

SIMM
Apuntes GNU/Linux

Héctor Fiel Martín
1º GSI

Copyright ©2008 Héctor Fiel Martín. Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos de Cubierta Trasera. Una copia de la licencia está incluida en la sección titulada GNU Free Documentation License.

Índice general

1. Comandos	8
1.1. Introducción: Particiones	8
1.1.1. Las particiones del disco duro	8
1.1.2. Preparación de los equipos del aula	9
1.1.3. Particiones en GNU/Linux	11
1.1.4. El gestor de arranque GRUB	13
1.2. Sistemas de ficheros	15
1.2.1. Características:	15
1.2.2. Tipos de archivos	16
1.2.3. superbloque	16
1.2.4. tipos de sistema de ficheros	18
1.2.5. estructura lógica	19
1.2.6. montaje de sistemas de archivos: comando mount	19
1.3. Inicio del sistema GNU/Linux	21
1.3.1. Arranque en Ubuntu: Upstart	23
1.4. Eliminar clave de root del sistema	23
1.5. Editores de texto	23
1.5.1. VI	23
1.5.2. Nano	27
1.6. Uso de comandos	27
1.6.1. Ayuda en GNU/Linux	27

1.6.2.	Comandos básicos	27
1.6.3.	Expresiones Regulares	28
1.6.4.	find. Buscador de archivos	30
1.6.5.	locate	32
1.6.6.	which	32
1.6.7.	cat. Redirecciones y algo más	32
1.6.8.	wc: word count	32
1.6.9.	grep	33
1.6.10.	sed	34
1.6.11.	cut	36
1.6.12.	tr	37
1.6.13.	tee	37
1.6.14.	pr	37
1.6.15.	lp	38
1.6.16.	file	38
1.6.17.	sort	38
1.6.18.	uniq	38
1.6.19.	tail head	39
1.6.20.	stat	39
1.6.21.	touch	39
1.6.22.	gzip y gunzip	39
1.6.23.	bzip y bunzip	39
1.7.	Comandos avanzados	39
1.7.1.	zip y unzip	39
1.7.2.	tar	39
1.8.	Otros comandos útiles	40
1.8.1.	df	40
1.8.2.	nano	40

1.8.3.	file-roller	40
1.8.4.	gnome-commander	40
1.8.5.	apt-get	40
1.8.6.	aptitude	40
2.	Administración del sistema. Ficheros en red	41
2.1.	Administración de usuarios y grupos	41
2.1.1.	Tabla de usuarios	41
2.1.2.	Extensión de la tabla de uuarios	42
2.1.3.	Tabla de grupos	42
2.1.4.	Procedimientos para crear usuarios y grupos	42
2.1.5.	Atributos de protección de procesos	43
2.1.6.	Atributos de protección de ficheros	43
2.1.7.	Protección básica	44
2.1.8.	Cambio de los atributos de protección	44
2.1.9.	Los bits SETUID y SETGID en ejecutables	44
2.1.10.	SETGID en directorios	44
2.1.11.	Comando UMASK	45
2.1.12.	Grupos privados	45
2.1.13.	Comandos relacionados	45
2.2.	Gestión de procesos	45
2.2.1.	Introducción	45
2.2.2.	Control de los procesos: fg, bg	47
2.2.3.	Comunicación con los procesos: kill, killall	47
2.2.4.	Prioridades y ámbito: nice, renice, nohup	47
2.2.5.	pstree	47
2.3.	Servidor NFS	48
2.3.1.	Pasos a seguir (EN FEDORA):	48

2.3.2.	Clientes NFS	50
2.4.	SAMBA	50
2.4.1.	Tipos de servidores SAMBA	53
2.5.	LDAP: Administración de dominios GNU/Linux usando LDAP	55
2.5.1.	Introducción	55
2.5.2.	Concepto de dominio	55
2.5.3.	Servicios de directorio y LDAP	56
2.5.4.	Implementar un dominio GNU/Linux con LDAP	56
2.5.5.	Instalación de LDAP	57
3.	Scripts	58
3.1.	Introducción a Shell Scripting en Bash	58
3.1.1.	Diferencias entre ", ' y '	59
3.1.2.	Ejecución secuencial y condicional	60
3.2.	Evolución de las shells	60
3.2.1.	Bourne	60
3.2.2.	CShell	61
3.2.3.	TCSH	61
3.2.4.	Shell Korn	62
3.2.5.	POSIX	62
3.2.6.	Bash	62
3.2.7.	Identificar la shell actual	62
3.3.	Variables en Scripts	63
3.3.1.	Operadores aritméticos para let	63
3.3.2.	Comparación de cadenas	64
3.3.3.	Comparación de datos numéricos	65
3.3.4.	Comparación de ficheros	65
3.3.5.	Errores típicos	65

- 3.4. Estructuras condicionales 66
 - 3.4.1. if 66
 - 3.4.2. Case 66
 - 3.4.3. for 67
 - 3.4.4. for (estilo C) 68
 - 3.4.5. seq 68
 - 3.4.6. while 68
 - 3.4.7. until 69
 - 3.4.8. select 69
 - 3.4.9. Funciones 70
 - 3.4.10. Consulta avanzada de variables 70
- 3.5. Introducción de datos en scripts 72
 - 3.5.1. Read 72
- 3.6. Parámetros 72
 - 3.6.1. Desplazamiento de parámetros: **shift** 73
- 3.7. Arrays en scripts 74
- 3.8. Ejercicios Scripts 74
- 3.9. AWK 80
 - 3.9.1. sintaxis AWK 80
 - 3.9.2. Tipos de patrones 81
 - 3.9.3. Operadores 82
 - 3.9.4. Variables internas de AWK 83
 - 3.9.5. Arrays y Matrices 83
 - 3.9.6. Funciones internas 84
 - 3.9.7. Manejo de cadenas 84
 - 3.9.8. Ejercicios 85

