de altas prestaciones (servidores)

- No sólo discos: cintas, CD-ROMs, escáneres, etc.

- Los dispositivos se conectan al bus en cadena (*daisy-chained*), actuando uno de ellos como controlador (interfaz con el host)

Evolución de SCSI

Versión	**Bus**	**Freq.**	**BW**	**Long.**	**N. disp.**
SCSI	8 bits	5 MHz	5 MB/s	6m	8
Fast SCSI	8 bits	10 MHz	10 MB/s	1.5-3m	8
Wide SCSI	16 bits	10 MHz	20 MB/s	1.5-3m	16
Ultra SCSI	8 bits	20 MHz	20 MB/s	1.5-3m	5-8
Ultra Wide SCSI	16 bits	20 MHz	40 MB/s	1.5-3m	5-8
Ultra2 SCSI	8 bits	40 MHz	40 MB/s	12m	8
Ultra2 Wide SCSI	16 bits	40 MHz	80 MB/s	12m	16
Ultra3 SCSI	16 bits	40 MHz DDR	160 MB/s	12m	16
Ultra-320 SCSI	16 bits	80 MHz DDR	320 MB/s	12m	16
Ultra-640 SCSI	16 bits	160 MHZ DDR	640 MB/s	12m	16

- Cada dispositivo en el bus (incluyendo el controlador) se identifica con un número (*SCSI address* o *target number*)
  - de 0 a 7 para bus de 8 bits y de 0 a 15 para bus de 16 bits
  - usualmente, el controlador tiene target 7 (en los dos buses)
- Algunos dispositivos, como RAID, tienen un sólo target y varios dispositivos lógicos:
  - LUN: *logical unit number*, identifica los dispositivos lógicos
  - en discos simples o cintas LUN=0

Ejemplo de configuración SCSI en Linux

Dispositivo	**Target**	**LUN**	**Disp. Linux**
Disco 0	0	-	`/dev/sda`
Disco 1	1	-	`/dev/sdb`
Cinta	5	-	`/dev/st0`
RAID disp. 0	6	0	`/dev/sdc`
RAID disp. 1	6	1	`/dev/sdd`
Controlador	7	-	-

Ejemplo, disco en Solaris:

- partición 6, del disco conectado al controlador 0, con target 9 y LUN 0:
  - `/dev/dsk/c0t9d0s6`

Otras versiones SCSI

- Serial Attached SCSI (SAS): bus serie, mayor velocidad (375-750 MB/s)
- iSCSI: Interner SCSI, permite el uso del protocolo SCSI sobre redes TCP/IP

2.5. Instalación de software

Tenemos, básicamente dos formas de instalar programas en Linux:

- Compilación e instalación desde las fuentes
  - Optimización para nuestro sistema
  - Más compleja
- Instalación desde paquetes precompilados
  - Menos optimización
  - Más sencilla

Instalación desde el código fuente

Pasos:

1. Descarga:
   - Normalmente se distribuye en forma de *tarballs*: ficheros `.tar.Z`, `.tar.gz`, `.tgz`, `.tar.bz2` o `.tbz`
2. Desempaquetado: comando `tar` (*Tape ARchive format*)
   - `tar` - crea y extrae ficheros de un archivo
   - Opciones principales:
     - `-c` o `--create` - Crea un archivo `tar`
     - `-t` o `--list` - Lista el contenido de un archivo
     - `-x` o `--extract` - Extrae los ficheros de un archivo
   - Otras opciones
     - `-f` o `--file` *fich* - Usa el archivo *fich* (por defecto "-" que significa entrada/salida estándar)
     - `-v` o `--verbose` - Lista los ficheros según se van procesando
     - `-z` o `--gzip` - Comprime/descomprime ficheros `gzip`

        - `-j` o `--bzip2` - Comprime/descomprime ficheros `bzip2`
    - Ejemplos
        - Muestra el contenido de un tar.gz
          `$ tar tzvf archivo.tar.gz | more`
        - Extrae un fichero tar.bz2
          `$ tar xjvf archivo.tar.bz2`
        - Crea un tar.gz con los ficheros del directorio dir
          `$ tar czvf archivo.tar.gz dir/`

3. Leer el fichero INSTALL, INSTALAR o similar

4. Configuración

    - El código fuente desarrollado con ayuda de las herramientas GNU (*autoconf*) contienen un script `configure`, que se encarga de:
        - chequear el entorno de compilación
        - chequear las librerías necesarias
        - generar los `Makefiles` que nos permitirán compilar el código
    - Ejecución
        - `./configure` <*opciones*>
    - Para ver opciones: `./configure --help`
    - Ejemplo:
        - `./configure --prefix=/opt`
        - instala el programa en `/opt` en vez de en el directorio por defecto (normalmente `/usr/local`)

5. Compilación

    - El proceso de configuración genera ficheros `makefile` o `Makefile` en los directorios del código fuente
        - indican reglas (*rules*) que especifican como ejecutar ciertas tareas (*targets*) sobre el código: compilar, enlazar, crear páginas de manual, instalar
    - Funcionamiento:
        - `make` (ejecuta el *target* por defecto, normalmente todo, menos instalar)
        - `make all` (si no existe el *target* por defecto)
        - `make clean` (borra ficheros objetos, ejecutables, etc)

6. Instalación
    - Si la compilación terminó con éxito, simplemente
        - `make install` (instala el programa ejecutable, librerías, páginas de manual)

Librerías compartidas Dos tipos de ejecutables:

1. Enlazados estáticamente (*statically linked*): son "completos"
2. Enlazados dinámicamente (*dynamically linked*): para ejecutarse necesitan librerías instaladas en el sistema
    - ocupan menos que los estáticos
    - librerías compartidas por varios programas

Para ver las librerías que un ejecutable necesita usar `ldd`:

```
# ldd /bin/ln
      libc.so.6 => /lib/tls/libc.so.6 (0xb7ea3000)
 /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb7fea000)
```

El cargador dinámico Se encarga de cargar los ejecutables con las librerías que necesitan

- en linux es `ld-linux.so.2`
- los directorios con librerías son (además de `/lib/` y `/usr/lib/`) los indicados en el fichero `/etc/ld.so.conf`
    - si modificamos ese fichero, debemos ejecutar el comando `ldconfig`, que regenera, a partir de los directorios indicados en `/etc/ld.so.conf`, el fichero `/etc/ld.so.cache`
    - para ver las librerías compartidas: `ldconfig -p |less`
    - si queremos que el cargador cargue las librerías de un directorio particular, antes de mirar los indicados en `ld.so.conf` usamos la variable de entorno `LD_LIBRARY_PATH`
        - `export LD_LIBRARY_PATH=/usr/lib/old:/opt/lib"`

Gestores de paquetes

En la mayoría de distribuciones Linux, es posible obtener los programas precompilados en formato de paquetes

- Ventajas:
  - Fáciles de instalar y desinstalar
  - Fáciles de actualizar
  - Fácil control de los programas instalados
- Inconvenientes
  - Binarios menos optimizados
  - Problemas de dependencias de paquetes
  - Problemas si la base de datos de paquetes se corrompe

Formatos de paquetes más populares

- Paquetes `DEB` (distribución Debian)
- Paquetes `RPM` (**R**edHat **P**ackage **M**anager, distribuciones Fedora, RedHat, Mandriva, etc.)

Gestión de paquetes en Debian

La distribución Debian incluye un elevado número de paquetes (más de 17.000)
Varias herramientas para el manejo de esos paquetes.

- `dpkg` - herramienta de bajo nivel, para gestionar directamente los paquetes DEB
- `apt-`*xxx* - herramientas APT, permiten gestionar los paquetes, descargándolos de varias fuentes (CDs, ftp, http)
- `dselect` - herramienta de administración de paquetes basada en menús (alto nivel)
- `tasksel` - interfaz para instalación de tareas (grupos de paquetes relacionados)
- `aptitude` - *front-end* de APT para usar en consola
- `synaptic` - *front-end* de APT para usar en entorno gráfico

- `alien` - permite convertir e instalar paquetes de otro tipo, p.e. RPMs

Para más información ver el capítulo Debian package management de la Debian Reference (v2)

dpkg Permite instalar, actualizar o desinstalar paquetes DEB
Los paquetes DEB contienen:

- Los binarios que se van a instalar
- Metadatos, con información sobre el paquete, *scripts* para su configuración, lista de dependencias, etc.

Nombre de los paquetes:

- *paquete_ versión-build_ arquitectura*.deb, donde
  - *paquete* - nombre de la aplicación
  - *versión* - número de versión de la aplicación
  - *build* - número de "compilación" (subversión)
  - *arquitectura* - plataforma para la que está compilado
- Ejemplo:
  - `ethereal_0.10.11-1_i386.deb`

Instalación y eliminación de paquetes con dpkg:

- Instalación de paquetes

  `dpkg --install` *paquete*`.deb`, o
  `dpkg -i` *paquete*`.deb`

  - la instalación chequea la existencia de dependencias, paquetes en conflicto, sobreescritura de ficheros existentes, etc.
  - se puede forzar la instalación usando la opción `--force-`*cosas*, donde *cosas*
    - `conflicts` - permite la instalación de paquetes en conflicto
    - `overwrite` - sobrescribe un fichero de un paquete con otro
    - `overwrite-dir` - sobrescribe el directorio de un paquete con uno nuevo
    - etc.
  - para ver todas las opciones de forzado hacer: `dpkg --force-help`

- Eliminación de paquetes, manteniendo los ficheros de configuración

  `dpkg --remove` *paquete*, o
  `dpkg -r` *paquete*

- Eliminación total de paquetes, eliminando los ficheros de configuración

  `dpkg --purge` *paquete*, o
  `dpkg -P` *paquete*

- Reconfiguración de un paquete ya instalado

  `dpkg-reconfigure` *paquete*

Información sobre los paquetes

- Listar paquetes

  `dpkg --list [`*patrón*`]`, o `dpkg -l [`*patrón*`]`

  - si no se pone *patrón* muestra los paquetes instalados
  - ejemplo

```
# dpkg -l 'telnet*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Nome                  Versión               Descripción
+++-=====================-=====================-======================================
ii  telnet                0.17-29               The telnet client
un  telnet-client        <ningunha>            (non hai ningunha descripción dispoñible)
un  telnet-hurd          <ningunha>            (non hai ningunha descripción dispoñible)
un  telnet-server        <ningunha>            (non hai ningunha descripción dispoñible)
pn  telnet-ssl           <ningunha>            (non hai ningunha descripción dispoñible)
pn  telnetd              <ningunha>            (non hai ningunha descripción dispoñible)
un  telnetd-hurd         <ningunha>            (non hai ningunha descripción dispoñible)
pn  telnetd-ssl          <ningunha>            (non hai ningunha descripción dispoñible)
```

- Las tres primeras columnas representan:
  - **Estado de selección**: indica el estado del paquete para su uso con dselect
    - u, *Unknown* - estado no conocido
    - i, *Install* - paquete seleccionado para instalar (se instala con `dselect install`)
    - r, *Remove* - paquete seleccionado para eliminar (se elimina con `dselect install`)
    - p, *Purge* - paquete seleccionado para purgar (se elimina con `dselect install`)
    - h, *Hold* - paquete retenido (no puede actualizarse)
  - **Estado actual**: indica el estado actual del paquete
    - n, *Not Installed* - paquete no instalado
    - i, *Installed* - paquete instalado en el sistema
    - c, *Config-files* - paquete no instalado, pero ficheros de configuración presentes (p.e. después de un remove)
    - u, *Unpacked* - paquete desempaquetado y listo para instalación
    - f, *Failed-config* - durante la instalación falló la configuración del paquete
    - h, *Half-installed* - paquete a medio instalar debido a algún problema
  - **Condiciones de error**
    - h, *Hold* - paquete retenido, no puede instalarse o eliminarse
    - r, *Reinstallation Required* - necesita reinstalarse
- Información y estado del paquete

- información general

  `dpkg --print-avail` *paquete*, o `dpkg -p` *paquete*

- estado del paquete

  `dpkg --status` *paquete*, o `dpkg -s` *paquete*

- ejemplo

```
# dpkg -s wget
Package: wget
Status: install ok installed
Priority: important
Section: web
Installed-Size: 1428
Maintainer: NoÃšl KÃ¶the <noel@debian.org>
Architecture: i386
Version: 1.10-2
Depends: libc6 (>= 2.3.2.ds1-21), libssl0.9.7
Conflicts: wget-ssl
Conffiles:
 /etc/wgetrc a9465704a21e403be628d38e10b0e128
Description: retrieves files from the web
 Wget is a network utility to retrieve files from the Web
  ....
```

- Ficheros de un paquete

  `dpkg --listfiles` *paquete*, o `dpkg -L` *paquete*

  - ejemplo

```
dpkg -L wget
/.
/etc
/etc/wgetrc
/usr
/usr/bin
/usr/bin/wget
/usr/share
/usr/share/doc
/usr/share/doc/wget
/usr/share/doc/wget/AUTHORS
/usr/share/doc/wget/ChangeLog.README
...
```

- Paquete al que pertenece un fichero

`dpkg --search` *fichero*, o `dpkg -S` *fichero*

- ejemplo

```
# dpkg --search /usr/bin/wget
wget: /usr/bin/wget
```

- Más información: ver ficheros en el directorio `/var/lib/dpkg/`
  - Fichero `/var/lib/dpkg/available`
    - muestra los paquetes disponibles
  - Fichero `/var/lib/dpkg/status`
    - muestra el estado de los paquetes
  - `dpkg` lee estos ficheros para obtener información sobre los paquetes

APT - Advanced Packaging Tools Permite descargar e instalar paquetes desde una fuente local y/o remota
Fuentes de apt: fichero `/etc/apt/sources.list`

```
# See sources.list(5) for more information
deb ftp://ftp.rediris.es/debian/ stable main contrib non-free
deb http://security.debian.org/ stable/updates main contrib non-free
#Para descargar fuentes, a través de apt-get source
deb-src ftp://ftp.rediris.es/debian/ stable main
```

- formato de `sources.list`

  `deb` *uri distribución componente* #Para binarios
  `deb-src` *uri distri. componente* #Para ficheros fuente

- *componente* puede ser:
  - `main` - conjunto principal de paquetes
  - `contrib` - paquetes adicionales
  - `non-free` - paquetes que no son libres

El fichero `sources.list` puede modificarse

- editándolo directamente, o
- a través del comando `apt-setup`

Opciones de configuración de `APT`

- Fichero `/etc/apt/apt.conf`
- Ficheros en el directorio `/etc/apt/apt.conf.d`

Cuando el fichero `sources.list` contiene referencias a más de una distribución (por ejemplo, estable y pruebas), APT asigna una prioridad a cada versión disponible

- es posible seleccionar una distribución ojetivo (*target release*) a la que se le asigna una mayor prioridad:
  - crear un fichero en el directorio `/etc/apt/apt.conf.d`, de nombre, por ejemplo, `99apt-default-release.conf` que contenga la línea

    `APT::Default-Release "`*`distribution`*`";`

    con *`distribution`* igual a *`stable`*, *`testing`* o *`unstable`*
- Si queremos instalar un paquete de una distribución distinta a la por defecto, añadir las líneas necesarias en el `sources.list` y usar `apt-get` con la opción `-t`

  `# apt-get -t `*`distribution`*` install `*`package`*
- podemos usar `apt-cache policy` para ver la política de prioridades configurada
- ver `man apt_preferences` y Debian package management para más detalles

Comando `apt-get`
Comando principal de las herramientas APT Permite descargar, instalar, actualizar o borrar un paquete

1. Actualizar la lista de paquetes

   `apt-get update`
2. Instalar un paquete

   `apt-get install` *nombre_paquete*
3. Actualizar los paquetes

   `apt-get upgrade`

- debe hacerse un `apt-get update` antes de un `apt-get upgrade`

4. Eliminar paquetes

   `apt-get remove` *nombre_paquete*

5. Actualizar la distribución

   `apt-get dist-upgrade`

   - maneja inteligentemente los cambios de dependencias debidos a nuevas versiones de paquetes

6. Eliminar los paquetes descargados

   - Cuando se instala un paquete a través de `apt-get` se guarda una copia en `/var/cache/apt/archives/`

     `apt-get clean` #Elimina todos los paquetes descargados
     `apt-get autoclean` #Elimina sólo los paquetes obsoletos

7. Descargar ficheros fuente

   `apt-get source` *nombre_paquete*

   - con la opción `--compile` compila el paquete después de descargarlo (y genera el `.deb`)

8. Descargar dependencias para compilar un paquete

   `apt-get build-dep` *nombre_paquete*

`apt-get` acepta diversas opciones, por ejemplo:

- `-s` - simula la acción, pero no instala nada
- `-y` - responde `y` a todas las preguntas

para más opciones `man apt-get`

Dependencias entre paquetes
Los paquetes pueden depender unos de otros:

- El paquete A depende (*Depends*) del paquete B si B es absolutamente necesario para usar A
- El paquete A recomienda (*Recommends*) el paquere B si se considera que la mayoría de los usuarios no querrían A sin las funcionalidades que proporciona B
- El paquete A sugiere (*Suggests*) el paquete B si B está relacionado y mejora las funcionalidades de A
- El paquete A está en conflicto (*Conflicts*) con B en el caso de que A no funcionae correctamente si B está instalado

Otras herramientas APT

1. `apt-cache` - permite manipular la caché de paquetes de APT, buscando paquetes o obteniendo información sobre los mismos
   - Ejemplo: buscar el paquete que contiene el firefox

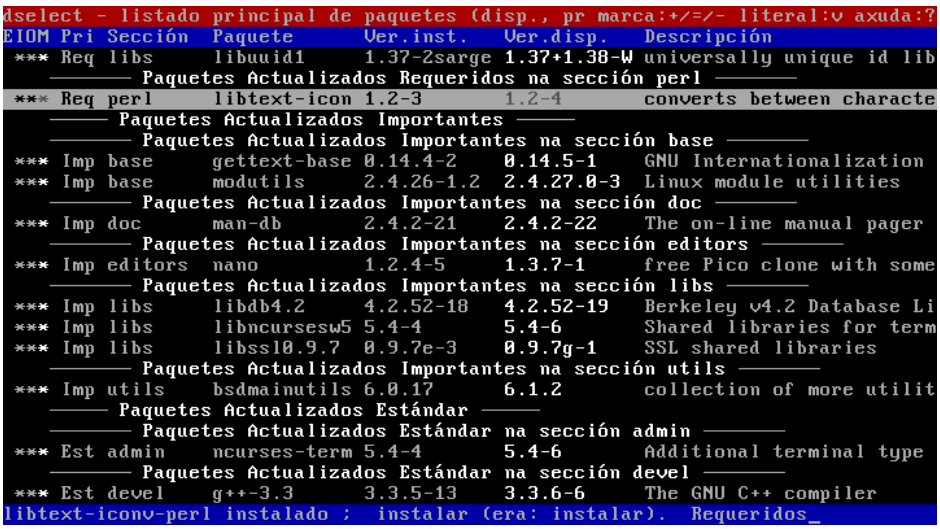
```
# apt-cache search firefox
bookmarkbridge - tool to synchronize bookmarks between browsers
gtkcookie - Editor for cookie files
latex-xft-fonts - Xft-compatible versions of some LaTeX fonts
libflash-mozplugin - GPL Flash (SWF) Library - Mozilla-compatible plugin
mozilla-firefox - lightweight web browser based on Mozilla
mozilla-firefox-dom-inspector - tool for inspecting the DOM of pages in Mozilla Fi
mozilla-firefox-gnome-support - Support for Gnome in Mozilla Firefox
mozilla-firefox-locale-af-za - Mozilla Firefox Afrikaans language/region package
...
```

   - el argumento puede ser una expresión regular

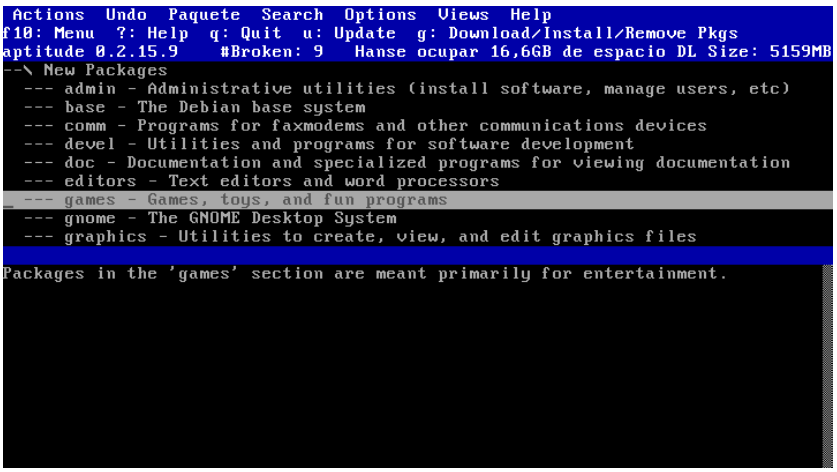
2. `apt-build` - permite descargar, compilar e instalar un paquete a partir de las fuentes

dselect, aptitude, tasksel, synaptic Interfaces del gestor de paquetes
Proporcionan interfaces para consola o gráficas para simplificar el manejo de los paquetes

- Ejemplo de `dselect`



- Ejemplo de `aptitude`



- `aptitude` tiene opciones similares a `apt-get`
  - `aptitude update`: actualiza la lista de paquetes
  - `aptitude search <nombre>`: busca paquetes
  - `aptitude show <nombre_paquete>`: muestra información del paquete
  - `aptitude install <nombre_paquete>`: instala paquetes
  - `aptitude remove <nombre_paquete>`: desinstala paquetes

- `aptitude purge <nombre_paquete>`: desinstala paquetes y sus archivos de configuración
- `aptitude clean`: elimina copias en cache de los ficheros deb descargados
- `aptitude autoclean`: elimina copias en cache de ficheros deb descargados obsoletos
- `aptitude hold <nombre_paquete>`: fuerza a que un paquete permanezca en su versión actual, y no se actualice
- `aptitude safe-upgrade`: actualiza los paquetes instalados, sin eliminar ninguno
- `aptitude full-upgrade`: actualiza los paquetes instalados, eliminando paquetes si es necesario

- `aptitude` podría manejar las dependencias mejor que `apt-get`, pero es menos estable

alien Convierte paquetes entre diferentes formatos
Soporta los formatos Red Hat rpm, Debian deb, Stampede slp, Slackware tgz, y Solaris pkg

- Por defecto, convierte paquetes a formato deb
- Algunas opciones (más opciones, ver página de manual):
  - `--to-rpm` o `-r` - crea un paquete rpm
  - `--to-tgz` o `-t` - crea un paquete tgz
  - `--to-slp` - crea un paquete slp
  - `--to-pkg` o `-p` - crea un paquete pkg
  - `--install` o `-i` - instala el paquete despues de crearlo
- Ejemplo:

```
# alien wget-1.9.1-5.i386.rpm
wget_1.9.1-6_i386.deb generated
```

Paquetes RPM: RedHat Package Manager

Instala software a partir de ficheros `.rpm`, manteniendo control de las dependencias

- Fichero RPM:

  nombre-versión-<release>.<arquitectura>.rpm

  - Ejemplos:

    `wget-1.9.1-5.i386.rpm`
    `xsnow-1.42-14.src.rpm`

  - Muchos RPMs pueden obtenerse en rpmfind.net
  - Otro repositorio: atrpms.net

- El sistema RPM mantiene una base de datos con información de los paquetes instalados en el sistema

  - si hay problemas, intentar reconstruirla con:

    `rpm --rebuilddb`

Comando `rpm` El comando `rpm` permite:

- instalar, actualizar y eliminar paquetes
- validar la integridad de un paquete
- consultar la base de datos RPM para obtener información
- construir un paquete binario a partir de las fuentes

Para más información sobre rpm:

- rpm Home Page
- RPM HOWTO
- Fedora Project Developer's Guide: Building RPM Packages

1. Instalar un rpm

   `rpm -i` *fichero.rpm*, o `rpm --install` *fichero.rpm*
   `rpm -ivh` *fichero.rpm* # Da una salida más visual

   - El proceso de instalación falla si detecta dependencias o si tiene que sobreescribir algún fichero existente
   - Opciones

- `--force` - Fuerza a `rpm` para que sobreescriba paquetes o ficheros existentes
- `--nodeps` - No chequea dependencias

- Ejemplo

```
# rpm -ivh xsnow-1.42-14.i386.rpm
Preparing...    ########################################
xsnow           ########################################
```

- En vez del fichero rpm puede usarse el URL del fichero, p.e.:

```
# rpm -ivh ftp://rpmfind.net/linux/fedora/core/2/i386/os/Fedora/RPMS/xs
Preparing...    ########################################
xsnow           ########################################
```

2. Borrar un paquete instalado

   `rpm -e` *paquete*, o `rpm --erase` *paquete*

   - Ejemplo

     ```
     # rpm -e xsnow
     ```

3. Actualizar un paquete

   `rpm -U` *fichero.rpm*, o `rpm --upgrade` *fichero.rpm*
   `rpm -F` *fichero.rpm*, o `rpm --freshen` *fichero.rpm*

   - si hay una versión del paquete instalada, la borra e instala la nueva; si no hay ninguna versión, simplemente instala la nueva
   - la opción `F` sólo actualiza si hay una versión más antigua instalada

4. Validar la integridad de un paquete

   `rpm --checksig` *fichero.rpm*, o `rpm -K` *fichero.rpm*

   - es necesario importar las claves públicas con el que se firmó el paquete
   - Ejemplo (buscamos la clave pública en el repositorio, por ejemplo, para paquetes de Fedora):

     ```
     # rpm -K xsnow-1.42-14.i386.rpm
     xsnow-1.42-14.i386.rpm: (SHA1) DSA sha1 md5 (GPG) NOT OK (MISSING KEYS:
     # rpm --import 4F2A6FD2.txt
     # rpm -K xsnow-1.42-14.i386.rpm
     xsnow-1.42-14.i386.rpm: (sha1) dsa sha1 md5 gpg OK
     ```

5. Información del paquete: uso `rpm -q` o `rpm --query`

   `rpm -q` [opciones] *paquete* # si el paquete está instalado
   `rpm -qp` [opciones] *fichero.rpm* # si el paquete **no** está instalado
   `rpm -qa` # muestra todos los paquetes instalados

   - Ejemplo:

     ```
     # rpm -qa |grep kernel
     kernel-smp-2.4.20-31.9
     kernel-pcmcia-cs-3.1.31-13
     kernel-source-2.4.20-31.9
     ```

   - Opciones de información

     a) Listar ficheros de un paquete
     `rpm -ql` *paquete*

     b) Determinar a que paquete pertenece un fichero
     `rpm -qf` *fichero*
     - Ejemplo:

       ```
       # rpm -qf /usr/bin/a2ps
       a2ps-4.13b-28
       ```

     c) Información del paquete
     `rpm -qi` *paquete*

     d) Requisitos previos (paquetes de los que depende)
     `rpm -qR` *paquete*

6. Verificar si algún fichero del paquete ha cambiado

   `rpm -V` *paquete*, o `rpm --verify` *paquete*

   - Ejemplo:

     ```
     # rpm -V pam
     S.5....T  c /etc/pam.d/system-auth
     ```

   - el fichero de configuración `system-auth` ha cambiado en tamaño (`S`), suma MD5 (`5`) y fecha de modificación (`T`)
   - otros indicadores:
     - `U`/`G` - cambio en el usuario/grupo del fichero
     - `M` - cambio en permisos o tipo de fichero

7. Compilar un paquete fuente

   - El paquete fuente se puede instalar con `rpm -i`

     ```
     rpm -ivh xsnow-1.42-14.src.rpm
     ```

   - los ficheros fuente se descomprimen en

     ```
     /usr/src/.../SOURCES/
     ```

   - `/usr/src/.../SPECS/` contiene el fichero de `spec`, que indica como compilar el código
   - el paquete se compila con el comando `rpmbuild`, generándose el RPM para instalar:

     ```
     rpmbuild -ba fichero.spec
     ```

   - podemos hacerlo directamente desde el rpm

     ```
     rpmbuild --rebuild fichero.rpm
     ```

YUM - Yellowdog Updater Modified Gestor de paquetes para sistemas basados en RPM

- Funcionalidad similar a APT
- Herramienta estándar en Fedora
- `up2date` puede usar YUM para actualizar el sistema

Algunas opciones:

- Busqueda de paquetes

  ```
  yum search nombre
  ```

- Instalación

  ```
  yum install nombre
  ```

- Actualización

  ```
  yum update nombre
  ```

Ficheros de configuración:

- Configuración base: `/etc/yum.conf`
- Repositorios: `/etc/yum.repos.d/`

APT con RPMs Es posible usar APT con RPMs:

- Instalar el paquete `apt`
- Configurar las fuentes:
  - pueden añadirse más fuentes en el `/etc/apt/sources.list`
  - Ejemplo (para Fedora):

```
# ATrpms for Fedora Core 1
# Possible sections: at-stable, at-good, at-testing, at-bleeding
rpm http://apt.atrpms.net fedora/2/en/i386 at-testing
#rpm-src http://apt.atrpms.net fedora/2/en/i386 at-testing
```

3. Uso de la línea de comandos

Veremos conceptos básicos para usar nuestro sistema desde la línea de comandos

3.1. El interprete de comandos (shell)

El shell se inicia cuando accedemos a nuestra cuenta
Proporciona:

- un interprete de comandos
- un entorno de programación

El shell nos permite ejecutar:

- Comandos externos, por ejemplo: `ls`, `cat`, `mkdir`, etc.
  - son programas ajenos al shell
  - cuando se lanzan inician un nuevo proceso
  - se buscan en los directorios indicados en la variable PATH
- Comandos internos (*builtin commands*), por ejemplo: `cd`, `bg`, `alias`, `eval`, `exec`, `pwd`, etc.
  - se ejecutan en el mismo proceso del shell, sin lanzar un nuevo proceso
  - ver el manual del shell para más información (o para el shell bash: `man bash-builtins`, o el comando `help`)

- En bash: para saber si un comando es externo o interno usar el comando interno `type`:

```
$ type cd
cd is a shell builtin
$ type cat
cat is /bin/cat
```

Principales shells:

- **sh** o *Bourne **sh**ell*: shell por defecto en las primeras versiones de UNIX
- **bash** o ***B**ourne **a**gain **sh**ell*: versión mejorada de **sh**
  - desarrollada en el proyecto GNU
  - es el shell por defecto en Linux
- **csh** o ***C sh**ell*: desarrollada para UNIX BSD, su sintaxis se basa en la del lenguaje C
- **tcsh** o ***T**urbo **C sh**ell*: versión mejorada de **csh**
- **ksh** o ***K**orn **sh**ell*: basado en Bourne shell con características del C shell

Otros shells:

- **ash** o ***A**lmquist **sh**ell*: clon ligero de **sh** (en Linux Debian, **dash** o *Debian ash*)
- **fish** o **Friendly Interactive Shell**: shell amigable para sistemas UNIX
- **zsh** o ***Z sh**ell*: extensión mejorada de **sh**, incorporando características de otros shells como bash, ksh y tcsh
- **rc shell**: shell del sistema operativo *Plan 9* de los Bell Labs., (existe un *porting* de **rc** para UNIX)
- **es shell**: reimplementación del **rc shell** para sistemas UNIX; basado en programación funcional

Para ver las shells conocidas ver el fichero `/etc/shells`

- El shell por defecto para cada usuario se especifica en el fichero `/etc/passwd`
- Para ver la shell por defecto: `echo $SHELL`

- Para ver la shell actual: `ps | grep $$`
- Para cambiar de shell, ejecutar el comando correspondiente, p.e. `/bin/csh`
  - para volver al shell anterior `exit` o `Ctrl-D`
- Para cambiar la shell por defecto: `chsh`

3.2. La línea de comandos

El shell nos permite enviar comandos al sistema
Los comandos usualmente constan de 4 componentes.

- el nombre del comando (con la ruta absoluta, si no está en el PATH)
- opciones, usualmente precedidas por uno o dos guiones (-)
- argumentos (o parámetros)

Ejemplo: comando `ls` (lista ficheros y directorios)

`$ ls` (lista los archivos del directorio actual)
`$ ls -l` (lista los archivos en formato detallado)
`$ ls -la /tmp` (lista todos los archivos del directorio `/tmp`)

En algunos casos no es necesario usar guión con las opciones, ya que el comando espera por lo menos una:

```
$ tar cf miarchivo.tar arch1 arch2 arch3
```

Pueden indicarse varios argumentos, separados por espacios en blanco

```
$ echo hola amigo
```

- Comando $\longrightarrow$ `echo`
- Argumento 1 $\longrightarrow$ `hola`
- Argumento 2 $\longrightarrow$ `amigo`

Varios espacios en blanco se interpretan como uno solo

```
$ echo hola            amigo
```

Para que interprete todos los espacios usar comillas simples o dobles

```
$ echo 'hola            amigo'
```

- Comando $\longrightarrow$ `echo`
- Argumento 1 $\longrightarrow$ `hola            amigo`

3.3. Comandos básicos

- Busqueda de información: `man`, `info`, `help`, `whatis`, `apropos`
  - Proporcionan información sobre otros comandos
  - Más detalles en: www.ac.usc.es/docencia/ASRI/Tema_3html/node1.html
- Ficheros y directorios
  - `cp`, `mv`, `rm` - copia, mueve y borra ficheros
  - `cd`, `mkdir`, `rmdir` - accede, crea y borra directorios
- Manejo de ficheros de texto
  - `cat`, `more`/`less` - muestra el contenido de un fichero (`more` o `less` lo hacen página a página)
  - `vi`, `nano`, `emacs` - potentes editores de consola (una explicación de `vi` en www.ac.usc.es/docencia/ASRI/Tema_3html/node19.html)
- Otros comandos básicos
  - `su`, `sudo` - permiten ejecutar comandos cambiando los permisos del usuario, o como administrador
  - `alias` - Permiten crear alias de comandos complejos (para eliminarlos `unalias`)

    ```
    $ alias l='ls -la'
    ```

  - `history` - muestra una lista con los últimos comandos ejecutados y permite reejecutarlos
- Manejo del historial de comandos

Comando	**Descripción**
`<up-arrow>`/`<down-arrow>`	Comando anterior/posterior
`!!`	Último comando ejecutado
`!`*n*	*n*-ésimo comando del historial
`!-`*n*	*n* comandos hacia atrás
`!`*cadena*	Último comando ejecutado que empieza por cadena
`!?`*cadena*	Último comando ejecutado que contiene cadena
`^`*cadena1*`^`*cadena2*	Ejecuta el último comando cambiando *cadena1* por *cadena2*
`Ctrl-r`	Busca hacia atrás en el historial
`fc`	Permite ver, editar y reejecutar comandos del historial