GNU Free Documentation License	88
1. APPLICABILITY AND DEFINITIONS	88
2. VERBATIM COPYING	90
3. COPYING IN QUANTITY	90
4. MODIFICATIONS	91
5. COMBINING DOCUMENTS	92
6. COLLECTIONS OF DOCUMENTS	93
7. AGGREGATION WITH INDEPENDENT WORKS	93
8. TRANSLATION	93
9. TERMINATION	93
10. FUTURE REVISIONS OF THIS LICENSE	94
ADDENDUM: How to use this License for your documents	94

Capítulo 1

Comandos

1.1. Introducción: Particiones

1.1.1. Las particiones del disco duro

Empleamos las particiones para dividir el espacio total de un disco duro de forma que podamos aprovecharlo para instalar más de un sistema operativo en él, o para permitir separar los datos del sistema operativo de los datos del usuario (como sus documentos), de forma que un problema del sistema no afecte a los documentos almacenados. Por norma general, los sistemas Windows se apropian de todo el espacio del disco duro durante la instalación. Por ello, debemos o bien modificar este aspecto durante la instalación del sistema, o recurrir a otros programas que nos ayuden a realizar dicha división, tales como Paragon o Partition Magic.

En un disco podemos tener tres tipos de particiones:

- **Particiones primarias** Originalmente los discos duros sólo podían tener este tipo de particiones. La estructura de las particiones se realiza mediante una lista denominada “Tabla de particiones”. Dicha tabla no puede almacenar más que la información de cuatro particiones, por lo tanto existe un límite en el número de particiones máximas (hasta 4 primarias, o hasta 3 primarias y una extendida) que pueden coexistir en un mismo disco duro. Las particiones primarias se suelen emplear para instalar sistemas DOS y Windows, ya que sus gestores de arranque (al menos en las versiones más antiguas) no eran capaces de iniciar un sistema instalado en otro tipo de particiones.
- **Particiones extendidas** Las particiones extendidas se crearon para superar el límite de cuatro particiones por disco, y consisten en una división del disco (que ocupa uno de los cuatro espacios disponibles en la tabla de particiones), dentro de la cual se pueden crear ilimitadas unidades lógicas. Por lo tanto, las particiones extendidas no pueden utilizarse para almacenar datos o sistemas operativos de manera directa, sino que dentro de ellas es obligatorio crear unidades lógicas. Algunos sistemas DOS muy antiguos no son capaces

de entender las particiones extendidas, y no pueden acceder a los datos almacenados en ellas. En un disco, sólo puede haber una única partición extendida.

- **Unidades lógicas** Se trata de una división de la partición extendida, que puede ser utilizada del mismo modo que una partición primaria, para instalar sistemas operativos o guardar datos en su interior. El número de unidades lógicas que puede haber en la partición extendida es ilimitado. Sólo algunos sistemas operativos (GNU/Linux, Unix, BSD, OS/2, Windows NT y sus derivados) son capaces de arrancar desde una unidad lógica.

Además de la división en particiones, para poder utilizar el espacio del disco cada partición deberá estar formateada en un sistema de ficheros. El sistema de ficheros lo que permite es organizar en el espacio de la partición en carpetas y archivos, manteniendo un índice de que posición ocupa cada dato dentro del disco.

Los sistemas Windows emplean los formatos FAT o NTFS, ofreciendo este último características de seguridad y gestión avanzadas, pero siendo más difícil su acceso desde otros sistemas operativos (los sistemas GNU/Linux más modernos son capaces de leer NTFS sin problemas, pero la escritura, aunque posible, sigue siendo poco recomendable). Los sistemas GNU/Linux emplean el sistema de ficheros Ext2 (ó Ext3 más recientemente). Windows es incapaz de leer estas particiones (aparecen como “partición desconocida”). Además, un sistema GNU/Linux necesita una partición especial, denominada swap, para actuar como partición de intercambio (es lo que se denomina memoria virtual, donde se guardan los datos que por su tamaño no pueden entrar en la memoria RAM en un momento dado). En Windows, el espacio de intercambio se lleva a cabo mediante un fichero (denominado pagefile.sys, y situado por defecto en la carpeta raíz de C:).

1.1.2. Preparación de los equipos del aula

En el caso de los ordenadores disponibles en la clase, dado que necesitamos instalar varios sistemas operativos, tendremos que dividir el disco en particiones. La secuencia recomendada de preparación de los equipos es la siguiente:

1. Instalación de la imagen de Windows XP de los sistemas:

Esta imagen contiene una instalación básica de Windows XP con todos los drivers necesarios para los equipos. Esta imagen ocupa todo el disco duro de los equipos, con una única partición (C: en Windows), por lo tanto habrá que cambiar su tamaño para permitir crear otras particiones.