3.4. Variables de shell

Uso de variables:

- control del entorno (*environment control*)
- programación shell

Dos tipos

- variables locales: visibles sólo desde el shell actual
- variables globales o de entorno: visibles en todos los shells

El comando `set` permite ver las variables definidas en nuestra shell

- El nombre de las variables debe:
  - empezar por una letra o _
  - seguida por cero o mas letras, números o _ (sin espacios en blanco)

Uso de las variables

- Asignar un valor: *nombre_variable=valor*

```
$ un_numero=15
$ nombre="Pepe Pota"
```

- Acceder a las variables: *${nombre_variable}* o *$nombre_variable*

```
$ echo $nombre
Pepe Pota
```

- Número de caracteres de una variable

```
$ echo ${#un_numero}
2
```

- Eliminar una variable: *unset nombre_variable*

```
$ unset nombre
$ echo ${nombre}mo
mo
```

- Variables de solo lectura: *readonly nombre_variable*

```
$ readonly nombre
$ unset nombre
bash: unset: nombre: cannot unset: readonly variable
```

Variables de entorno

Cada shell se ejecuta en un *entorno* (*environment*)

- el entorno de ejecución especifica aspectos del funcionamiento del shell
- esto se consigue a través de la definición de variables de entorno (o variables globales)
- algunas variables son:

Nombre	Propósito
`HOME`	directorio base del usuario
`SHELL`	shell por defecto
`USERNAME`	el nombre de usuario
`PWD`	el directorio actual
`PATH`	el *path* para los ejecutables
`MANPATH`	el *path* para las páginas de manual
`PS1/PS2`	*prompts* primario y secundario
`LANG`	aspectos de localización geográfica e idioma
`LC_*`	aspectos particulares de loc. geográfica e idioma

- Para definir una nueva variable de entorno: `export`

```
$ nombre="Pepe Pota"          # Define una variable de shell
$ echo $nombre                # Usa la variable en el shell
Pepe Pota                     # padre
$ export nombre               # Exporta la variable
$ bash                        # Inicia un nuevo shell
$ echo Mi nombre es $nombre # Intenta usar la variable
Mi nombre es Pepe Pota        # del shell padre
$
```

- La variable exportada (variable de entorno) es visible en el shell hijo
  - el shell hijo crea una copia local de la variable y la usa
  - las modificaciones de esa copia no afectan al shell padre
- Para ver las variables de entorno definidas usar `env` o `printenv`

Más detalles sobre las variables del shell en

www.ac.usc.es/docencia/ASRI/Tema_3html/node11.html

3.5. Expansiones del shell

La sustitución de una variable por su valor se conoce como *expansión de parámetros*

```
$ A=Pepe
$ echo $A
Pepe
```

Otras expansiones

- Expansión de nombres de ficheros (*globbing*)
- Expansión de comandos
- Expansión de llaves
- Expansión de la tilde
- Expansión aritmética

Para más detalles sobre la expansión del shell mirar el manual de bash, sección `EXPANSION`

Expansión de nombres de ficheros

Los *comodines* (*wildcards*) permiten especificar múltiples ficheros al mismo tiempo:

`$ ls -l *html` # Lista los ficheros del directorio actual con terminación html

- también se conoce como *expansión de la shell* o *globbing*
- podemos ver como se hace la expansión poniendo `set -x` o `set -o xtrace`
  - `set +x` para no ver detalles
- podemos desactivar la expansión con `set -f` o `set -o noglob`

Lista de comodines

Carácter	Corresponde a
*	0 o más caracteres
?	1 carácter
[ ]	uno de los caracteres entre corchetes
[!] o [^]	cualquier carácter que no esté entre corchetes

Los ficheros "ocultos" (que empiezan por `.`) no se expanden

- debemos poner el `.` de forma explícita

Nota importante: en **bash** el comportamiento de los rangos depende de la configuración de nuestro sistema, en particular, de la definición de la variable `LC_COLLATE`

- si `LC_COLLATE=C`, `[L-N]` implica `LMN` y `[l-n]` implica `lmn`
- en otro caso (p.e. si `LC_COLLATE=.es_ES.UTF-8"` o `"gl_ES@euro"`) entonces `[L-N]` implica `LmMnN` y `[l-n]` implica `lLmMn`

Para referirnos a mayúsculas o minúsculas podemos usar los siguientes patrones:

- `[[:lower:]]`: corresponde a un carácter en minúsculas
- `[[:upper:]]`: corresponde a un carácter en minúsculas
- `[[:alpha:]]`: corresponde a un carácter alfabético
- `[[:digit:]]`: corresponde a un número

Para más detalles: `man 7 glob`

Expansión de comandos

Permite que la salida de un comando reemplace el propio comando
Formato:

`$(comando)` o `` `comando` ``

Ejemplos:

```
$ echo date
date
$ echo `date`
Xov Xul 21 13:09:39 CEST 2005
$ echo líneas en fichero=$(wc -l fichero)
# wc -l cuenta el número de líneas en el fichero; el comando se
ejecuta y su salida se pasa al echo
```

Expansión de llaves

Permite generar strings arbitrarios

- no tiene para nada en cuenta los ficheros existentes en el directorio actual

```
$ echo a{d,c,b}e
ade ace abe
```

Expansión de la tilde

Expande la tilde como directorio HOME del usuario indicado

- si no se indica usuario, usa el usuario actual

```
cd ~              # Accedemos al nuestro HOME
cd ~root          # Accedemos al HOME de root
ls ~pepe/cosas/   # Vemos el contenido del directorio
                  cosas de pepe
```

Expansión aritmética

Permite evaluar expresiones aritméticas enteras

- se usa `$(( expresión ))` o `$[ expresión ]`
- `expresión` tiene una sintaxis similar a la del lenguaje C
  - permite operadores como `++`, `+=`, `&&`,...
- También se puede usar `let`

```
$ let numero=(numero+1)/2  #usar " si se dejan espacios en blanco
```

- Ejemplos:

```
$ echo $(((4+11)/3))
5
$ numero=15
$ echo $((numero+3))
18
$ echo $numero
15
$ echo $((numero+=4))
```

```
19
$ echo $numero
19
$ numero=$(((numero+1)/2))
$ echo $numero
10
```

Eliminación del significado especial

bash permite eliminar el significado de los caracteres especiales, usando comillas simples, dobles o \

Carácter	Acción
'	el shell ignora todos los caracteres especiales contenidos entre un par de comillas simples
"	el shell ignora todos los caracteres especiales entre comillas dobles excepto $, ` y \
\	el shell ignora el carácter especial que sigue a \

Ejemplos:

```
ls "/usr/bin/a*"
echo '$PATH'
echo "$PATH"
echo I\'m Pepe
```

3.6. Redirección de la entrada/salida

Es posible cambiar la fuente de la entrada o el destino de la salida de los comandos

- toda la E/S se hace a través de ficheros
- cada proceso tiene asociados 3 ficheros para la E/S

Nombre	Descriptor de fichero	Destino por defecto
entrada estándar (*stdin*)	0	teclado
salida estándar (*stdout*)	1	pantalla
error estándar (*stderr*)	2	pantalla

- por defecto, un proceso toma su entrada de la entrada estándar, envía su salida a la salida estándar y los mensajes de error a la salida de error estándar

Ejemplo

```
$ ls /bin/bash /kaka
ls: /kaka: Non hai tal ficheiro ou directorio # Error
/bin/bash           # Salida estándar
$
```

Para cambiar la entrada/salida se usan los siguientes caracteres:

Carácter	**Resultado**
`comando < fichero`	Toma la entrada de `fichero`
`comando > fichero`	Envía la salida de `comando` a `fichero`; sobreescribe cualquier cosa de `fichero`
`comando 2> fichero`	Envía la salida de error de `comando` a `fichero` (el 2 puede ser reemplazado por otro descriptor de fichero)
`comando >> fichero`	Añade la salida de `comando` al final de `fichero`
`comando << etiqueta`	Toma la entrada para `comando` de las siguientes lineas, hasta una línea que tiene sólo `etiqueta`
`comando 2>&1`	Envía la salida de error a la salida estándar (el 1 y el 2 pueden ser reemplazado por otro descriptor de fichero, p.e. `1>&2`)
`comando &> fichero`	Envía la salida estándar y de error a `fichero`; equivale a `comando > fichero 2>&1`
`comando1 | comando2`	pasa la salida de `comando1` a la entrada de `comando2` (*pipe*)

Ejemplos

- `ls -l > lista.ficheros`
  Crea el fichero `lista.ficheros` conteniendo la salida de `ls -l`
- `ls -l /etc >> lista.ficheros`
  Añade a `lista.ficheros` el contenido del directorio `/etc`
- `cat < lista.ficheros | more`
  Muestra el contenido de `lista.ficheros` página a página (equivale a `more lista.ficheros`)
- `ls /kaka 2> /dev/null`
  Envía los mensajes de error al dispositivo nulo (a la *basura*)
- `> kk`
  Crea el fichero `kk` vacío

- `cat > entrada`
  Lee información del teclado, hasta que se teclea `Ctrl-D`; copia todo al fichero entrada

- `cat << END > entrada`
  Lee información del teclado, hasta que se introduce una línea con `END`; copia todo al fichero `entrada`

- `ls -l /bin/bash /kaka > salida 2> error`
  Redirige la salida estándar al fichero `salida` y la salida de error al fichero `error`

- `ls -l /bin/bash /kaka > salida.y.error 2>&1`
  Redirige la salida estándar y de error al fichero `salida.y.error`; el orden es importante:

  ```
  ls -l /bin/bash /kaka 2>&1 > salida.y.error
  ```

  no funciona, por qué?

- `ls -l /bin/bash /kaka &> salida.y.error`
  Igual que el anterior

- `cat /etc/passwd > /dev/tty2`
  Muestra el contenido de `/etc/passwd` en el terminal `tty2`

  - usar el comando `tty` para ver el nombre del terminal en el que estamos

Comandos útiles con pipes y redirecciones

1. `tee`

   - copia la entrada estándar a la salida estándar y también al fichero indicado como argumento:
     - `ls -l | tee lista.ficheros | less`
       Muestra la salida de `ls -l` página a página y la almacena en `lista.ficheros`
   - Opciones:
     - `-a`: no sobreescribe el fichero, añade al final

2. `xargs`

   - permite pasar un elevado número de argumentos a otros comandos

- lee la entrada estándar, y ejecuta el comando uno o más veces, tomando como argumentos la entrada estándar (ignorando líneas en blanco)
- Ejemplos:

  ```
  $ locate README | xargs cat | fmt -60 >\
  /home/pepe/readmes
  ```
  `locate` encuentra los ficheros `README`; mediante `xargs` los ficheros se envían a `cat` que muestra su contenido; este se formatea a 60 caracteres por fila con `fmt` y se envía al fichero `readmes`

  ```
  $ locate README | xargs -i cp {} /tmp/
  ```
  copia los `README` en el directorio `/tmp`; la opción `-i` permite que {} sea reemplazado por los nombres de los ficheros

3. `exec`
   - ejecuta un programa reemplazando el shell actual con el programa (es decir, al programa se le asigna el PID del shell, dejando el shell de existir)

     ```
     $ echo $$ #$$ indica el PID del shell actual
     4946
     $ exec sleep 20
     ```
     En otro terminal, ejecutamos
     ```
     $ ps a | grep 4946
     4946 pts/13   Ss+   0:00 sleep 20
     ```
   - si no se especifica el programa, exec puede usarse para redireccionar las entradas y salidas
     - Redirecciona la salida estándar a el fichero `/tmp/salida`

       ```
       $ exec > /tmp/salida
       ```
     - Redirecciona el fichero `/tmp/entrada` como entrada estándar

       ```
       $ exec < /tmp/entrada
       ```

3.7. Orden de evaluación

Desde que introducimos un comando hasta que se ejecuta, el shell ejecuta los siguientes pasos, y en el siguiente orden:

1. Redirección E/S

2. Sustitución (expansión) de variables: reemplaza cada variable por su valor

3. Sustitución (expansión) de nombres de ficheros: sustituye los comodines por los nombres de ficheros

Si no se tiene en cuenta ese orden, pueden aparecer problemas:

```
$ star=\*
$ ls -d $star
cuatro  dos  tres  uno
$ pipe=\|
$ cat uno $pipe more
cat: |: Non hai tal ficheiro ou directorio
cat: more: Non hai tal ficheiro ou directorio
```

Comando `eval`

Evalúa la línea de comandos 2 veces:

- la primera hace todas las substituciones
- la segunda ejecuta el comando

Ejemplo:

```
$ pipe=\|
$ eval cat uno $pipe more
Este es el fichero uno
...
$
```

- En la primera pasada reemplaza `$pipe` por |
- En la segunda ejecuta el comando `cat uno | more`

3.8. Ficheros de inicialización de bash

Cuando se inicia bash se leen automáticamente distintos ficheros de inicialización

- En estos ficheros el usuario define variables de entorno, alias, el prompt, el path, etc.
- Los ficheros que se leen dependen de la forma de invocar bash

Formas de invocar bash:

1. Invocado como un *login shell* interactivo
   - cuando entramos en el sistema con login y password, usamos `su -`, o iniciamos bash con la opción `--login`
   - cuando se inicia, se leen los siguientes ficheros:
     a) `/etc/profile`
     b) el primero que exista de : `~/.bash_profile`, `~/.bash_login` o `~/.profile`
   - al dejar el shell se lee `~/.bash_logout`
2. Invocado como un *non-login shell* interactivo
   - cuando lo iniciamos sin opciones (bash), abrimos una nueva ventana de comandos (entramos sin login ni password), o usamos `su`
   - se leen los ficheros:
     a) `/etc/bash.bashrc`
     b) `~/.bashrc`[2]
   - al salir no se ejecuta nada
3. Invocado como un shell no interactivo
   - por ejemplo, cuando se lanza un script
   - en un shell no interactivo, la variable `$PS1` no está disponible
   - se lee el fichero definido en la variable `BASH_ENV`

4. Programación de scripts de administración

Un administrador de sistemas debe crear scripts para realizar tareas complejas

- La mayoría de los ficheros de configuración de Unix son ficheros ASCII
- Disponemos de potentes herramientas para manejar estos ficheros

Veremos

[2]Usualmente, desde `.bash_profile` se invoca al `bashrc` de la siguiente forma:
`if [ -f ~/.bashrc ]; then . ~/.bashrc; fi`

- Programación de scripts con bash
- Herramientas de manejo de ficheros de texto usando expresiones regulares
- Programación en Python
- Introducción a Perl y Ruby

4.1. Programación Shell-Script

Bash (y otros *shells*) permiten programar *scripts*:

Script o programa *shell*: fichero de texto conteniendo comandos externos e internos, que se ejecutan línea por línea

- El programa puede contener, además de comandos
  1. variables
  2. constructores lógicos (`if...then`, `AND`, `OR`, etc.) y lazos (`while`, `for`, etc.)
  3. funciones
  4. comentarios

Para saber más:

- *Advanced Bash-Scripting Guide*, Mendel Cooper, Última revisión Mayo 2005, www.tldp.org/guides.html
- *The Deep, Dark Secrets of Bash*, Ben Okopnik, Linux Gazette, okopnik.freeshell.org/articles/4.html
- *Introduction to Shell Scripting*, Ben Okopnik, okopnik.freeshell.org/writings.html

Más detalles en:

www.ac.usc.es/docencia/ASRI/Tema_3html/node34.html

Ejecución de un script

Los scripts deben empezar por el *número mágico* `#!` seguido del programa a usar para interpretar el script:

- `#!/bin/bash` script de bash
- `#!/bin/sh` script de shell
- `#!/usr/bin/perl` script de perl

Las forma usuales de ejecutar un script es:

- darle permiso de ejecución al fichero y ejecutarlo como un comando:

```
$ chmod +x helloworld
./helloworld
```

- ejecutar una shell poniendo como argumento el nombre del script (sólo necesita permiso de lectura)

```
$ bash helloworld
```

- ejecutarlo en la shell actual

```
$ . helloworld
```

  o bien:

```
$ source helloworld
```

Paso de parámetros

Es posible pasar parámetros a un scripts: los parámetros se recogen en las variables `$1` a `$9`

Variable	Uso
`$0`	el nombre del script
`$1` a `$9`	parámetros del 1 al 9
`${10}`, `${11}`,...	parámetros por encima del 10
`$#`	número de parámetros
`$*`, `$@`	todos los parámetros

Ejemplo:

```
$ cat parms1.sh
#!/bin/bash
VAL=$((${1:-0} + ${2:-0} + ${3:-0}))
echo $VAL
$ bash parms1.sh 2 3 5
10
$ bash parms1.sh 2 3
5
```

El comando `shift` desplaza los parámetros hacia la izquierda el número de posiciones indicado:

```
$ cat parms2.sh
#!/bin/bash
echo $#
echo $*
echo "$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11}"
shift 9
echo $1 $2 $3
echo $#
echo $*
$ bash parms2.sh a b c d e f g h i j k l
12
a b c d e f g h i j k l
a b c d e f g h i j k
j k l
3
j k l
```

Entrada/salida

Es posible leer desde la entrada estándar o desde fichero usando `read` y redirecciones:

```
#!/bin/bash
echo -n "Introduce algo: "
read x
echo "Has escrito $x"
echo -n "Escribe 2 palabras: "
read x y
echo "Primera palabra $x; Segunda palabra $y"
```

Si queremos leer o escribir a un fichero utilizamos redirecciones:

```
echo $X > fichero
read X < fichero
```