2. Creación de nuevas particiones:

Iniciando el PC desde CD, y empleando una utilidad como Paragon o Partition Magic, reducimos el tamaño de la partición de Windows XP (dejaremos el suficiente espacio para que podamos instalar los programas necesarios sin problemas, con 10-12GB es más que suficiente), y creamos otras dos particiones primarias (una para instalar Windows Server

2003 y otra de reserva, para el caso de que necesitemos instalar otro sistema Windows, podemos darles los mismos tamaños que a la anterior), y una extendida, con el resto del espacio libre. Dentro de la extendida, crearemos tres unidades lógicas. Una de ellas será para el archivo de intercambio de GNU/Linux, (tipo swap) y deberá tener un tamaño entre 1 y 2 GB. La otra será donde montemos la carpeta raíz (/) de GNU/Linux, y le daremos también un tamaño en torno a los 10GB. Por último, creamos en el resto del espacio una unidad lógica de tipo FAT32, que será la que emplearemos para guardar nuestros datos, y que puedan estar disponibles desde cualquier sistema operativo.

3. Ocultación de las particiones no utilizadas:

Dado que tenemos que instalar diversos sistemas Windows, vamos a dejar como única partición visible aquella donde vayamos a instalar el siguiente sistema operativo. Esto se hace para evitar que la instalación detecte otras copias de Windows en el equipo, en cuyo caso crearía un gesto de arranque que podría llevarnos a tener problemas al trabajar con los otros sistemas operativos. Ocultando las particiones, cada instalación del sistema operativo se verá como la única disponible. En este caso, instalamos un Windows Server 2003 en la segunda partición primaria, por lo que marcamos como ocultas (Hide) todas las demás particiones excepto esta.¹

4. Instalación de Windows Server 2003:

La instalación se realiza en la segunda partición primaria. De hecho, el instalador sólo nos ofrece esta como destino posible, dado que el resto están ocultas.

5. Otros sistemas Windows:

Si debemos instalar otro sistema Windows (Windows 98 o Windows 2000), procederemos de nuevo a emplear el Paragon para ocultar la segunda partición primaria y mostrar la tercera. Instalaremos el sistema como en el caso anterior.

6. Instalación de GNU/Linux: Dado que vamos a emplear el gestor de arranque de GNU/Linux para iniciar todos los sistemas disponibles, vamos a quitar la marca de oculto a todas las particiones, para que GNU/Linux sea capaz de detectar todos los sistemas operativos de manera automática. Una vez que las particiones se muestran correctamente, instalamos GNU/Linux en las unidades lógicas, indicándole que debe montar la partición de intercambio en la unidad lógica swap que creamos al principio, y el sistema raíz (/) en la unidad lógica ext2 (o ext3) que también creamos al comienzo. El instalador de GNU/Linux se encarga de formatear de manera adecuada las particiones correspondientes, y de localizar las copias de Windows existentes en el resto de particiones, configurando de manera automática el gestor de arranque GRUB para que podamos iniciar directamente en cualquiera de los sistemas instalados en el equipo.

¹Hay que notar que haciendo este proceso, cada Windows nombra su propia partición como C:, y una vez que se terminen todas las instalaciones y se muestren de nuevo todas las particiones, detectará al resto de particiones con otras letras, (D:, E:, F:,etc.). Sin embargo, cada vez que abramos un Windows nos aparecerá como instalado en la partición C:

En este punto, si lo deseamos podemos crear una nueva imagen del disco duro, para que reinstalar el sistema en caso de problemas sea más rápido.

1.1.3. Particiones en GNU/Linux

GNU/Linux necesita como mínimo dos particiones para poder instalarse: la partición raíz (/), donde se instala el sistema en sí, y la partición de intercambio (swap). Además, podemos crear otras particiones que faciliten el mantenimiento posterior del sistema, o que aporten mayor seguridad al mismo. De esta forma, podemos emplear otra partición para almacenar el directorio /home, donde se guardan los datos de los usuarios, y otra para /var, donde se alojan diversas utilidades a las que los usuarios pueden tener acceso.

Esta división de particiones puede ser interesante en caso de servidores, para separar los datos que se comparten de los programas y archivos del sistema operativo, o como medida de precaución, para que en caso de que llegue a ser necesario reinstalar el sistema, los usuarios no pierdan sus documentos creados con anterioridad.

Los nombres de particiones en GNU/Linux

GNU/Linux emplea el siguiente esquema para nombrar las particiones: dos letras para indicar el tipo de disco (IDE, SCSI,...), una letra para indicar el número y posición del disco, y un número para indicar la partición dentro del disco.

El número y posición de los discos IDE sigue este esquema²:

- a: canal IDE 1, disco maestro.
- b: canal IDE 1, disco esclavo.
- c: canal IDE 2, disco maestro.
- d: canal IDE 2, disco esclavo.

Por ejemplo, los discos duros IDE se denominan con las letras “hd”, el disco maestro del primer canal con la letra “a”, y la primera partición con un 1, con lo que en ese caso concreto el nombre sería “hda1”.

De manera similar, la sexta partición del disco IDE esclavo del canal 2 se denominaría “hdd6”

²Si tu equipo tiene instalada una tarjeta RAID IDE (con lo que puede tener más de dos canales IDE) esta denominación puede cambiar, y el fabricante de la tarjeta te indicará en su documentación como detecta GNU/Linux dichos discos. Normalmente, los considera discos SCSI (ver más adelante).

Un disco duro SCSI se denomina con el código “sd”, una tercera letra que indica el número de disco³, y un número que indica la partición de ese disco, de manera que “sda2” sería la segunda partición del primer disco SCSI, y “sdc4” sería la cuarta partición del tercer disco SCSI.

El caso de las unidades extendidas es especial, ya que no guardan datos (los guardan las unidades lógicas dentro de las extendidas), pero sí ocupan un número. Como ejemplo, las particiones en uno de mis sistemas (similar a las que hay en los equipos de la clase) se pueden ver en el cuadro 1.1. Tener en cuenta esto cuando más adelante se vayan a formatear, crear o montar particiones: ¡No se puede montar una partición extendida!

Cuadro 1.1: Ejemplo de esquema de particiones

Unidad	Sistema de archivos (según fdisk)	Tipo de partición	Uso real
hda1	FAT16	Primaria	MS-DOS
hda2	HPFS/NTFS	Primaria	WindowsXP
hda3	W95 FAT32 (LBA)	Primaria	Windows 98
hda4	W95 Ext'd (LBA)	Extendida	Partición extendida
hda5	Linux Swap / Solaris	Lógica	Swap de GNU/Linux
hda6	Linux	Lógica	GNU/Linux (/)
hda7	W95 FAT32	Lógica	Intercambio de datos

Respecto a la numeración, (hda1, hdc6...), los números de 1 al 4 están reservados para las 4 primeras particiones (primarias o extendidas), mientras que las unidades lógicas comienzan siempre a partir del 5. Por lo tanto, si tuvieras un disco IDE en el maestro del primer canal, con una partición primaria, una extendida, y dos unidades lógicas dentro de la extendida, sus nombres serían:

- hda1: primera partición primaria.
- hda2: partición extendida.
- hda5: primera unidad lógica.
- hda6: segunda unidad lógica.

Como se puede ver, no existe ninguna partición hda3, ni hda4. Simplemente sus nombres se quedan reservados, por si en algún momento se crean nuevas unidades primarias. Ten en cuenta que nadie ha dicho que se tenga que emplear todo el espacio de disco al formatear, puede que hayas dejado un hueco libre entre el final de hda1 y el comienzo de hda2 para futuras particiones.

³Ten en cuenta que los discos SCSI no se dividen en Maestro/Esclavo, sino que son una “cadena” de discos numerados del 0 al 7, donde el disco 0 se conecta directamente al puerto SCSI (de hecho, el 0 es realmente el propio controlador SCSI), el 1 se conecta al 0, el 2 al 1 y así sucesivamente. Claro está, esto es así si sigues un orden: si quieres, puedes tener un sólo disco SCSI y ponerle el número 5.

1.1.4. El gestor de arranque GRUB

GRUB es el programa encargado de arrancar los distintos sistemas operativos que tengamos instalados en el equipo. Al iniciar el ordenador, ofrece un menú al usuario desde donde escoge que sistema operativo se va a iniciar. Este programa se instala en el sector de arranque del disco duro (denominado MBR, Master Boot Record). El MBR es lo primero que se lee del disco duro cuando se inicia el ordenador, y marca el punto del disco donde está almacenado el programa encargado de lanzar el sistema operativo.

En grub, el nombre de las unidades de disco se encierra entre parentesis, y se empieza a contar desde el n° cero, tanto para los discos como para las particiones: (fd0) es el disquete, (hd0,0) es el primer disco, primera partición.

Grub arranca en tres etapas (**stages**):

- Etapa 1: se encuentra alojada en el sector de arranque del disco (MBR), se encarga de cargar el resto de etapas.
- Etapa *1_5: hay una por cada sistema de archivos que vayamos a usar. Se encarga de lanzar la etapa 2. Se encuentra en el primer sector del sistema de archivos donde esté la etapa 2, para que pueda alcanzarla
- Etapa 2: es común a todos los sistemas, arranca el menú de grub.

En el menú de grub podemos editar las líneas de arranque para hacer modificaciones antes de iniciar el arranque, pulsando 'e' sobre la línea a editar.

Mediante Grub podemos acceder como root al sistema (el *ataque monousuario*) si forzamos la opción `single`, con lo cual el sistema arranca en modo de recuperación monousuario, con privilegios de root sin tener que introducir clave de administrador. Ten en cuenta que si nuestra máquina puede arrancar desde CD, podemos hacer lo mismo iniciando el sistema desde un Live-CD de GNU/Linux (por ejemplo, un disco de instalación de Ubuntu), con lo que tendríamos acceso completo como root a los discos duros. Para protegernos de este ataque, debemos proteger la edición de los comandos de grub mediante una contraseña, y asegurarnos de que la BIOS está configurada para arrancar sólo desde el disco duro donde hemos instalado grub ⁴.

El fichero de configuración del arranque se encuentra en el archivo `/boot/grub/menu.lst`

En este archivo es donde podemos configurar la opción para pedir una clave al intentar editar los comandos de inicio. Esto se hace almacenando una password encriptada en formato md5 en el archivo de configuración, añadiendo la línea:

```
password -md5 CLAVE_MD5
```

⁴En realidad, si un atacante logra acceso físico a nuestra máquina, no hay mucho que hacer, porque nada le impide resetear la BIOS, sacar el disco duro (o incluso poner otro). Todo equipo *importante* debe estar bajo llave, en habitación cerrada. Y si puede estar sin conectar a la red y apagado, mucho mejor.

Dicha clave la podemos generar desde el propio grub (con la orden `md5crypt`), o en un terminal de GNU/Linux con la orden `md5`. En este ejemplo, la clave es *hola* y la redirijo desde un `echo` hasta el comando `md5` mediante un pipe (`|`). La salida del comando es la palabra *hola* encriptada en `md5`.

```
echo hola | md5
```

En el archivo `menu.lst` se definen los distintos sistemas operativos que se pueden arrancar. En función del sistema operativo a iniciar, empleamos distintas opciones.