Este último caso lee la primera línea de `fichero` y la guarda en la variable `X`

- Si queremos leer un fichero línea a línea podemos usar `while`:

```
#!/bin/bash
# FILE: linelist
# Usar: linelist filein fileout
# Lee el fichero pasado en filein y
# lo salva en fileout con las lineas numeradas
count=0
while read BUFFER
do
   count=$((++count))
   echo "$count $BUFFER"» $2
done < $1
```

  - el fichero de entrada se va leyendo línea a línea y almacenando en BUFFER
  - `count` cuenta las líneas que se van leyendo

- El uso de lazos para leer ficheros es bastante ineficiente
  - deberían evitarse (por ejemplo, usar `cat fichero`)

Ejemplo de lectura de fichero

```
#!/bin/bash
# Usa $IFS para dividir la línea que se está leyendo
# por defecto, la separación es "espacio"
echo "Lista de todos los usuarios:"
OIFS=$IFS # Salva el valor de IFS
IFS=: # /etc/passwd usa ":"para separar los campos
cat /etc/passwd |
while read name passwd uid gid fullname ignore
do
  echo "$name ($fullname)"
done
IFS=$OIFS # Recupera el $IFS original
```

- El fichero `/etc/passwd` se lee línea a línea
  - para cada línea, sus campos se almacenan en las variables que siguen a `read`
  - la separación entre campos la determina la variable `$IFS` (por defecto, espacio en blanco)

Redirecciones

Las redirecciones y pipes pueden usarse en otras estructuras de control

Ejemplo: lee las 2 primeras líneas de un fichero

```
if true
then
  read x
  read y
fi < fichero1
```

Ejemplo: lee líneas de teclado y guardalas en un fichero temporal convirtiendo minúsculas en mayúsculas

```
#/bin/bash
read buf
while [ "$buf" ]
do
  echo $buf
  read buf
done | tr 'a-z' 'A-Z' > tmp.$$
```

Tests

Los comandos que se ejecutan en un shell tienen un *código* de salida, que se almacena en la variable `$?`

- si `$?` es 0 el comando terminó bien
- si `$?` es > 0 el comando terminó mal

Ejemplo:

```
$ ls /bin/ls
/bin/ls
$ echo $?
```

```
0
$ ls /bin/ll
ls: /bin/ll: Non hai tal ficheiro ou directorio
$ echo $?
1
```

Podemos chequear la salida de dos comandos mediante los operadores `&&` (AND) y `||` (OR)

- estos operadores actúan en cortocircuito:

  comando1 `&&` *comando2*
  comando2 sólo se ejecuta si *comando1* acaba bien
  comando1 `||` *comando2*
  comando2 sólo se ejecuta si *comando1* falla

- comandos `true` y `false`: devuelven 0 y 1, respectivamente

Ejemplo con `&&`:

```
$ ls /bin/ls && ls /bin/ll
/bin/ls
ls: /bin/ll: Non hai tal ficheiro ou directorio
$ echo $?
1
$ ls /bin/ll && ls /bin/ls
ls: /bin/ll: Non hai tal ficheiro ou directorio
$ echo $?
1
```

Ejemplo con `||`:

```
$ ls /bin/ls || ls /bin/ll
/bin/ls
$ echo $?
0
$ ls /bin/ll || ls /bin/ls
ls: /bin/ll: Non hai tal ficheiro ou directorio
/bin/ls
$ echo $?
0
```

Estructura `if...then...else`

Podemos usar el estado de salida de uno o varios comandos para tomar decisiones:

```
if comando1
then
  ejecuta otros comandos
elif comando2
then
  ejecuta otros comandos
else
  ejecuta otros comandos
fi
```

- debe respetarse la colocación de los `then`, `else` y `fi`
  - también puede escribirse `if` *comando1* `; then`
- el `elif` y el `else` son opcionales, no así el `fi`

Ejemplo:

```
$ cat if.sh
#!/bin/bash
if (ls /bin/ls && ls /bin/ll) >/dev/null 2>&1
then
  echo "Encontrados ls y ll"
else
  echo "Falta uno de los ficheros"
fi
$ bash if.sh
Falta uno de los ficheros
```

Comando `test`

Notar que `if` sólo chequea el código de salida de un comando, no puede usarse para comparar valores: para eso se usa el comando `test`
El comando test permite:

- chequear la longitud de un string
- comparar dos strings o dos números

- chequear el tipo de un fichero
- chequear los permisos de un fichero
- combinar condiciones juntas

`test` puede usarse de dos formas:

`test` *expresión*

o bien

`[` *expresión* `]`[3]

Si la expresión es correcta `test` devuelve un código de salida 0, si es falsa, devuelve 1:

- este código puede usarse para tomar decisiones:

```
if [ "$1" = "hola" ]
then
  echo "Hola a ti también"
else
  echo "No te digo hola"
fi
if [ $2 ]
then
  echo "El segundo parámetro es $2"
else
  echo "No hay segundo parámetro"
fi
```

- en el segundo `if` la expresión es correcta si `$2` tiene algún valor; falsa si la variable no está definida o contiene null (`""`)

Expresiones

Existen expresiones para chequear strings, números o ficheros

[3] Notar los espacios en blanco entre los [ ] y *expresión*

Chequeo de strings

Expresión	Verdadero sí
string	el string es no nulo ("")
-z *string*	la longitud del string es 0
-n *string*	la longitud del string no es 0
string1 = *string2*	los strings son iguales
string1 != *string2*	los strings son distintos

Chequeo de enteros

Expresión	Verdadero sí
int1 -eq *int2*	los enteros son iguales
int1 -ne *int2*	los enteros son distintos
int1 -gt *int2*	*int1* mayor que *int2*
int1 -ge *int2*	*int1* mayor o igual que *int2*
int1 -lt *int2*	*int1* menor que *int2*
int1 -le *int2*	*int1* menor o igual que *int2*

Chequeo de ficheros

Expresión	Verdadero sí
-e *file*	*file* existe
-r *file*	*file* existe y es legible
-w *file*	*file* existe y se puede escribir
-x *file*	*file* existe y es ejecutable
-f *file*	*file* existe y es de tipo regular
-d *file*	*file* existe y es un directorio
-c *file*	*file* existe y es un dispositivo de caracteres
-b *file*	*file* existe y es un dispositivo de bloques
-p *file*	*file* existe y es un pipe
-S *file*	*file* existe y es un socket
-L *file*	*file* existe y es un enlace simbólico
-u *file*	*file* existe y es *setuid*
-g *file*	*file* existe y es *setgid*
-k *file*	*file* existe y tiene activo el *sticky bit*
-s *file*	*file* existe y tiene tamaño mayor que 0

Operadores lógicos con `test`

Expresión	Propósito
`!`	invierte el resultado de una expresión
`-a`	operador AND
`-o`	operador OR
`( expr )`	agrupación de expresiones; los paréntesis tienen un significado especial para el shell, por lo que hay que *escaparlos*

Ejemplos:

```
$ test -f /bin/ls -a -f /bin/ll ; echo $?
1
$ test -c /dev/null ; echo $?
0
$ [ -s /dev/null ] ; echo $?
1
$ [ ! -w /etc/passwd ] && echo "No puedo escribir"
No puedo escribir
$ [ $$ -gt 0 -a \( $$ -lt 5000 -o -w file \) ]
```

Comando de test extendido A partir de la versión 2.02 de Bash se introduce el *extended test command*: `[[ expr ]]`

- permite realizar comparaciones de un modo similar al de lenguajes estándar:
  - permite usar los operadores && y || para unir expresiones
  - no necesita *escapar* los paréntesis

Ejemplos:

```
$ [[ -f /bin/ls && -f /bin/ll ]] ; echo $?
1
$ [[ $$ -gt 0 && ($$ -lt 5000 || -w file) ]]
```

Control de flujo

Además del `if` bash permite otras estructuras de control de flujo: `case`, `for`, `while` y `until`

Estructura `case`

```
case valor in
  patrón_1)
    comandos si value = patrón_1
    comandos si value = patrón_1 ;;
  patrón_2)
    comandos si value = patrón_2 ;;
  *)
    comandos por defecto ;;
esac
```

- si *valor* no coincide con ningún patrón se ejecutan los comandos después del `*)`
  - esta entrada es opcional
- *patrón* puede incluir comodines y usar el símbolo | como operador OR

Ejemplo:

```
#!/bin/bash
echo -n "Respuesta:" read RESPUESTA
case $RESPUESTA in
  S* | s*)
    RESPUESTA="SI";;
  N* | n*)
    RESPUESTA="NO ";;
  *)
    RESPUESTA="PUEDE";;
esac
echo $RESPUESTA
```

Lazos `for`

```
for var in lista
do
  comandos
done
```

- *var* toma los valores de la lista
  - puede usarse *globbing* para recorrer los ficheros

Ejemplo: recorrer una lista

```
LISTA="10 9 8 7 6 5 4 3 2 1"
for var in $LISTA
do
  echo $var
done
```

Ejemplo: recorrer los ficheros `*.bak` de un directorio

```
dir="/var/tmp"
for file in $dir/*.bak
do
  rm -f $file
done
```

Sintaxis alternativa, similar a la de C

```
LIMIT=10
for ((a=1, b=LIMIT; a <= LIMIT; a++, b--))
do
  echo "$a-$b"
done
```

Bucle `while`

```
while comando
do
  comandos
done
```

- se ejecuta mientras que el código de salida de `comando` sea cierto

Ejemplo:

```
while [ $1 ]
do
  echo $1
  shift
done
```

Bucle `until`

```
until comando
do
  comandos
done
```

- se ejecuta hasta que el código de salida de `comando` sea hace cierto

Ejemplo:

```
until [ "$1" = ""]
do
  echo $1
  shift
done
```

`break` y `continue` Permiten salir de un lazo (`break`) o saltar a la siguiente iteración (`continue`)

- `break` permite especificar el número de lazos de los que queremos salir (`break` *n*)

Ejemplo con `break`:

```
# Imprime el contenido de los ficheros hasta que
# encuentra una línea en blanco
for file in $*
do
  while read buf
  do
    if [ -z "$buf"]
    then
      break 2
    fi
    echo $buf
  done <$file
done
```

Ejemplo con `continue`:

```
# Muestra un fichero pero no las líneas de más
# de 80 caracteres
```

```
while read buf
do
  cuenta=`echo $buf | wc -c`
  if [ $cuenta -gt 80 ]
  then
    continue
  fi
  echo $buf
done <$1
```

Funciones

Podemos definir funciones en un script de shell:

```
funcion() {
  comandos
}
```

y para llamarla:

```
funcion p1 p2 p3
```

Siempre tenemos que definir la función antes de llamarla:

```
#!/bin/bash
# Definición de funciones
funcion1() {
  comandos
}
funcion2() {
  comandos
}
# Programa principal
funcion1 p1 p2 p3
```

Paso de parámetros La función referencia los parámetros pasados por posición, es decir, `$1`, `$2`, ..., y `$*` para la lista completa:

```
$ cat funcion1.sh
#!/bin/bash
funcion1()
{
  echo "Parámetros pasados a la función: $*"
```

```
  echo "Parámetro 1: $1"
  echo "Parámetro 2: $2"
}
# Programa principal
funcion1 "hola" "que tal estás" adios
$
$ bash funcion1.sh
Parámetros pasados a la función: hola que tal estás adios
Parámetro 1: hola
Parámetro 2: que tal estás
```

`return` Después de llamar a una función, `$?` tiene el código se salida del último comando ejecutado:

- podemos ponerlo de forma explícita usando `return`

```
#!/bin/bash
funcion2() {
  if [ -f /bin/ls -a -f /bin/ln ]; then
    return 0
  else
    return 1
  fi
}
# Programa principal
if funcion2; then
  echo "Los dos ficheros existen"
else
  echo "Falta uno de los ficheros - adiós"
  exit 1
fi
```

Otros comandos

`wait` Permite esperar a que un proceso lanzado en *background* termine

```
sort $largefile > $newfile &
ejecuta comandos
wait
usa $newfile
```

Si lanzamos varios procesos en background podemos usar `$!`

- `$!` devuelve el `PID` del último proceso lanzado

```
sort $largefile1 > $newfile1 &
SortPID1=$!
sort $largefile2 > $newfile2 &
SortPID2=$!
ejecuta comandos
wait $SortPID1
usa $newfile1
wait $SortPID2
usa $newfile2
```

`trap` Permite *atrapar* las señales del sistema operativo

- permite hacer que el programa termine limpiamente (p.e. borrando ficheros temporales, etc.) aún en el evento de un error

```
$ cat trap.sh
#!/bin/bash
cachado() {
  echo "Me has matado!!!"
  kill -15 $$
}
trap "cachado" 2 3
while true; do
  true
done
$ bash trap.sh
(Ctrl-C)
Me has matado!!!
Terminado
```

Las señales más comunes para usar con `trap` son:

Señal	Significado
0	salida del shell (por cualquier razón, incluido fin de fichero)
1	colgar
2	interrupción (Ctrl-C)
3	quit
9	kill (no puede ser parada ni ignorada)
15	terminate; señal por defecto generada por kill

`exit` Finaliza el script

- se le puede dar un argumento numérico que toma como estado de salida, p.e. `exit 0` si el script acaba bien y `exit 1` en caso contrario
- si no se usa `exit`, el estado de salida del script es el del último comando ejecutado

Referencias indirectas

Permiten definir variables cuyo contenido es el nombre de otra variable:

```
a=letra
letra=z
# Referencia directa
echo "a = $a"  # a = letra
# Referencia indirecta
eval a=\$$a
echo "Ahora a = $a"  # Ahora a = z
```

Las versiones de bash a partir de la 2 permiten una forma más simple para las referencias indirectas:

```
a=letra
letra=z
# Referencia directa
echo "a = $a"  # a = letra
# Referencia indirecta
echo "Ahora a = ${!a}"  # Ahora a = z
```

Otro ejemplo con `eval`

```
$ cat dni.sh
#!/bin/bash
dniPepe=23456789
dniPaco=98765431
echo -n "Nombre: "; read nombre
eval echo "DNI = \$dni${nombre}"
$ bash dni.sh
Nombre: Pepe
DNI = 23456789
```

Optimización de scripts

El shell no es especialmente eficiente a la hora de ejecutar trabajos pesados

- Ejemplo: script que cuenta las líneas de un fichero:

```
$ cat cuentalineas1.sh
#!/bin/bash
count=0
while read line
do
  count=$(expr $count + 1)
done < $1
echo "El fichero $1 tiene $count líneas"
```

  - si medimos el tiempo que tarda

```
$ time bash cuentalineas1.sh Quijote.txt
El fichero Quijote.txt tiene 36855 líneas
real 0m59.757s
user 0m17.868s
sys  0m41.462s
```

- Podemos mejorarlo si usamos aritmética de shell en vez de el comando `expr`

```
$ cat cuentalineas2.sh
#!/bin/bash
count=0
while read line
do
  count=$(($count+1))
done < $1
echo "El fichero $1 tiene $count líneas"
```

  - el tiempo ahora

```
$ time bash cuentalineas2.sh Quijote.txt
El fichero Quijote.txt tiene 36855 líneas
real 0m1.014s
user 0m0.887s
sys  0m0.108s
```

- Y todavía mejor:

```
$ cat cuentalineas3.sh
#!/bin/bash
count=$(wc -l $1 | cut -d " " -f 1)
echo "El fichero $1 tiene $count líneas"
$
$ time bash cuentalineas3.sh Quijote.txt
El fichero Quijote.txt tiene 36855 líneas
real 0m0.096s
user 0m0.005s
sys  0m0.009s
```

- Conclusiones
  - Intenta reducir el número de procesos creados al ejecutar el script, por ejemplo, usando las funciones aritméticas del shell
  - Siempre que sea posible, intenta usar comandos del shell (`wc`, `tr`, `grep`, `sed`, etc.) en vez de lazos

Depuración

Para depurar un script de shell podemos usar la opción `-x` o `-o xtrace` de bash:

- muestra en la salida estándar trazas de cada comando y sus argumentos, después de que el comando se haya expandido pero antes de que se sea ejecutado

```
$ bash -x cuentalineas3.sh Quijote.txt
++ wc -l Quijote.txt
++ cut -d ' ' -f 1
+ count=36855
+ echo 'El fichero Quijote.txt tiene 36855 líneas'
El fichero Quijote.txt tiene 36855 líneas
```

Es posible depurar sólo parte de un script:

- poner `set -x` o `set -xv` al inicio del trozo a depurar
- `set +x` o `set +xv` para cancelar

```
$ cat cuentalineas3.sh
#!/bin/bash
set -x
count=$(wc -l $1 | cut -d " "-f 1)
set +x
echo "El fichero $1 tiene $count líneas"
$
$ bash cuentalineas3.sh Quijote.txt
++ wc -l Quijote.txt
++ cut -d ' '-f 1
+ count=36855
+ set +x
El fichero Quijote.txt tiene 36855 líneas
```

5. Manejo de ficheros de texto

Los ficheros de configuración y logs de Unix son, normalmente, ficheros de texto

- se necesitan herramientas para manejar estos ficheros
- Unix dispone de potentes herramientas que hacen uso extensivo de expresiones regulares

5.1. Expresiones regulares

Muchos comandos de procesamiento y búsqueda de texto como `ed`, `grep`, `egrep`, `sed`, `awk` o `vi` usan expresiones regulares:

- permiten reconocer una serie de cadenas de caracteres que obedecen a cierto patrón
- Ejemplos
  - `egrep unix tmp.txt`
    busca en el fichero `tmp.txt` las líneas que contienen la palabra `unix`
  - `egrep '[Uu]nix' tmp.txt`
    busca las líneas que contienen `unix` o `Unix`
  - `egrep 'hel.' tmp.txt`
    busca las líneas que contienen `hel` seguido de cualquier carácter

- `egrep 'ab*c' tmp.txt`
  localiza las cadenas que empiecen por `a`, que continúen con 0 o más `b`, y que sigan con una `c`, por ejemplo: `abbbc` o `aaacb`, pero no `axc` o `cba`
- `egrep 't[^aeiouAEIOU][a-zA-Z]*' tmp.txt`
  localiza las cadenas que empiecen por `t`, seguido de algún carácter no vocálico y 0 o más apariciones de otro carácter

Importante: no debemos confundir las expresiones regulares con la sustitución de nombres de ficheros (*globbing*)

- si ponemos el último ejemplo sin comillas

  ```
  egrep t[^aeiouAEIOU][a-zA-Z]* tmp.txt
  ```

  la shell extiende los comodines y convierte este comando en:

  ```
  egrep tmp.txt tmp.txt
  ```

- para evitar esto, `siempre` usar comillas con las expresiones regulares

Comandos grep y sed

`grep` y `sed` son dos comandos que usan REGEXP

`grep` Busca en ficheros por un patrón determinado

`grep [`*`opciones`*`] `*`patrón`*` [`*`fichero...`*`]`

Opciones:

- `-E` o `egrep`: usa expresiones regulares extendidas
- `-F` o `fgrep`: interpreta los patrones no como expresiones regulares sino como cadenas de caracteres fijas
- `-R` o `rgrep`: lee todos los ficheros bajo cada directorio, recursivamente
- `-i` o `--ignore-case`: busca ignorando diferencias entre mayúsculas y minúsculas
- `-w` o `--word-regexp`: para forzar que la cadena reconocida sea una palabra completa
- `-l` o `--files-with-matches`: no muestra el contenido de la linea encontrada pero si que muestra el fichero que contiene la cadena buscada

- `-n` o `--line-number`: muestra el número de línea dentro del fichero
- `-v` o `--invert-match`: en lugar de sacar la lineas que cumplen la búsqueda sacará las que no cumplen

Si no especificamos fichero, `grep` usa la entrada estándar:

- podemos usarlo para probar las expresiones regulares:

```
$ egrep '[Uu]nix'
unix
unix
Unix
Unix
Linux
```

`sed` (***stream editor***) Editor de flujo; permite realizar transformaciones básicas de un flujo de entrada (un fichero o una entrada desde una tubería)

Formato (para substituciones):

`sed` [*opciones*] 's/*REGEXP*/*reemplazo*/*flag*' [*fichero*]

Algunos comandos:

- `s` substitución
- `d` borrado
- `i\`, `a\`, añade antes/después de la línea afectada
- `c\` reemplaza la línea afectada

Algunas ociones:

- `-e` *comando*: añade *comando*
- `-i` edita el fichero *in-place*
- `-n` suprime la salida

Algunos flags:

- `g`: aplica los cambios globalmente (por defecto, sólo se cambia la primera aparición en cada línea)

- `p` imprime las líneas afectadas, incluso con la opción `-n`.
- *NUMERO*: reemplaza la aparición número *NUMERO*
- `w` *fichero*: escribe las líneas con sustituciones al fichero indicado

Ejemplo: cambia, en el fichero `amigos`, todas las apariciones de `pepe` y `paco` por `Pepe` y `Paco`, respectivamente:

> `$ sed -e 's/pepe/Pepe/g' -e 's/paco/Paco/g' amigos` (también `sed 's/pepe/Pepe/g ; s/paco/Paco/g' amigos`)

Ejemplo: cambia `pepe` por `Pepe`, pero sólo en las líneas que tengan `Potamo`

```
$ sed '/Potamo/s/pepe/Pepe/g' amigos
```

Ejemplo: muestra sólo las lineas que contengan `jaime`

```
$ sed -n '/jaime/p' amigos
```

Ejemplo: borra las lineas que contengan `jaime`

```
$ sed '/jaime/d' amigos
```

Ejemplo: cambia las lineas que contengan `jaime` por otra cosa

```
$ sed '/jaime/c\BORRADO' amigos
```

Ejemplo: inserta una línea, con la palabra 'APARICION', antes de las líneas que contengan `jaime`

```
$ sed '/jaime/i\APARICION' amigos
```

Ejemplo: reemplaza, en cada línea de `fichero`, la quinta ocurrencia de `stop` por `STOP`

```
$ sed 's/stop/STOP/5' fichero
```

Ejemplo: igual que antes pero guarda cada línea reemplazada en el fichero `f2`

```
$ sed 's/stop/STOP/5w f2' fichero
```

Indicación de líneas: podemos especificar las líneas del fichero en las que queremos que se realicen las operaciones:

> `sed '3s/stop/STOP/g'` (reemplaza sólo en la línea 3)
> `sed '3,10s/stop/STOP/g'` (reemplaza de la línea 3 a la 10)
> `sed '3,$s/stop/STOP/g'` (reemplaza de la línea 3 al final)
> `sed '!3s/stop/STOP/g'` (reemplaza en todas las líneas menos la 3)

Operador `&`: se sustituye por el patrón reconocido

Ejemplo: reemplaza `stop` por `<stop>`

```
$ sed '3s/stop/<&>/g' fichero
```

Comandos desde fichero: la opción -f permite leer comandos de `sed` agrupados en un fichero

Ejemplo: reemplazo desde la línea 1 hasta una línea que comience por `END` (o el final, si no hay ninguna)

```
$ cat file.sed
1,/^END/{
  s/[Ll]inux/GNU\/Linux/g
  s/samba/Samba/g
}
$ sed -f file.sed fichero
```

Más información: `sed` es un comando muy complejo con muchas posibilidades

Para saber más:

- mirar la página de `info` de `sed`
- Sed - An Introduction
- Ejemplos con `sed`
- Sed by example, IBM developerworks
- sed & awk, by Dale Dougherty, Arnold Robbins, O'Reilly

o, simplemente, busca *sed tutorial* en *google*

Expresiones regulares básicas

UNIX admite dos tipos de expresiones regulares: básicas y extendidas

- las básicas son las clásicas de UNIX, aunque se consideran obsoletas en POSIX
- aplicaciones como `grep` o `sed` las usan por defecto

- para usar las extendidas:
  - `grep` ⟶ `egrep` o `grep -E`
  - `sed` ⟶ `sed -r`
- las expresiones extendidas proporcionan más potencia

La mayoría de los caracteres son tratados como literales:

- concuerdan (*match*) consigo mismos:
  - `a` concuerda con `a`, `ab` con `ab`, etc.
- la excepción son los metacaracteres:

```
. [ ] ^ $ * ( ) \
```

ER de un sólo carácter

ER	concuerda con
`.`	cualquier carácter
`[ ]`	cualquiera de los caracteres entre corchetes, p.e. `[abc]` concuerda con `a`, `b` o `c`; `[a-z]` concuerda con cualquier letra minúscula
`[^ ]`	cualquier carácter que no esté entre corchetes
`^`	principio de línea
`$`	final de línea
`*`	0 o más ocurrencias de la expresión regular anterior
`\( \)`	permite agrupar ER
`\`	*escapa* un metacarácter

- Dentro de `[ ]` los metacaracteres pierden su significado especial: p.e. `[a.]c` concuerda con `ac` y `.c`
- Para incluir un carácter `]` en una lista colocarlo al principio; para incluir un `^` en cualquier lugar menos al principio; para incluir un `-` al final: p.e. `[a^]c` concuerda con `ac` y `^c`

Ejemplos:

ER	concuerda con
`a..c`	cadena que empiece por `a`, seguida por dos caracteres y `c`: `a00c`, `xaxxcxx`, `aacc`,...
`0[abc]0`	cadenas que tengan un `0` seguido de un carácter `a`, `b`, o `c` y seguido de otro `0`: `0a0`, `00ab0b0`, `bc0c0`,...
`0[^abc]0`	cadenas que tengan un `0` seguido de un carácter distinto a `a`, `b`, o `c` y seguido de otro `0`
`0[a-z]0`	cadenas que tengan un `0` seguido de una letra minúscula, y `0`
`^abc`	líneas que empiecen por `abc`
`abc$`	líneas que terminen por `abc`
`ab*c`	cadenas que empiecen por `a`, que continúen con 0 o más `b`, y una `c`: `abc`, `ac`, `abbc`, `aaccab`,... pero no `cba` o `aaab`
`b[cq]*e`	cadenas que empiecen por `b`, que continúen con 0 o más `c` o `q`, y una `e`: `be`, `bcce`, `bccqqee` o `bqqqce`
`.*`	cualquier cadena
`abc.*`	cualquier cadena que empiece por `abc`
`0\(abc\)*0`	cadenas que tengan un `0` seguido de 0 o más ocurrencias de `abc`, y seguido de otro `0`: `0abc0`, `00`, `0abcabc0`,..., pero no `0ac0` o `0cba0`
`^#.*\.$`	línea que empiece por `#` y termine por `.` (notar que el segundo `.` está *escapado* por la \; la ER `.*` implica 0 o más caracteres cualquiera)

Repetición Podemos repetir una regexp usando `\{ \}`

Constructor	Propósito
`\{`n`\}`	concuerda con exactamente n ocurrencias de la RE previa
`\{`n`,\}`	concuerda con al menos n ocurrencias de la RE previa
`\{`n`, `m`\}`	concuerda con entre n y m ocurrencias de la RE previa

Ejemplos:

- `a\{5\}`: 5 ocurrencias del carácter `a`
- `.\{5,\}`: al menos 5 ocurrencias de cualquier carácter

Expresiones regulares extendidas

Los sistemas UNIX actuales admiten extensiones a las expresiones regulares básicas:

- debemos usar `egrep`, `grep -E`, `sed -r`

ER	concuerda con
`+`	una o más ocurrencias de la RE anterior
`?`	cero o una ocurrencia de la RE anterior

Además, `\( \)` y `\{ \}` se reemplazan por `( )` y `{ }`

- Ejemplos:
  - `ab+c` concuerda con `abc`, `abbc`, pero no con `ac`
  - `ab?c` concuerda con `ac`, `abc`, pero no con `abbc`
- Para usar los caracteres `(`, `)`, `{` o `}` *escaparlos* con `\`

Alternancia El carácter `|` permite alternar entre 2 o más RE

- `(a|b)c` concuerda con `ac` o `bc`

Etiquetado Las RE que se ponen entre `( )` quedan etiquetadas, y podemos hacer referencia a esos elementos mediante $\backslash n$, con n el número de la etiqueta

- Ejemplos:
  - (.)oo\1 concuerda con `moom`, `noon`, pero no con `moon`
  - (.)oo\1-(.)aa\1\2 concuerda con `moom-paamp`

Otros caracteres Además de los ya vistos, pueden usarse otros metacaracteres:

ER	concuerda con
\n, \r, \t	LF, CR y tab (no siempre funcionan)
[:space:]	caracteres en blanco ([\t\n\r\f\v])
[:blank:]	espacio y tabulado
[:alnum:] o \w	caracteres alfánuméricos (letras y números)
[:digit:]	dígitos
[:alpha:]	alfabéticos
[:upper:]	mayúsculas
[:lower:]	minúsculas
[:xdigit:]	dígitos hexadecimales
[:punct:]	signos de puntuación
[:cntrl:]	caracteres de control
[:graph:]	caracteres imprimibles (sin espacio)
[:print:]	caracteres imprimibles (con espacio)
\<, \>	inicio/fin de palabra
\b	posición entre palabras
\B	posición en medio de una palabra

- `[[:upper:]]bc` concuerda con `Abc`, pero no `abc`
- `\babc\b` concuerda con `ab abc df`, pero no con `abcdef`
- `\Babc\B` concuerda con `ababcdf`, pero no con `ab abc df`

Más ejemplos

1. `\w+@\w+\.\w+((\.\w+)*)?` concuerda con direcciones de e-mail
2. `(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[012])-(19|20)[0-9]{2}` concuerda con fechas en el formato *dd-mm-yyyy* (años entre el 1900 y 2099)
3. `[-+]?([0-9]*\.)?[0-9]+([eE][-+]?[0-9]+)?` concuerda con números en punto flotante (con o sin exponente)

Ejemplos de uso con `sed`:

```
$ echo .ªbc1234def"| sed -r "s/[0-9]+/NUMERO/"
abcNUMERODdef
$ echo .ªbc1234def"| sed -r 's/[0-9]+/<&>/'
abc<1234>def
# En el siguiente ejemplo, notar que las ER intentan siempre
reconocer la secuencia más larga posible
$ echo "000x111x222x333"| sed 's/x.*x/<&>/'
000<x111x222x>333
```

```
# Eliminar blancos a principio y al final de línea y sustituir
más de un blanco seguido por uno solo
$ sed -r "s/^␣+// ; s/␣+$// ; s/␣␣+/␣/g"fich
# Pon los 4 primeros caracteres de cada línea al final
de la misma
$ sed -r 's/^(.{4,4})(.*)/\2\1/' fich
# Cambia de minúsculas a mayúsculas la primera letra de
cada palabra
$ sed -r 's/\<./\u&/g'
# Convierte DOS newlines (CR/LF) a formato Unix (LF)
$ sed 's/^M$//'[4]
# también funcionaría
$ sed 's/\r//'
```

Para más información: Regular-expressions.info

5.2. Comandos para el procesamiento de textos

Además de los ya vistos (`vi`, `grep`, `sed`) existen una serie de comandos para manejar ficheros de texto, como `tac`, `rev`, `nl`, `head`, `tail`, `sort`, `uniq`, `expand`, `fmt`, `cut`, `paste`, `tr`, `join`, `split`, `wc`, `od` o `awk`

- también se conocen como *filtros*: obtienen su entrada de la entrada estándar (o un fichero) y envían la salida a la salida estándar:

  ```
  sort < archivo.txt | head -3 > otro_archivo.txt
  ```

- casi todos estos comandos tienen, entre otras opciones, las siguientes dos:
  - `--help` muestra una pequeña ayuda y sal
  - `--version` muestra la versión del comando y sal
- también podemos saber más del comando a través de la página de manual o de `info`

Comandos simples

Existe una serie de comandos simples para realizar operaciones concretas sobre ficheros de texto

[4] Para introducir un carácter de control, como `^M`, tenemos que pulsar primero `Ctrl-V` y luego el carácter, en este caso `Enter`

- Ordena las líneas alfabéticamente: `sort`
- Escribe partes seleccionadas de un fichero a la salida estándar: `cut`
- Une texto de varios ficheros: `paste`
- Formatea párrafos: `fmt`
- Borra y/o reemplaza caracteres: `tr`
- Elimina líneas repetidas: `uniq`
- Combina varios ficheros: `join`
- Divide un fichero en ficheros más pequeños: `split`
- Muestra el principio/final de un fichero: `head`/`tail`
- Muestra el fichero al revés: `tac`, `rev`
- Muestra el número de líneas, palabras y bytes de un fichero: `wc`
- Añade números de línea: `nl`
- Convierte `TABs` en espacios: `expand`
- Muestra un fichero en diferentes formatos: `od`

Comentaremos brevemente cada uno de ellos

sort ordena alfabéticamente líneas de texto y las muestra en la salida estándar
Formato:

`sort` [opciones] *fichero*

Algunas opciones:

- `-b` ignora blancos al principio de línea
- `-f` no distingue mayúsculas/minúsculas
- `-r` orden inverso
- `-m` mezcla ficheros previamente ordenados
- `-n` ordena numéricamente

- `-k` *POS1[, POS2]* ordena según los campos desde *POS1* a *POS2*, o el final si no está *POS2* (el primer campo es 1)

Ejemplos:

```
$ cat nombres.txt
María Pérez
luis Andión
Adriana Gómez
jorge pena
$ sort nombres.txt
Adriana Gómez
María Pérez
jorge pena
luis Andión

$ sort -f nombres.txt
Adriana Gómez
jorge pena
luis Andión
María Pérez
$ sort -f +1 +0 nombres.txt #Obsoleto (no usar)
luis Andión
Adriana Gómez
jorge pena
María Pérez
$ sort -f -k 2,2 nombres.txt
luis Andión
Adriana Gómez
jorge pena
María Pérez
```

cut Escribe partes seleccionadas de un fichero a la salida estándar; puede usarse para seleccionar columnas o campos de un fichero específico
Formato:

`cut` [opciones] *fichero*

Algunas opciones:

- `-b`, `-c`, `-f` corta por bytes, caracteres o campos, respectivamente
- `-d` fija el carácter delimitador entre campos (por defecto, `TAB`)

Ejemplos:

```
$ cat nombres-ord.txt
Luis Andión
Adriana Gómez
Jorge Pena
María Pérez

$ cut -c 1-7 nombres-ord.txt
Luis An
Adriana
Jorge P
María P
$ cut -c 1-5,9-10 nombres-ord.txt
Luis ió
AdriaGó
Jorgena
Maríare
$ cut -d ' ' -f 1 nombres-ord.txt
Luis
Adriana
Jorge
María
```

paste Permite unir texto de varios ficheros, uniendo las líneas de cada uno de los ficheros

Formato:

`paste` [opciones] *fichero1 [fichero2]* ...