Para arrancar un GNU/Linux necesitamos:

- `root (hd0,1)` Localización del disco/partición donde esta GNU/Linux
- `kernel /boot/vmlinuz-2.6.XXX.X.X` Localización del kernel en la partición indicada en el `root` del paso anterior
- `initrd /boot/initrd.img-2.6` Localización del `initrd` correspondiente al kernel indicado en el paso anterior.
- `boot` es opcional, esta opción da la orden de arranque, pero actualmente esto es automático y no es obligatorio ponerla.

Para arrancar un windows necesitamos:

- `root (hd0,0)` Disco/partición donde esta windows
- `makeactive` Hace que la particion sea la activa (windows no puede arrancar desde particiones no activas)
- `chainloader+1` Entrega el control primer sector de la partición windows, para que su propio gestor de arranque siga con el proceso (el famoso `ntldr` que a veces desaparece y corrompe el inicio de los sistemas Windows.⁵)

En caso de que Grub quede dañado (por ejemplo, si hemos reinstalado un Windows o se ha borrado el MBR), podemos recuperarlo. Para esto, iniciamos el sistema con un Live-CD o empleamos la utilidad `supergrub` (que realiza el proceso de manera automática). Para hacerlo a mano, necesitamos acceder como `root` a una consola, e iniciar grub. Sólo tenemos que llevar a cabo 3 pasos: localizamos la partición que contiene los archivos de la etapa 1(`find`), configuramos esta partición como raíz (`root`), y realizamos la instalación en el `mbr` (`setup`) del disco. Por ejemplo, una instalación en la segunda unidad lógica del primer disco:

⁵Por cierto, si ocurre esto, muchas veces basta con copiar de nuevo el fichero `ntldr.com` desde un CD de instalación de XP. Guardando el archivo en la raíz de la partición de Windows tenemos muchas posibilidades de recuperar el sistema sin tener que reinstalar.

```
root@equipo:~# grub
GRUB> find /boot/grub/stage1
```

```
(hd0,6)
```

```
GRUB> root (hd0,6)
```

```
GRUB setup (hd0)
```

Para localizar y comprobar los números que ocupan las particiones en grub, usamos el comando `geometry`, indicando el disco del que queremos obtener la información:

```
geometry (hd0)
```

1.2. Sistemas de ficheros

El sistema de ficheros gestiona la organización de los datos en el disco, es la estructura con la que se guardan los datos. Un sistema de ficheros está ligado al tipo de formato que asignamos a la partición: cada tipo (FAT, NTFS; EXT...) organiza los datos de forma distinta. Cuando un sistema se formatea, se crean las estructuras necesarias (índices, listas de ficheros...) en el disco para que el sistema pueda reconocer y localizar los datos que luego se ubiquen en él.

La unidad básica del sistema de ficheros GNU/Linux (EXT) es el bloque. Un disco se divide en distintos bloques, y cada bloque se divide en partes:

- La primera se llama boot (contiene el código necesario para arrancar el sistema);
- Tras ella viene el superbloque: contiene información de todo el disco.
- Después viene una lista de inodos, y tras ella los bloques de datos en sí.

1.2.1. Características:

- Estructura de directorios jerárquica: forma de árbol invertido con una sola raíz (/), aunque existan múltiples dispositivos, todos cuelgan de esa raíz.
- Provee una gran consistencia de datos, lo que implica gran seguridad (respecto a la supervivencia de los datos ante desastres físicos) a la hora de guardar los datos: se hacen varias copias del superbloque a lo largo del bloque de datos, para que si sufre alguna alteración se pueda recuperar una de las copias desde la zona de datos.
- Es una estructura dinámica: permite la creación, modificación y eliminación de los ficheros.

- Esta dotado de protección y seguridad en los archivos, mediante derechos o permisos para cada uno de los archivos y directorios, lo que otorga seguridad (respecto a los usuarios) y confidencialidad para los archivos
- Para GNU/Linux, todo son archivos (dispositivos, impresoras, particiones, pantalla, teclado, cdrom...)

1.2.2. Tipos de archivos

- Ficheros regulares: ficheros normales, creados por el usuario (también llamado fichero normal o fichero ordinario)
- Directorios
- Enlaces: apuntadores de otros ficheros.
 1. Enlaces duros
 2. Enlaces blandos o simbólicos
- Ficheros de dispositivo. normalmente se albergan en `/dev/`

1.2.3. superbloque

Guarda la información de todo el sistema de ficheros: tamaño completo de todo el sistema de ficheros, el tamaño asignado a cada bloque del disco... Los bloques grandes desperdician espacio en caso de ocuparlos con archivos pequeños, lo que genera fragmentación interna (se desperdicia espacio). Pero si se hace muy pequeño, en sistemas Windows se complican las operaciones de archivo (dado que se hacen en el propio disco, con la consiguiente pérdida de velocidad). En el caso de GNU/Linux, dado que trabaja con una lista de bloques que se almacena en memoria (ram, swap), no tarda tanto tiempo en hacer operaciones sobre los archivos. En el arranque del sistema se accede al superbloque, se comprueba la información que contiene, y se va a la tabla de inodos. Esa tabla se copia en memoria (entera) y se trabaja sobre esa tabla. Cuando se formatea, GNU/Linux asigna 512 bytes por defecto a cada bloque del disco.

Un i-nodo guarda toda la información sobre un archivo. Es un número, que guarda:

- Propietario y grupo
- Tipo de archivo
- Fechas (creacion, acceso, etc)
- Permisos o derechos de acceso: propietario, grupo, resto de usuarios
- Numero de enlaces

- Tamaño
- Lista de bloques (donde está ubicado el archivo):
 - Bloques directos: hasta 20 de ellos, son posiciones directas a la zona de datos.
 - Un bloque indirecto (normal). apunta a un bloque que contiene direcciones donde se almacenan las direcciones de los datos.
 - Un bloque indirecto doble: apunta a un bloque que contiene la dir de dos bloques, cada uno de ellos contiene información sobre la posición de la zona de datos donde se almacena
 - Un bloque indirecto triple: apunta a un bloque que almacena dir de un bloque que almacena dir de un bloque con la dir del dato. se hace para que el fichero pueda tener cualquier tamaño. (Tres niveles de almacenamiento de dirección, permite extender el archivo a cualquier tamaño).

para ver los inodos de un archivo, usamos `ls`

```
ls -lia
```

las opciones del comando que se han usado son:

l: formato largo; i: inodos; a: ficheros ocultos; Esto nos da como primer dato el inodo, luego los permisos, luego el número de enlaces, el propietario y el grupo, tamaño, fechas, horas, nombre del archivo.

para ver sólo los inodos,

```
ls -i
```

El tipo de archivo viene dado por el primer carácter de conjunto de permisos: - significa fichero regular, d significa directorio, c dispositivo carácter, p dispositivo impresora.

Los nombres de los ficheros en GNU/Linux pueden emplear cualquier carácter excepto /, hasta un máximo de 256 caracteres. Se recomienda no usar caracteres que la shell pueda confundir con comandos de control. Por ejemplo, no se debe usar <, >, :, \$...

GNU/Linux no entiende la extensión de ficheros: esta extensión sólo sirve para que el usuario tenga una idea del tipo de archivo, pero al SO le da igual que tenga extensión (o incluso que tenga múltiples extensiones).

Los ficheros ocultos en GNU/Linux son aquellos cuyo primer carácter es un punto: .oculto

Sólo aparecen con `ls -a`

1.2.4. tipos de sistema de ficheros

Hoy en día se usa ext3 (una mejora de ext2) que proporciona más seguridad, y una más rápida recuperación ante un desastre.

GNU/Linux soporta multiples sistemas de ficheros: msdos (fat16), vfat (fat32), ntfs (soporte parcial, escritura no recomendada), iso9660 (cdrom), hpfs (OS/2), nfs, sysv (unix V, Xenix), minix, proc (sistema de archivos virtual: lista de procesos en uso en el sistema)

Como información extra y curiosidad, vamos a ver a grosso modo las diferencias entre los 4 principales sistemas de archivo que se utilizan en GNU/Linux:

- ext2: GNU/Linux EXTended 2, es un sistema basado en i-nodos, con un límite de 2TB para un archivo y 4TB para una partición. Es de los más rápidos en la transferencia de archivos (porque no hace comprobaciones), pero si tienes un disco muy grande en ext2 y se va la luz, cuando vuelvas a encender el equipo tendrás que esperar varios minutos haciendo comprobaciones de inodos para evitar perder datos.
- ext3: es un sistema ext2 con journalling (de hecho, puede ser montado y usado como un ext2, ignorando el journalling), lo que quiere decir que se guarda un “diario” de las operaciones que se llevan a cabo en el sistema de ficheros. Soporta cuotas de disco (indicas cuanto espacio puede usar cada usuario como máximo). Si se va la luz, se consulta el diario, y la recuperación es muy rápida. Dada la compatibilidad con ext2, se puede actualizar del 2 al 3 sin tener que formatear ni perder archivos. El tamaño de archivo es de 2TB, como en ext2, pero un volumen puede llegar a los 32TB.
- ReiserFS: sistema moderno que proporciona journalling, aumento del tamaño en caliente (puedes añadir más discos para incrementar el tamaño sin tener que desmontar primero el sistema de archivos ni parar la máquina, aunque para esto se requiere hardware especial o discos externos), sistemas anti-fragmentación automáticos, y se puede pasar de ext3 a ReiserFS automáticamente (los usuarios de ext2 primero tienen que pasar a ext3, debido al journalling). Tiene algunas desventajas, como por ejemplo su poca resistencia ante la regeneración del árbol ⁶, no se puede realizar una defragmentación manual más allá de la que presenta el sistema automático (sólo se puede defragmentar al 100% volcando en otro disco los datos). El tamaño máximo de ficheros es de 8TB, y para una sistema de ficheros completo 16TB. Hay versiones especiales que llegan a soportar 1EB(Exa-byte) por archivo, es decir 1.000.000TB en un único fichero⁷. Se usa en bases de datos muy grandes, como la que tiene el WDC (Centro Mundial de Datos para el Clima) que

⁶Si el árbol de directorios se corrompe, puede que al intentar repararlo se acabe por destruir del todo, dado que la manera de almacenar los datos en el disco confunde a los programas de reparación, que no diferencian la información del propio sistema de ficheros (los inodos) de la información real almacenada en el disco.

⁷Teniendo en cuenta que en 2003 toda la información de Internet junta ocupaba 533.000 terabytes, aún nos sobraba espacio en el archivo. Lástima que actualmente (2007) se estime esa cifra en 12.000.000 de terabytes, con un incremento anual de 400.000TB sólo en correo electrónico, y un aumento sostenido del 30% en el total de información cada año. ¡¡¡Y estas cifras no tienen en cuenta el vídeo por internet, como YouTube, ni las redes P2P!!! Me temo que tendremos que usar más de un archivo.

ocupa casi 220 terabytes, junto con otros 6 petabytes de datos sueltos. Lógicamente, las copias de seguridad son infernales.