Algunas opciones:

- `-s` pega los ficheros secuencialmente, en vez de intercalarlos
- `-d` especifica los carácteres delimitadores en la salida (por defecto, `TAB`)

Ejemplos:

```
$ cat nombres.txt
Luis
Adriana
Jorge
María
```

```
$ cat apellidos.txt
Andión
Gómez
Pena
Pérez
$ paste nombres.txt apellidos.txt
Luis     Andión
Adriana  Gómez
Jorge    Pena
María    Pérez
$ paste -d ' ' nombres.txt apellidos.txt
Luis Andión
Adriana Gómez
Jorge Pena
María Pérez
$ paste -s -d '\t\n' nombres.txt
Luis   Adriana
Jorge  María
```

fmt Formatea cada párrafo, uniendo o separando líneas para que todas tengan el mismo tamaño
Algunas opciones:

- `-`*n* o `-w` *n* pone la anchura de las líneas a *n* (por defecto, 75)
- `-c` conserva la indentación a principio de línea y alinea a la izquierda la segunda línea
- `-s` las líneas pueden dividirse, no unirse
- `-u` uniformiza el espaciado entre palabras

Ejemplo:

```
$ cat quijote.txt
En un lugar de la Mancha, de    cuyo nombre no
quiero acordarme, no ha mucho tiempo
que vivía un
hidalgo    de los de lanza en astillero, adarga
antigua, rocín flaco y galgo corredor.

$ fmt -w 45 -u quijote.txt
En un lugar de la Mancha, de cuyo nombre
```

```
no quiero acordarme, no ha mucho tiempo
que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y
galgo corredor.
```

tr Borra caracteres o reemplaza unos por otros
Formato:

`tr` [opciones] *set1 set2*

Algunas opciones:

- `-d` borra los caracteres especificados en *set1*
- `-s` reemplaza caracteres repetidos por un único carácter

Ejemplos:

```
$ tr 'a-z' 'A-Z' < quijote.txt
EN UN LUGAR DE LA MANCHA, DE CUYO NOMBRE...
$ tr -d ' ' < quijote.txt
EnunlugardelaMancha,decuyonombre...
$ tr au pk < quijote.txt
En kn lkgpr de lp Mpnchp, de ckyo nombre...
$ tr lcu o < quijote.txt | tr -s o
En on ogar de oa Manoha, de oyo nombre
```

uniq Descarta todas (menos una) las líneas idénticas sucesivas en el fichero
Formato:

`uniq` [opciones] *fichero*

Algunas opciones:

- `-d` muestra las líneas duplicadas (sin borrar)
- `-u` muestra sólo las líneas sin duplicación
- `-i` ignora mayúsculas/minúsculas al comparar
- `-c` muestra el número de ocurrencias de cada línea
- `-s` n no compara los n primeros caracteres
- `-f` n no compara los n primeros campos

- `-t` *c* usa el carácter *c* como separador de campos (por defecto, espacio o tabulado)

Ejemplo:

```
$ cat nombres.txt
Julio Lorenzo
Pedro Andión
Celia Fernández
Celia Fernández
Juan Fernández
Enrique Pena
$ uniq nombres.txt
Julio Lorenzo
Pedro Andión
Celia Fernández
Juan Fernández
Enrique Pena
$ uniq -f 1 -c nombres.txt
  1 Julio Lorenzo
  1 Pedro Andión
  3 Celia Fernández
  1 Enrique Pena
```

join Permite combinar dos ficheros usando campos: busca en los ficheros por entradas comunes en el campo y une las líneas; los ficheros deben estar ordenados por el campo de unión

Formato:

`join` [opciones] *fichero1 fichero2*

Algunas opciones:

- `-i` ignora mayúsculas/minúsculas
- `-1` *FIELD* une en el campo *FIELD* (entero positivo) de *fichero1*
- `-2` *FIELD* une en el campo *FIELD* de *fichero2*
- `-j` *FIELD* equivalente a `-1` *FIELD* `-2` *FIELD*
- `-t` *CHAR* usa el carácter *CHAR* como separador de campos
- `-o` *FMT* formatea la salida (*M.N* fichero *M* campo *N*, 0 campo de unión)

- `-v` N en vez de la salida normal, muestra las líneas que no se unen del fichero N
- `-a` N además la salida normal, muestra las líneas que no se unen del fichero N

Ejemplo:

```
$ cat nombres1.txt
Luis Andión
Adriana Gómez
Jorge Pena
María Pérez
$ cat nombres2.txt
Pedro Andión
Celia Fernández
Julio Lorenzo
Enrique Pena
$ join -j 2 nombres1.txt nombres2.txt
Andión Luis Pedro
Pena Jorge Enrique
$ join -j 2 -o 1.1 2.1 0 nombres1.txt nombres2.txt
Luis Pedro Andión
Jorge Enrique Pena
```

split Divide un fichero en ficheros más pequeños; los ficheros más pequeños se nombran a partir del *prefijo* especificado (*prefijo*aa, *prefijo*ab,...)
Formato:

`split` [opciones] *fichero prefijo*

Si no se pone *fichero*, o se pone - se lee la entrada estándar
Algunas opciones:

- `-l` n pone n lineas en cada fichero de salida (por defecto 1000)
- `-b` n pone n bytes en cada fichero de salida
- `-C` n pone en cada fichero de salida tantas líneas completas como sea posible sin sobrepasar n bytes
- `-d` usa números en vez de letras para el nombre de los ficheros de salida

Ejemplo:

```
$ split -l 2 quijote.txt quij
$ ls quij*
quijaa quijab quijac quijote.txt
$ cat quijaa
En un lugar de la Mancha, de cuyo nombre
no quiero acordarme, no ha mucho tiempo
$ cat quijac
galgo corredor.
$ split -l 2 -d quijote.txt quij
$ ls quij*
quij00 quij01 quij02 ...
```

head Muestra el principio de un fichero
Formato:

`head` [opciones] *fichero*

Algunas opciones:

- `-n` N ó `-`N muestra las primeras N líneas
- `-c` N muestra los primeros n bytes
- `-v` le añade una línea de cabecera, con el nombre del fichero

Ejemplo:

```
$ head -n 2 -v quijote.txt
==>quijote.txt <==
En un lugar de la Mancha, de cuyo nombre
no quiero acordarme, no ha mucho tiempo
```

tail Muestra el final de un fichero
Algunas opciones:

- `-n` N ó `-`N muestra las últimas N líneas (por defecto, 10)
- `+`N muestra de la línea N al final
- `-c` N muestra los últimos N bytes
- `-f` hace que `tail` corra en un lazo, añadiendo líneas a medida que el fichero crece (útil para cuando queremos ver como se modifica un fichero)

- `--retry` útil con `-f`; aunque el fichero no exista o sea inaccesible continua intentando hasta que puede abrirlo
- `-v` le añade una línea de cabecera, con el nombre del fichero

Ejemplo:

```
$ tail -n 2 -v quijote.txt
==>quijote.txt <==
astillero, adarga antigua, rocín flaco y
galgo corredor.
```

tac, rev `tac` imprime el fichero de la última a la primera línea (opuesto a `cat`); `rev` invierte las lineas del fichero
Ejemplos:

```
$ tac quijote.txt
galgo corredor.
astillero, adarga antigua, rocín flaco y
que vivía un hidalgo de los de lanza en
no quiero acordarme, no ha mucho tiempo
En un lugar de la Mancha, de cuyo nombre

$ rev quijote.txt
erbmon oyuc ed ,ahcnaM al ed ragul nu nE
opmeit ohcum ah on ,emradroca oreiuq on
ne aznal ed sol ed ogladih nu aíviv euq
y ocalf nícor ,augitna agrada ,orellitsa
.roderroc oglag
```

wc Muestra el número de líneas, palabras y bytes de un fichero
Formato:

`wc` [opciones] *fichero*

Algunas opciones:

- `-l` muestra sólo el número de lineas
- `-w` muestra sólo el número de palabras
- `-c` muestra sólo el número de bytes
- `-L` muestra la longitud de la línea más larga

Ejemplo:

```
$ wc quijote.txt
  5 33 178 quijote.txt
$ wc -l quijote.txt
5 quijote.txt
$ wc -w quijote.txt
33 quijote.txt
$ wc -c quijote.txt
178 quijote.txt
```

nl Añade números de línea; `nl` considera los ficheros separados en *páginas lógicas*, cada una de ellas con una cabecera, cuerpo y pie, cada una de estas secciones se numera de forma independiente, y la numeración se reinicia para cada página; los comienzos de cabecera, cuerpo y pie de cada página se marcan, respectivamente, con `\:\:\:`, `\:\:` y `\:`
Formato:

`nl` [opciones] *fichero*

Algunas opciones:

- `-b`, `-h` o `-f` *ESTILO* indica el estilo de numeración para cuerpo, cabecera o pie, que puede ser:
  - `a`: numera todas las líneas
  - `t`: numerar sólo las líneas no vacías (por defecto para el cuerpo)
  - `p` *REGEXP*: numera sólo las líneas que concuerdan con *REGEXP*
  - `n`: no numera ninguna línea (por defecto para cabecera y pie)
- `-v` n inicia la numeración en n (por defecto, 1)
- `-i` n incrementa los números por n (por defecto, 1)
- `-p` no reinicia la numeración al principio de cada página
- `-s` *STRING* una STRING para separar los números de línea del texto (por defecto ' ')

Ejemplo:

```
$ nl -s 'q ' quijote.txt
  1q En un lugar de la Mancha, de cuyo nombre
  2q no quiero acordarme, no ha mucho tiempo
  3q que vivía un hidalgo de los de lanza en
  4q astillero, adarga antigua, rocín flaco y
  5q galgo corredor.
```

expand Convierte `TABs` en espacios; útil debido a que la representación del `TAB` puede ser diferente en distintos sistemas
Formato:

`expand` [opciones] *fichero* ...

Algunas opciones:

- `-t` *n* reemplaza cada `TAB` por *n* espacios (por defecto, 8)
- `-i` solo reemplaza los `TABs` de principio de línea

Ejemplos:

```
$ cat hola.c
main() {
        for(i=0; i<10;i++)
                printf("Hola mundo!\n");
}
$ expand -t 2 hola.c
main() {
  for(i=0; i<10;i++)
    printf("Hola mundo!\n");
}
```

El comando `unexpand` hace la operación contraria

od Muestra un fichero en octal, hexadecimal o otros formatos; en cada línea muestra (en la primera columna) el *offset*
Formato:

`od` [opciones] *fichero*

Algunas opciones:

- `-t` *TIPO* especifica el formato de la salida (por defecto octal): `o` para octal, `x` para hexadecimal, `d` para decimal, `c` para caracteres ASCII, `a` para caracteres con *nombre...*
- `-A` *TIPO* especifica el formato del offset (por defecto octal): `o`, `x`, `d` como antes, `n` para que no aparezca
- `-w` *BYTES* número de bytes por linea (por defecto 16)

Ejemplo:

```
$ od -t x -A x quijote.txt
000000 75206e45 756c206e 20726167 6c206564
000010 614d2061 6168636e 6564202c 79756320
000020 6f6e206f 6572626d 206f6e0a 65697571
...
```

awk

Lenguaje diseñado para procesar datos basados en texto; el nombre AWK deriva de los apellidos de los autores: Alfred V. Aho, Peter J. Weinberger, y Brian W. Kernighan

- los administradores de sistemas utilizan `awk` para procesar los ficheros de configuración y logs de los sistemas
- estos ficheros, normalmente, se organizan en forma de *tabla* (líneas compuestas por campos)
  - `awk` es ideal para tratar esos ficheros
- sólo veremos algunos de los aspectos más importantes del uso de `awk` para el manejo de ficheros de texto

Funcionamiento básico `awk` lee el fichero que se le pase como entrada (o la entrada estándar) línea a línea, y sobre cada línea ejecuta una serie de operaciones

Ejemplo:

```
# echo -e interpreta "\nçomo un retorno de carro,
# lo que envía 2 líneas al comando awk
$ echo -e "\n"| awk '{ print "Hola mundo!"}'
Hola mundo!
Hola mundo!
```

Formas de ejecutar `awk` Podemos usar `awk` de varias formas:

- En la línea de comandos:

  `awk` *`PROGRAMA fichero_entrada`*

- Escribiendo el programa en un fichero:

  `awk -f` *`FICHERO_PROGRAMA fichero_entrada`*

- Ejecutando el *`FICHERO_PROGRAMA`* como un script:

  poner

  ```
  #!/usr/bin/awk -f
  ```

  al principio de *`FICHERO_PROGRAMA`*

Ejemplos:

```
$ echo '{ print "Hola mundo!"}' > hola.awk
$ echo -e "\n"| awk -f hola.awk
Hola mundo!
Hola mundo!
$ echo '#!/usr/bin/awk -f' > hola.awk
$ echo '{ print "Hola mundo!"}'» hola.awk
$ chmod +x hola.awk
$ echo -e "\n"| ./hola.awk
Hola mundo!
Hola mundo!
```

Estructura de un programa `awk` Un programa `awk` tiene tres secciones:

1. Parte inicial, que se ejecuta sólo una vez, antes de empezar a procesar la entrada:

   `BEGIN {` *`operaciones`* `}`

2. Parte central, con instrucciones que se ejecutan para cada una de las líneas de la entrada; tienen en siguiente formato:

   /`PATRÓN`/ `{` *`operaciones`* `}`

   las *`operaciones`* se realizan sólo sobre las líneas que verifiquen la REGEXP indicada en *`PATRÓN`*

- si ponemos `!/`*PATRÓN*`/` las operaciones se ejecutan en las líneas que no concuerden con el patrón

3. Parte final, se efectúa sólo una vez, después de procesar la entrada:

   `END { `*operaciones*` }`

Manejo de ficheros de texto `awk` divide las líneas de la entrada en campos:

- la separación entre campos la determina la variable `FS` (por defecto, uno a más blancos o `TAB`s
- las variables $1, $2, . . . , $N contienen los valores de los distintos campos
  - $0 contiene la línea completa

Ejemplos:

```
$ ls -ldh * |\
> awk '{print "Fichero ", $8, .°cupa ", $5, "bytes"}'
Fichero proba ocupa 36 bytes
Fichero uy_hist1_nodos.txt ocupa 9,1K bytes
Fichero vimbook-OPL.pdf ocupa 3,7M bytes


$ df -h | sort -rnk 5,5 |\
> awk 'BEGIN { print "Nivel de ocupación"}\
> /^\/dev\/hd/ {print "Partición ",$6,": ",$5}\
> END { print "Terminado"}'
Nivel de ocupación
Partición /home : 87% ocupación
Partición /mnt/hda2 : 51% ocupación
Partición / : 38% ocupación
Terminado

$ # Usando un fichero
$ cat ocupacion.awk
BEGIN {
     print "Nivel de ocupación"
}
/^\/dev\/hd/ {
     print "Partición ",$6,": ", $5
```

```
}
END { print "Terminado" }
$ df -h | sort -rnk 5,5 | awk -f ocupacion.awk
```

Variables predefinidas: `awk` tiene un conjunto de variables predefinidas, como `FS` que nos permite especificar el separador de campos

Esas variables son:

Nombre	Significado
`FS`	Carácter separador entre campos de entrada (por defecto, blanco o tabulado)
`NR`	Número de registros de entrada
`NF`	Número de campos en el registro de entrada
`RS`	Carácter separador entre registros de entrada (por defecto, nueva línea)
`OFS`	Carácter separador entre campos en la salida (por defecto, un espacio en blanco)
`ORS`	Carácter separador entre registros de salida (por defecto, nueva línea)
`FILENAME`	Nombre del fichero abierto

Ejemplo:

```
$ cat usuarios.awk
BEGIN {
   FS = ":"; OFS = "-->"; ORS = "\n============\n";
}
{
   print NR, $1, $5
}
$ awk -f usuarios.awk /etc/passwd
...
37 -->tomas -->Tomás Fernández Pena,,,
============
38 -->caba -->José Carlos Cabaleiro Domínguez,,,
============
...
```

Otras características `awk` es un lenguaje completo:

- permite definir variables de usuario

- permite realizar operaciones aritméticas sobre las variables
- permite utilizar condiciones, lazos, etc.
- permite definir funciones

La sintaxis de `awk` es prácticamente idéntica a la del lenguaje C

- podemos usar `printf` en lugar de `print` (con la sintaxis de C)
- también podemos usar arrays

Ejemplos:

1. Lista el tamaño de los ficheros y el tamaño total

```
$ cat lista-ficheros.awk
BEGIN { total = 0; }
{
   total += $5;
   printf("Fichero %s ocupa %d bytes\n", $8,$5);
}
END {
   printf("Ocupación total = %d bytes\n", total);
}
$ ls -ld * | awk -f lista-ficheros.awk
Fichero ancestros.awk ocupa 370 bytes
Fichero hola.c ocupa 66 bytes
Fichero lista-ficheros.awk ocupa 143 bytes
Ocupación total = 579 bytes
```

2. Muestra una advertencia si el nivel de ocupación de una partición supera un límite

```
$ cat ocupacion2.awk
BEGIN { limite = 85; }
/^\/dev\/hd/ {
   if($5 >limite)
     printf("PELIGRO: el nivel de ocupación de %s es %s\n %",
$6, $5);
}
$ df -ah | tr -d '%' | awk -f ocupacion2.awk
PELIGRO: el nivel de ocupación de /home es 87 %
```

Paso de parámetros: es posible pasar parámetros en la llamada a `awk`
Ejemplo: Indicando el `PID` de un proceso obtiene el `PID` de todos sus ancestros (padres, abuelos, ...)

```
$ cat ancestros.awk
BEGIN { ind=0; }
function padre(p) {
   for(i=0; i <ind; i++)
     if(pid[i] == p) return(ppid[i]);
}
!/PID/ { pid[ind]=$3; ppid[ind]=$4; ind++; }
END {
   do {
     printf("%d --> ", proc); proc = padre(proc);
   } while(proc >= 1);
   printf("\n\n");
}
$ ps axl | awk -f ancestros.awk proc=4258
4258 --> 3326 --> 1 -->
```

Arrays asociativos: `awk` permite el uso de arrays asociativos, es decir, que pueden tener como índice una cadena de caracteres
Ejemplo

```
$ cat usuarios2.awk
BEGIN { FS = ":"}
{ nombre[$1] = $5; }
END {
  for(;;){
    printf("Nombre de usuario: ");
    getline user < ";
    if( user ==  )
      break;
    printf("<%s>: %s\n", user, nombre[user]);
  }
}
$ awk -f usuarios2.awk /etc/passwd
Nombre de usuario: tomas
<tomas>: Tomás Fernández Pena,,,
Nombre de usuario:
```

Funciones predefinidas En `awk` existen una serie de funciones predefinidas

- `getline`: lee la siguiente línea de la entrada, pudiendo asignarla a una variable
  - `getline` *variable* `<` *fichero*
    lee una línea de fichero y la mete en *variable*
  - `getline` *variable* `< "`
    lee una línea de la entrada estándar y la mete en *variable*
  - `"`*comando*`" | getline`
    coge la salida de comando y la pone en la variable $0, descomponiéndola en campos ($1, $2, ...)