- XFS: eXtended FileSystem, sistema desarrollado por SGI ⁸, donado a la comunidad GNU/Linux. Soporta sistemas de archivo de hasta 9Exabytes (tú echa cuentas...), con un journaling de alto rendimiento y tolerancia a errores, cuya reparación no depende del tamaño del sistema de archivos (tarda lo mismo, ocupe lo que ocupe). Soporta cuotas de disco. Tiene sistemas de seguridad mejorados, como la posibilidad de terminar las operaciones de disco incluso después de que se haya producido un bloqueo del sistema, sin necesidad de verificación por parte del SO. Se usa en entornos donde nunca se pueden perder datos, o donde es importante que toda la información se mantenga segura y encriptada, pase lo que pase. ¿Inconvenientes?: no se puede utilizar para arrancar un SO, y es el sistema que más consume de todos los comparados en cuanto a CPU, aunque las tasas de lectura son las más rápidas de todos (lo que lo hace muy útil en entornos multiprocesador, como en cálculos científicos), y se desperdicia mucho espacio al almacenar archivos pequeños. La seguridad tiene un precio.

1.2.5. estructura lógica

La estructura en forma de árbol de GNU/Linux, es decir, los nombres y ubicación de los distintos directorios que forman el sistema. Actualmente se intenta que todas las distribuciones alcancen un estándar en la localización de los directorios y archivos, el FHS (Filesystem Hierarchy Standard), se puede consultar en www.pathname.com/fhs.

(ver diagrama en los apuntes de clase)

1.2.6. montaje de sistemas de archivos: comando mount

El comando `mount` nos permite montar sistemas de ficheros en el árbol de directorios de GNU/Linux: de esta forma tenemos el acceso a los datos de una partición desde uno de los directorios del árbol. Notar que el directorio de destino tiene que existir previamente. Supongamos que tenemos un sistema de ficheros fat32 en `hda7`, y deseamos montarlo en la carpeta `/media/DatosWindows`:

```
mkdir /media/DatosWindows/  
mount -t vfat /dev/hda7 /media/DatosWindows
```

Aunque hay muchas opciones, la sintaxis básica del comando `mount` sería:

⁸Antigua Silicon Graphics Incorporated

```
mount -t [tipo de sistema de ficheros] origen destino
```

Existe un archivo, `/etc/fstab`, que permite omitir cierta información al montar un nuevo sistema de archivos: en él se guarda el dispositivo de origen y el sistema de archivos que posee, con lo que solo hay que indicar que dispositivo se desea montar, y se consulta ese archivo para obtener los datos que le faltan. En este fichero aparecen los sistemas de archivos que hay en los discos del equipo, y que han sido detectados durante la instalación.

Ese archivo contiene una línea para cada sistema de archivos que se puede montar.

La primera línea sin comentar contiene información sobre el dispositivo `/`, que es donde está montado el sistema.

Después aparecen los demás sistemas de ficheros que haya en el equipo. Cada línea contiene los siguientes campos:

- Dispositivo
- Punto de montaje
- Tipo de sistema de ficheros
- Opciones (parámetros opcionales)
- Dump (frecuencia): intervalo para copias de seguridad del sistema automáticas, con el comando `dump`. Suele estar a 1 para `/` y a 0 para el resto. Podemos ponerlo a 1 para que también se incluya un sistema de ficheros en la copia de seguridad.
- Prioridad (1, 2 ó 0): 1 indica que es el primer sistema de archivos que se monta al arrancar. `/` siempre será 1, y se pueden asignar otras particiones para que sean 1, y también se monten automáticamente en el arranque; El 0 indica que no se va a montar al iniciar; Con 2 se montará después de haber montado todas las que tienen prioridad 1.

Por ejemplo: `/dev/hdc /media/cdrom0 udf,iso9660 user,auto 0 0`

Los parámetros pueden ser:

- `auto`: monta el sistema de archivos automáticamente al iniciar
- `noauto`: no monta automáticamente
- `user`: permite que los usuarios monten el sistema de archivos
- `users`: hace que no se puedan desmontar
- `UID`: especifica a que usuario se le aplica la opción
- `GID`: especifica el grupo a la que se permite la opción

- r: montar como solo lectura
- w: montar como escritura
- rw: montar como lectura / escritura
- ro: readonly, lo mismo que r

1.3. Inicio del sistema GNU/Linux

Los programas que se están ejecutando en un sistema GNU/Linux se denominan procesos, y cada uno de ellos se identifica con un nº (PID). En la mayoría de los sistemas GNU/Linux se emplea el método de arranque `init`. El arranque del sistema lo inicia el programa `init`, que en el momento de comenzar a ejecutarse pasa a ser considerado un proceso, y que al ser el primer proceso del sistema tiene siempre una PID de 1.

La configuración de `init` viene en el fichero `/etc/inittab`. Ese es uno de los más importantes, ya que contiene toda la información para que se llegue a ejecutar `init`. El fichero está dividido en una serie de entradas de la forma: `id:niveles_de_ejecucion:accion:proceso`

La `id` es el número entre 0 y 9999 (sólo actúa como número de secuencia). Los niveles de ejecución indican los niveles para los que va a ponerse en marcha el proceso indicado (si quiero que un proceso se inicie en los niveles 3 y 5, aparecen los números 3 y 5 en esa línea), La acción indica que se hace con el proceso.