  Ejemplo:

  ```
  $ awk 'BEGIN{ "date"| getline; print $4 }'
  15:16:59
  ```

- `system`: ejecuta un comando del sistema operativo; en caso de éxito retorna 0, y en caso de error retornará un valor distinto de cero
  Ejemplo:

  ```
  $ awk 'BEGIN {\
  > if (system("ls")!=0)\
  > printf (.Error de ejecución"); }'
  ```

6. Programación en Python

Además de la programación con bash, sed y awk, existen otros lenguajes adecuados para la creación de scripts de administración

Perl: lenguaje de propósito general originalmente desarrollado para la manipulación de textos

Python: alternativa a Perl, más limpio y elegante

Ruby: combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos

Los tres son lenguajes de propósito general

- Permiten programar aplicaciones de muy diversos tipos
- Veremos solo una introducción a sus principales características, centrándonos principalmente en Python

Un buen administrador de sistemas debería dominar al menos uno de ellos

6.1. Introducción a Python

Bash es complejo y el código Perl puede resultar demasiado "ofuscado"

- Python es una buena alternativa a los lenguajes de script tradicionales

Principales características

- Soporte de diversos paradigmas: imperativo, orientado a objetos y funcional
- Sistema de tipos dinámico y gestión automática de memoria
- Énfasis en la legibilidad
- Uso de identación para delimitar bloques de código
- Gran librería con módulos para múltiples tareas

Ejemplo sencillo:

```
#!/usr/bin/env python
# coding: utf-8
# Abre el fichero sólo lectura
try:
  f = open("/etc/passwd","r")
except IOError:
  print "No puedo abrir /etc/passwd"
else:
  # Lee las líneas en una lista
  lista = f.readlines()
  # Recorre e imprime la lista
  for l in lista:
    print l, # La coma elimina \n
  f.close()
```

6.2. Tipos de datos en Python

Además de los tipos "estándar" Python proporciona:

1. Listas: mutables, pueden contener tipos mezclados

   ```
   frutas=["naranjas", "uvas", 123, "limones", "uvas"]
   frutas.append("peras")
   frutas.remove(123)
   ```

```
frutas.remove("uvas") # [naranjas,limones,uvas,peras]
frutas[2:2] = ["fresas", "pomelos"] # inserta en pos 2
print frutas       # naranjas,limones,fresas,pomelos,uvas,peras
print len(frutas)  # 6
print frutas[0:3]  # naranjas, limones, fresas
print frutas[-3]   # pomelos
print frutas[1:-3] # limones, fresas
frutas.pop()       # Elimina el último elemento
del frutas[2:4]    # Elimina los elementos 2 y 3
frutas.sort()      # Ordena
print frutas       # [limones,naranjas,uvas]
a=list("hola")     # a=["h","o","l","a"]
"o" in a           # True
```

Las listas pueden enlazarse

```
a = [[0,1],[2,3]]
print a[1][1]   # 3
a.append([4,5])
print a[2][0]   # 4
del a[1]
print a         # [0,1], [4,5]
```

`range`: función built-in que genera listas de valores en secuencia:

```
l = range(5)         # l = [0, 1, 2, 3, 4]
l = range(2, 5)      # l = [2, 3, 4]
l = range(2, 10, 3)  # l = [2, 5, 8]
l = range(5, -5, -2) # l = [5, 3, 1, -1, -3]
a = sum(range(1,4))  # a = 6
```

Las listas son objetos mutables (string, enteros, etc. no)

```
a = 1      # nuevo objeto entero (1) al que a referencia
b = a      # a y b referencias al mismo objeto entero (1)
a += 5     # se crea un nuevo objeto 6 (1+5)
print b    # 1, b sigue referenciando al objeto 1
a = [1, 2] # nuevo objeto lista
b = a      # a y b referencias al mismo objeto lista
a[0] += 5  # se modifica el objeto (mutable)
print b    # [6, 2] b es modificado
```

Copia de listas

```
a = [1, 2] # nuevo objeto lista
b = a[:]   # a y b referencias objetos diferentes
a[0] += 5  # se modifica el objeto (mutable)
print b    # [1, 2] b no se modificado
c=list(a)  # otra forma
```

2. Tuplas: listas inmutables

```
y=("enero","febrero","marzo","abril", "mayo", "junio",\
"julio","agosto","septiembre","octubre","noviembre",\
"diciembre")  # Paréntesis opcionales
print y[3]    # Abril
```

3. Conjuntos (Sets): sin elementos duplicados

```
cesta=["naranjas", "uvas", "limones", "uvas"]
frutas=set(cesta)
print frutas  # naranjas,uvas,limones
a = set("abracadabra")
b = set("alacazam")
print a      # "a", "r", "b", "c", "d"
print a-b    # "r", "b", "d"
print a | b  # "a", "c", "b", "d", "m", "l", "r", "z"
print a & b  # "a", "c"
print a ^ b  # "b", "d", "m", "l", "r", "z"
```

4. Diccionarios

```
edad_de = {"Eva":23, "Ana":19, "Oscar":41}
print edad_de["Ana"]  # Imprime 19
edad_de["Eva"] = 18   # Cambia un valor
edad_de["Juan"] = 26  # Añade un elemento
del edad_de["Oscar"]  # Borra un elemento
edad_de.keys()        # ["Eva", "Juan", "Ana"]
edad_de.values()      # [18, 26, 19]
for key,value in edad_de.items():
  print key,"->",value
dict([("a",1),("b",2),("c",3)])  # {"a":1, "c":3, "b":2}
dict(a=1, b=2, c=3) # {"a":1, "c":3, "b":2}
```

Compresión de listas

```
x = [1, 2, 3, 4, 5, 6, 7, 8]
xx = [n ** 2 for n in x if n > 4]  # xx=[25, 36, 49, 64]
```

```
l = [0, 1, 2, 3]
m = ["a", "b"]
n = [s*v for s in m
         for v in l
         if v > 0]    # n = ["a", "aa", "aaa", "b", "bb", "bbb"]

dict([(x, x**2) for x in (2, 4, 6)]) # {2:4, 4:16, 6:36}
```

6.3. Control de flujo

Lazos

```
frutas=["naranjas", "uvas"]
for f in frutas:
  print f, len(f)  # naranjas, 8; uvas, 4

for i in range(len(frutas)):
  print i, frutas[i] # 0, naranjas; 1, uvas

nf = raw_input("Añade otra fruta: ")
while nf:             # Si la entrada no está vacía
  frutas.append(nf)   # añádela a la lista
  nf = raw_input("Añade otra fruta: ")
```

Condicionales

```
x = int(raw_input("Introduce un entero: "))
if x < 0:
  x = 0
  print "Negativo cambiado a 0"
elif x == 0:
  print "Cero"
else:
  print "Positivo"
```

Funciones

```
def compra(fr, nf="manzanas"):
  fr.append(nf)

frutas=[]     # También frutas=list()
```

```
compra(frutas, "peras")
compra(frutas)
compra(nf="limones", fr=frutas)
print frutas   # peras, manzanas, limones
```

Funciones con argumentos arbitrarios

```
def fun(*args, **kwargs):
  for arg in args: print arg
  for kw in kwargs.keys(): print kw, ":", kwargs[kw]
fun("peras", 1, manzanas=2, limones=3)
```

Salida:

```
peras
1
limones : 3
manzanas : 2
```

6.4. Orientación a objetos

```
class fruteria(object):
   """Ejemplo simple de clase"""
   def __init__(self, f):
       self.stock = list()
       self.stock.append(f)
   def compra(self, f):
       self.stock.append(f)
   def vende(self, f):
       if f in self.stock:
           self.stock.remove(f)
       else:
           print f, "no disponible"

def main():
   mi_fruteria = fruteria("pera")
   mi_fruteria.compra("manzana")
   print mi_fruteria.stock       # ["pera", "manzana"]
   mi_fruteria.vende("pera")
   mi_fruteria.vende("platano")  # platano no disponible
   print mi_fruteria.stock       # ["manzana"]
   mi_fruteria.vende("pera")     # pera no disponible
   print mi_fruteria.__doc__     # Ejemplo simple de clase
```

```
if __name__ == "__main__":
   main()
```

Herencia múltiple

Se permite herencia múltiple:

```
class fruteria(object):
    def que_vendo(self):
        print "Vendo frutas"

class carniceria(object):
    def que_vendo(self):
        print "Vendo carne"

# Herencia múltiple
class tienda(carniceria, fruteria):
    pass

# La clase carniceria está más a la
# izquierda en la deficición de tienda
tienda().que_vendo() # Vendo carne
```

Métodos y atributos privados

Los métodos o atributos privados se definen con dos guiones bajos antes del nombre (y no pueden terminar en dos guiones bajos)

```
class Ejemplo(object):
    def publico(self):
        print "Uno"
        self.__privado()

    def __privado(self):
        print "Dos"

ej = Ejemplo()
ej.publico()   # Imprime Uno Dos
ej.__privado() # Da un error
```

6.5. Procesamiento de textos

Muchos métodos de interes para manejar cadenas de texto

```
# Elimina caracteres y separa por espacios
l = "Hola que tal!".strip("!").split() # l=["Hola", "que", "tal"]
# Une utilizando un caracter
s = ",".join(l)  # s="Hola,que,tal"
#Cuenta el número de ocurrencias de un caracter
c = s.count(",") # c=2
# Reemplaza un caracter por otro
ss = s.replace(",", "\t")  # ss="Hola    que    tal"
# Separa por otro tipo de caracter, e invierte la lista
l=ss.split("\t")
l.reverse()     # l=["tal", "que", "Hola"]
# Localiza una subcadena en el string
c=ss.find("tal")  # c=9
c=ss.find("tall") # c=-1 (no encuentra la subcadena)
# Separa por líneas
ml = """Esto es
un texto con
varias lineas"""
l = ml.splitlines()  # l=["Esto es", "un texto con", "varias lineas"]
```

Expresiones regulares

```
import sys, re  # Módulo para REGEXPR
# Comprueba direcciones de e-mail
s=raw_input("Introduce un e-mail: ")
if re.match("\w+@\w+\.\w+((\.\w+)*)?", s):
  print "Dirección correcta"

# Busca URLs en un fichero de texto
try:
  f = open("fich.txt","r")
except IOError:
  print "No puedo abrir"
  sys.exit(1)
for l in f:
  # Busca todas las URLs en la línea actual
  # y guárdalas (sin http) en la lista h
  h = re.findall("http://([^\s]+)", l)
  if h:         # Si la lista no está vacía
    for w in h: # recorrela e imprime las URLs
        print w

# Separa un string en una lista
```

```
s = "Uno:Dos.Tres-Cuatro"
l = re.split("[:.-]", s)
```

6.6. Otros aspectos

- Funciones anónimas (lambda): permiten definir una función de una instrucción en una línea de código

```
neto = lambda bruto, iva=21: bruto + (bruto*iva/100)
print neto(100)          # 121

def suma (n):
  return lambda x: x + n
f=suma(2)
g=suma(8)
print f(10), g(10) # 12, 18
print suma(5)(11)  # 16
```

- Métodos map, filter y reduce

```
foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
print filter(lambda x: x % 3 == 0, foo)
# [18, 9, 24, 12, 27]
print map(lambda x: x * 2 + 10, foo)
# [14, 46, 28, 54, 44, 58, 26, 34, 64]
print reduce(lambda x, y: x + y, foo)
# 139
```

- Decoradores: permiten cambiar el comportamiento dinámico de una función

```
def check(f):
  def wrapper(*args, **kwargs):
    if 0 in args:
      return None
    else:
      return f(*args, **kwargs)
  return wrapper

@check
def inv(*args):
  return [1.0/x for x in args]
```

```
print(inv(1,2,3))
print(inv(1,0,3)) # None
```

- Iteradores

```
# Iterador implícito en el for
for i in "papanatas":
  print i,        # p a p a n a t a s

# Iterador explícito
it = iter("papanatas")
it.next()  # p
it.next()  # a
it.next()  # p
it.next()  # a
it.next()  # n
it.next()  # a
it.next()  # t
it.next()  # a
it.next()  # s
it.next()  # Error
```

- Generadores

```
a = xrange(1000000)           # a no es una lista
b = (n for n in a if n%2==0)  # b no es una lista
print b   # <generator object <genexpr> at 0xb77c939c>
for i in b: print i,  # 2 4 6 8 10 ...

def generador():
    i = 0
    while True:     # un iterador infinito
        yield i     # devuelve i en este punto
        i = i + 1
mi_gen = generador() # creamos el generador
mi_gen.next()        # 0
mi_gen.next()        # 1
mi_gen.next()        # 2
```

- Métodos especiales:

```
class miclase:
  def __init__(self, n1, n2):
    self.n1 = n1
    self.n2 = n2
  # Representación del objeto como string
  def __str__(self):
    return "Soy un miclase con: n1="
            +str(self.n1)+", n2="+str(self.n2)
  # Permite asignar nuevos atributos
  def __setattr__(self, name, val):
    self.__dict__[name] = val
  # Se llama con atributos no conocidos
  def __getattr__(self, name):
    return "No se lo que es "+name
o = miclase(2, 5)
print o    # Soy un miclase con: n1=2, n2=5
o.n3 = 5
print o.n3 # Imprime "5"
print o.n4 # Imprime "No se lo que es n4"
```

6.7. Subprocesos

El módulo `subprocess` permite lanzar subprocesos, por ejemplo, comandos del SO

```
import subprocess
# Ejecuta el comando df -h (sintaxis de línea de comandos)
subprocess.call("df -h", shell=True)
# Ejecuta ls /usr/ppp, redireccionando la salida estándar
# y de error. El código de salida a ret
ret=subprocess.call(["ls", "/usr/ppp"],
                    stdout=open("/dev/null","w"),
                    stderr=subprocess.STDOUT)
# Ejecuta df -h; la salida estándar va al objeto p
p=subprocess.Popen(["df", "-h"], stdout=subprocess.PIPE)
# Lee e imprime las líneas de la salida de df -h
out = p.stdout.readlines()
for line in out:
   print line,
```

6.8. Otros módulos de interés

os Uso de funcionalidades dependientes del SO

- `os.getlogin()` nombre de login del usuario
- `os.getloadavg()` carga media del sistema
- `os.getcwd()` obtiene el directorio actual
- `os.chdir(`*`path`*`)` cambia el directorio actual a *path*
- `os.listdir(`*`path`*`)` lista de todas las entradas del directorio *path*

os.path Manipulación de ficheros y/o directorios

- `os.path.isfile(`*`path`*`)` True si *path* es un fichero regular
- `os.path.split(`*`path`*`)` Divide *path* en directorio+fichero
- `os.path.splitext(`*`path`*`)` Divide *path* en nombre_fichero+ extensión
- `os.path.getsize(`*`path`*`)` Devuelve el tamaño de *path*

glob Expansión de nombres de ficheros estilo UNIX (*globbing*)

- `glob.glob(`*`expr`*`)` Lista de ficheros indicados por *expr* (puede contener comodines)

shutil Operaciones de alto nivel con ficheros

- `shutil.copy(`*`src, dst`*`)` Copia el fichero *src* al fichero o directorio *dst*
- `shutil.move(`*`src, dst`*`)` Mueve recursivamente un fichero o directorio

tempfile Genera ficheros y directorios temporales

- `tempfile.NamedTemporaryFile()` Crea un fichero temporal con nombre

optparse Parsea las opciones en línea de comandos (reemplazado por `argparse`)

gzip, bz2, zipfile, tarfile Manejo de fichero comprimidos

sys Parametros y funciones dependientes del sistema

- `sys.argv` Lista de argumentos en línea de comandos (`sys.argv[0]` es el nombre del script)
- `sys.exit(`*`[code]`*`)` Termina el script con código de salida *code*

6.9. Ejemplos

1. En un directorio, renombra *.xml a *.html

```
import os.path, glob, shutil, optparse
def main():
    p = optparse.OptionParser(description="Renombra XML a HTML",
                              usage="%prog [directory]")
    options, args = p.parse_args()
    if len(args) == 1:
        # Chequea que sea un directorio
        if not os.path.isdir(args[0]):
            print args[0] + " no es un directorio"
            sys.exit(1)
        try:
            os.chdir(args[0]) # Cambia al directorio
            # Recorre los ficheros .xml
            for f in glob.glob("*.xml"):
                # Construye el nuevo nombre y renombra los ficheros
                new = os.path.splitext(f)[0] + ".html"
                shutil.move(f, new)
        except:
            print "Hubo un problema ejecutando el programa."
    else:
        p.print_help()
if __name__ == "__main__":
    main()
```

2. Muestra información sobre un proceso en ejecución

```
from subprocess import Popen, PIPE
proc = raw_input("Proceso a chequear: ")
try:
    # Ejecuta el comando ps y obten la salida
    output = Popen("ps -edf | grep "+proc,shell=True,stdout=PIPE)
    procs = output.stdout.readlines()
    for procinfo in procs:
        # Separa la salida en campos
        info = procinfo.split()
        # Muestra los resultados
        print "\n\
        Ejecutable:\t", info[-1], "\n\
        Propietario:\t", info[0], "\n\
        PID:\t\t", info[1], "\n\
```

```
        PPID:\t\t", info[2], "\n\
        Hora inicio:\t", info[4], "\n"
except:
    print "Hubo un problema ejecutando el programa."
```

3. Realiza acciones sobre un tar, seleccionándolas de un menú

```
import tarfile, sys
try:
  f = True
  while f:
    # Abre el fichero tar (especificado como argumento)
    tar = tarfile.open(sys.argv[1], "r")

    # Presenta el menú y obtiene la selección
    selection = raw_input("""
    Selecciona
       1 para extraer un fichero
       2 para mostrar información sobre un fichero en """
         + sys.argv[1] + """
       3 para listar los ficheros de """ + sys.argv[1] +
       """
       4 para terminar""" + "\n")

    # Realiza la acción en función de la selección
    if selection == "1":
      filename = raw_input("Indica el fichero a extraer:  ")
      tar.extract(filename)
    elif selection == "2":
      filename = raw_input("Indica el fichero a inspeccionar:  ")
      for tarinfo in tar:
        if tarinfo.name == filename:
          print "\n\
          Nombre:\t", tarinfo.name, "\n\
          Tamaño:\t", tarinfo.size, "bytes\n"
    elif selection == "3":
      print tar.list(verbose=True)
    elif selection == "4":
      f = False
    else:
        print "Selección incorrecta"
```

```
except:
    print "Hubo un problema ejecutando el programa."
```

Referencias

- Python Official Website: página principal de Python
- Python Documentation: documentación diversa, tutoriales, etc.
- The Python tutorial: un buen sitio para empezar
- The Python Standard Library: la librería estándar
- Módulos útiles
- Índice alfabético de módulos
- Python para todos: tutorial en castellano

7. Introducción a Perl y Ruby

7.1. Perl

Principales aplicaciones de Perl:

- Administración de sistemas
- Desarrollo web
- Programación en red
- Desarrollo de GUI
- ...

Algunas características

- Combina características de shell, awk y sed con otros lenguajes de alto nivel
- Soporte de distintos paradigmas de programación (imperativa, orientada a objetos y funcional)
- Potente sistema de procesamiento de texto mediante expresiones regulares
- Enorme colección de módulos disponibles

Ejecución de un script Perl

- Directamente en la línea de comandos:

```
# Renombra *.txt a *-2010.txt
$ perl -e 'foreach (<*.txt>)
>  { s/\.txt$//; rename("$_.txt", "$_-2010.txt") }'
```

- En un script

```
#!/usr/bin/perl
use strict;   # Exige predeclarar las variables (my)
use warnings; # Avisa de posibles errores
#
# Abre el fichero de contraseñas y lee cada línea.
my $filename = "/etc/passwd";  # Nombre del fichero
open(FILE, "<", $filename)     # Abre el fichero (solo lectura)
  or die "No puedo abrir: $!"; # Termina si falla
while(my $line = <FILE>) {     # Lee cada línea
        print $line;
}
close(FILE);                  # Cierra el fichero
```

Tipos de datos en Perl

1. Escalares (números o strings)

```
$a = "manzanas";
$b = "peras";
print $a." y ".$b."\n"; # Muestra "peras y manzanas"
print "$a y $b\n";      # Muestra "peras y manzanas"
```

2. Arrays

```
@frutas = ("naranjas", "limones", "uvas");
print $frutas[2];       # uvas
($n, $l) = @frutas;     # $n="naranjas", $l="limones"
push(@frutas, "cocos"); # $frutas[3] = "cocos"
$c = pop(@frutas);      # $c = "cocos"
$nf = scalar(@frutas);  # $nf = 3
$fr = "@frutas";        # $fr = "naranjas limones uvas"
@fo = split(/ /, $fr);  # @fo = ("naranjas", "limones", "uvas")
```

3. Mapas (arrays asociativos)

```
%edad_de = {
    Eva  => 23,
    Ana  => 19,
    Oscar => 41
}
print $edad_de{Ana};    # Imprime 19
$edad_de{Eva}=18;       # Cambia un valor
$edad_de{Juan} = 26;    # Añade un elemento al mapa
```

4. Variables especiales
   - `$_` Variable por defecto (la mayoría de las funciones de Perl toman `$_` como argumento por defecto)
   - `@ARGV` array con los argumentos de la línea de comandos
   - `%ENV` Mapa con las variables de entorno

Control de flujo

Lazos

```
foreach (@frutas) { # Recorre el array
    print $_."\n";  # Imprime un elemento por
                    # línea. El punto concatena
}                   # dos strings.

print "\nAñade más frutas ";  # Imprime un mensaje
$a = <STDIN>;                # Lee de la entrada estándar
chop $a;                     # y elimina el \n
while ( $a ) {               # Si la entrada no está vacía
    push(@frutas, $a);       # añádela al array
    $a = <STDIN>; chop $a;   # y lee una nueva entrada
}
```

Condicionales

```
if ( not $tengo_manzanas ) {
  compra(\@frutas,"manzanas" ); # El array se pasa por
}                               # referencia
```

Alternativa:

```
unless ($tengo_manzanas) {
  compra(\@frutas,"manzanas");
}
```

También es válido:

```
compra(\@frutas,"manzanas") if not $tengo_manzanas;
```

Subrutinas

- Los parámetros se recogen en `@_`

```
sub compra {
    ( $array, $string ) = @_; # Los parametros se recogen
                              # como escalares
    push(@$array, $string);   # La referencia se convierte
                              # a array
}
```

Expresiones regulares

```
             # Sin argumentos, lee la entrada estandar
while(<>) {  # con argumentos, usa estos como nombres
             # de ficheros y los lee línea a línea
  print if /http:\/\//; # Muestra las líneas con http://
  print if s/ttx/txt/ig; # Muestra las líneas con "ttx"
                         # y hace el cambio por "txt"
                         # g=global, i=case insensitive
}

$string = "oCme mas futra";
$string =~ s/oCme/Come/;  # =~ Aplica sustitución a $string
$string =~ s/futr/frut/;
print $string; # Imprime "Come mas fruta"
```

Ejemplos

1. Muestra las terminaciones de los ficheros del directorio actual

   ```
   #!/usr/bin/perl
   use strict;
   use warnings;
   foreach (glob("*")) {       # Recorre los ficheros
   ```

```
    my @file = split(/\./); # Los separa por .
    my $term = pop(@file);  # Extrae el último elemento
    print "$term\n";
}
```

2. En un directorio, renombra *.xml a *.html

```
#!/usr/bin/perl
use strict;
use warnings;
unless (scalar(@ARGV) == 1) {
  print "Necesito un directorio como argumento\n"; exit 1;
}
if( not -d $ARGV[0] ) {
  print "$ARGV[0] no es un directorio\n"; exit 1;
}
# Cambia al directorio
chdir $ARGV[0];
# Recorre los ficheros .xml
foreach my $file (glob "*.xml") {
  # Construye el nuevo nombre
  my $new = substr($file, 0, -3) . "html";
  # Renombra los ficheros
  rename $file, $new;
}
```

3. Lee un fichero de texto numerando las líneas no vacías

```
#!/usr/bin/perl
use strict;
use warnings;
open(my $fichero, "<", "f.txt")
     or die "No puedo abrir f.txt:$!";
my $nl="001"; # Entero de tres dígitos
while(<$fichero>) {
  if(!/^$/) {        # Sólo las líneas no vacías
    print "$nl $_";  # Pon un número de línea
    $nl++;
  }
  else {
    print "$_";      # Línea vacía sin número
```

```
  }
}
```

4. Script para añadir usuarios al sistema

```
use strict; use warnings;
# Módulo para leer parámetros de entrada
use Getopt::Long;
my $addusr = "/usr/sbin/adduser";
my $nombre=""; my $apellido="";
# Obtiene los parámetros
GetOptions("nombre=s" => \$nombre,
     "apellido=s" => \$apellido ) or uso();
# Comprueba los parámetros sean correctos
if( not $nombre or not $apellido ) {
    uso();
}
if ( $nombre !~ /^[a-zA-Z]+$/) {
    uso("El nombre debe ser alfabético");
}
if ( $apellido !~ /^[a-zA-Z]+$/) {
    uso("El apellido debe ser alfabético");
}

# Construye el username
my $username = lc( substr($apellido, 0, 1) . $nombre);
# Directorio HOME
my $home     = "/home/$username";
# Comando a ejecutar
my $comando = qq($addusr --home $home --disabled-password \\
            --gecos "$nombre $apellido" $username);
system $comando; # Ejecuta el comando

# Error e información de uso
sub uso {
  my ($msg) = @_;      # Recogo los parámetros
  if ($msg) {          # Si se pasa un mensaje de error,
     print "$msg\n\n"; # lo muestra
  }
  print "Usar: $0 --nombre Nombre --apellido Apellido\n";
  exit;
}
```

Referencias

- The Perl Directory: página principal de Perl
- Perl programming documentation: extensa documentación
- Comprehensive Perl Archive Network: módulos y documentación de Perl
- The CPAN search site: para buscar en el CPAN

7.2. Ruby

Lenguaje dinámico, de propósito general, creado a mediados de los 90 por Yukihiro "Matz"Matsumoto

- Expresiones regulares nativas similares a las de Perl
- Soporte de múltiples paradigmas: imperativo, orientado a objetos y funcional
- "Todo" es un objeto
- Amplia librería estándar

Ejemplo sencillo:

```
#!/usr/bin/ruby
=begin
Abre y lee un fichero
Se usa un bloque (entre do - end)
El identado no es necesario
El fichero se cierra
automáticamente al acabar el bloque.
=end
File.open("/etc/passwd", "r") do |f1|
  while linea = f1.gets
    puts linea
  end
end    # Fin del bloque
```

Tipos de datos en Ruby

1. Arrays

```
frutas=[ "naranjas", "uvas", 123, "limones", "uvas" ]
frutas<<"peras"     # Añade un string
frutas.delete(123)
frutas.uniq!        # Elimina elementos duplicados
frutas.insert(2, %w{fresas pomelos}) # Inserta otro array
                                     # %w -> array de strings
                                     # sin usar comillas
puts frutas # naranjas,uvas,fresas,pomelos,limones,peras
puts frutas.length  # 5
puts frutas[2][1]   # pomelos
frutas.delete_at(2)
frutas.insert(3, "cerezas", "kiwis")  # Inserta
frutas.sort! # Ordena ``in-place''
puts frutas  # cerezas, kiwis, limones, naranjas, peras, uvas
```

2. Rangos

```
nums = -1..9
puts nums.include?(10) # false (10 no en el rango)
puts nums === 0        # true (0 en el rango)
puts nums.first        # -1
puts nums.last         # 9
puts nums.to_a         # [-1,0,1,2,3,4,5,6,7,8,9]
puts nums.to_s         # "-1..9"
array = nums.reject {|i| i < 7}
puts array             # [7, 8, 9]
```

3. Arrays asociativos

```
edad_de = {'Eva'=>23, 'Ana'=>19, 'Oscar'=>41}
puts edad_de['Ana']     # Imprime 19
edad_de['Eva'] = 18     # Cambia un valor
edad_de['Juan'] = 26    # Añade un elemento
edad_de.delete('Oscar') # Borra un elemento
```

Control de flujo

Lazos

```
frutas=["naranjas", "uvas"]

# Bloque usando do-end
frutas.each do |f|
  puts "#{f}:#{f.length}"  # naranjas:8
end                            # uvas:4

print "Añade otra fruta: "
nf = gets.chomp    # Lee stdin y elimina el \n
while nf != ""     # Si la entrada no está vacía
  frutas<<nf.to_s  # añádela a la lista
  print "Añade otra fruta: "
  nf = gets.chomp
end

# Bloque usando llaves
3.times { |i| puts i }   # 0, 1, 2
```

Condicionales

```
print "Introduce un entero: "
x = gets.chomp.to_i
if x < 0
  x = 0
  puts "Negativo cambiado a 0"
elsif x == 0
  puts "Cero"
else
  puts "Positivo"
end

# Forma unless
unless x == 0
  puts x
end
# Case
scale = 8
case scale
  when    0: puts "lowest"
  when 1..3: puts "medium-low"
  when 4..5: puts "medium"
```

```
  when 6..7: puts "medium-high"
  when 8..9: puts "high"
  when   10: puts "highest"
  else       puts "off scale"
end
```

Funciones

```
# Argumento con valor por defecto
def compra(fr, nf="manzanas")
  fr<<nf
end
# Número de argumentos variable
def compram(fr, *nf)
  # Recorro todos los argumentos
  nf.each { |f| fr<<f }
end

frutas=[]
# Los paréntesis no son obligatorios
compra frutas, "peras"
# Usa el valor por defecto
compra(frutas)
# Usa múltiples argumentos
compram(frutas, "limones", "naranjas")
puts frutas   # peras, manzanas, limones, naranjas
```

Expresiones regulares

```
# Comprueba direcciones de e-mail
print "Introduce un e-mail: "
s = gets.chomp
if /\w+@\w+\.\w+((\.\w+)*)?/.match(s)
  puts "Dirección correcta"
end

# Busca URLs en un fichero de texto
# Abre el fichero de solo lectura
# comprobando excepciones
begin
```

```
  f = File.open("fich.txt","r")
rescue Exception => msg
  print "No puedo abrir --> ", msg, "\n"
  exit(1)
end

# Expresión regular a buscar (\s == [:space:])
urlreg = /http:\/\/([^\s]+)/
nl=1
f.each do |l|
  # Busca todas las URLs en la línea actual
  # e imprimelas
  l.scan(urlreg) { |m| print "Línea #{nl}-><#{m}>\n" }
  nl+=1
end
f.close

# Corrige un string
s = "oCme más futra"
s.gsub!("oCme", "Come")
s.gsub!("futr", "frut")
puts s  # Imprime "Come más fruta"
# Separa un string en una lista
s = "Uno:Dos.Tres-Cuatro"
l=s.split(/[:.-]/)
```

Ejemplos

1. En un directorio, renombra *.xml a *.html

```
# Módulo con utilidades para ficheros
require 'fileutils'
# Comprueba argumentos
if ARGV.length < 1
  puts "Necesito un directorio como argumento"
  exit
end
dir=ARGV[0]

# Chequea que sea un directorio
unless File.directory?(dir)
  puts dir+" no es un directorio"
```

```
  exit
end

# Recorre los ficheros .xml
begin
  # Cambia al directorio
  FileUtils.cd(dir)
  Dir.glob("*.xml") do |f|
    # Construye el nuevo nombre
    new = File.basename(f, ".xml")+".html"
    # Renombra los ficheros
    File.rename(f, new)
  end
rescue Exception => msg
  puts "Error: "+msg
end
```

2. Muestra información sobre un proceso en ejecución

```
print "Proceso a chequear: "
proc = gets.chomp
begin
  # Ejecuta el comando ps y obten la salida
  output = `ps -edf|grep #{proc}`
  # Separa la salida en campos
  procinfo = output.split()

  # Muestra los resultados
  puts "Ejecutable  : #{procinfo[7]}"
  puts "Propietario : #{procinfo[0]}"
  puts "PID         : #{procinfo[1]}"
  puts "PPID        : #{procinfo[2]}"
  puts "Hora inicio : #{procinfo[4]}"
rescue Exception => msg
  puts "Error: "+msg
end
```

3. Busca recursivamente ficheros que cumplen un patrón

```
# Módulo adicional
require 'find'
print "Directorio inicial: "
```

```
searchpath = gets.chomp
print "Patrón de búsqueda: "
pattern = gets.chomp
# Busca recursivamente
Find.find(searchpath) do |path|
  # Comprueba si el patrón corresponde con el fichero
  if File.fnmatch(pattern, File.basename(path))
    # Muestra el nombre del fichero
    puts  "Fichero             : " +  File.basename(path)
    # Información sobre el fichero
    stat = File.stat(path)
    # Muestra los permisos en octal
    printf("Permisos            : %o\n", stat.mode)

    # Muestra el UID y el GID del propietario
    print "UID del propietario : "
    puts stat.uid
    print "GID del propietario : "
    puts stat.gid
    # Muestra el tamaño del fichero
    print "Tamaño (bytes)      : "
    puts stat.size
    puts "---------------------------------------"
  end
end
```

Referencias

- Página principal de Ruby
- Ayuda y documentación para Ruby
- Core API docs para Ruby 1.8.7
- Ruby en 20 minutos