El script que se ejecuta al principio es:

```
/etc/rc.d/rc.sysinit
```

en el directorio `/etc/rc.d/` hay muchos subdirectorios, uno por cada nivel de ejecución (`rc0.d`, `rc1.d`), y dentro hay enlaces simbólicos a los scripts que hay dentro de `/etc/rc.d/`. Los scripts lanzan los procesos que deseamos arrancar.

Los enlaces comienzan por la letra `S` o la letra `K`. `S` significa `Start`. El nombre del enlace es una `S`, seguida de un código numérico secuencial (10, 11,...) y alguna descripción. Ese script apunta al fichero que tiene que lanzarse para iniciar el proceso. Los que comienzan por `K` (`Kill`) son para detener un proceso. Funciona de la misma manera que los scripts de `Start`.

Los números indican la secuencia en la que se van a ejecutar. En ocasiones es importante, porque hay procesos que dependen de otros para poder funcionar bien.

Los scripts están todos en el directorio `rc.d`, con una carpeta para cada nivel, pero no tenemos que duplicar los scripts en cada una de las carpetas, sólo tengo que hacerles enlaces simbólicos. De esa forma, no tenemos scripts repetidos, y para modificar un script sólo hay que hacerlo una vez.

Todos los scripts son lanzados por el script `rc`, que es el lanzador de procesos.

Las acciones que se pueden llevar a cabo sobre un script son:

- `Initdefault`
- `Respawncd`
- `once`
- `wait`
- `boot`
- `bootwait`
- `sysinit`
- `powerwait`
- `powerfail`
- `powerokwait`
- `ctrlaltdel`

Los niveles de ejecución que existen en GNU/Linux son:

- 0 Detener sistema (este nivel de ejecución detiene los procesos y dispositivos, y apaga el sistema)
- 1 Monousuario sin red (single, como usuario root, sin conexiones de red, sólo línea de comandos)
- 2 Multiusuario sin red (se permite acceder como cualquier usuario, sin conexiones de red, línea de comandos)
- 3 Monousuario completo (single, como usuario root, con red, línea de comandos)
- 4 No se usa
- 5 Multiusuario grafico completo (Cualquier usuario, con red, con interfaz gráfica)
- 6 Reiniciar sistema (detiene los procesos y dispositivos, y reinicia el sistema)
- s Igual que el 1 (s: single)

1.3.1. Arranque en Ubuntu: Upstart

En los sistemas GNU/Linux Ubuntu, el sistema de arranque con `init` se ha sustituido por `upstart`. Básicamente el sistema es el mismo, pero el arranque ahora se basa en eventos, con lo que se mejora el inicio de los procesos (el arranque en `upstart`, en lugar de seguir un orden secuencial al iniciar los procesos, los intenta cargar en paralelo, para acelerar el proceso). Ver apuntes del tema 15, inicio de GNU/Linux.

1.4. Eliminar clave de root del sistema

Arrancamos con un Live-CD, y localizo donde está el sistema de archivos que se utiliza.

Lo monto como lectura y escritura (`rm`).

Ahora cambiamos la contraseña de root:

1. Necesito ser root (del live-cd), luego hacemos `su -l`. (o `sudo su`, en ubuntu).
2. Cambiamos nuestra raíz (/) al dispositivo de disco donde tengamos instalado el GNU/Linux al cual queremos acceder: `chroot /dispositivo`. Con esto, logramos conectarnos como si fuéramos el root original del sistema, y tenemos acceso completo al mismo.
3. Cambiamos la clave de superusuario con el comando `passwd`.
4. Salimos del `chroot` con el comando `exit`, y reiniciamos el equipo, quitando el live-cd. Cuando se inicie el GNU/Linux instalado en el disco duro, el usuario root tendrá la nueva clave que hemos indicado.

1.5. Editores de texto

Vamos a ver los distintos editores que tenemos en GNU/Linux.

1.5.1. VI

Comenzamos con el editor `vi`, ya que este es el editor que vamos a encontrar siempre en cualquier distribución `*nix`, estando disponible incluso en el arranque del sistema. El editor VI se basa en comandos. actualmente todos los editores VI se han sustituido por VIM (Vi IMproved, *Vi mejorado*), pero sigue respondiendo al mismo comando VI.

Ventajas:

- Incorporado en todos los sistemas Unix
- Potente
- Rapido
- Reducido
- No necesita interfaz gráfica

Inconvenientes:

- Manejo difícil de aprender
- Aridez (no es intuitivo)
- ??

Para iniciar vi, podemos indicar el archivo a usar tras el comando:

```
vi archivo.sh
```

Modos de trabajo

Vi tiene dos modos de trabajo: Comando y edición.

- Comando: permite introducir comandos o instrucciones
- Edición: permite editar e insertar texto

Pasamos de un modo a otro pulsando teclas. Para ir del modo comando al modo edición, podemos usar varias teclas (i, a, ...) Para salir de salir del modo edición y volver al modo comando, pulsamos ESC.

Siempre iniciamos en modo de comando.

Los comandos disponibles son: ⁹

- i Inserta antes del cursor
- a Inserta despues del cursor
- A Inserta al final de la linea

⁹se puede encontrar una lista bastante completa en: <http://club.telepolis.com/jagar1/Unix/Vi.htm>

- o Nueva línea después del cursor
- O Nueva línea antes del cursor
- rx Reemplaza el carácter del cursor por x (PE, rs, rt, re)... Se puede ampliar el número de caracteres indicando el número antes de la r: 3rW me cambia tres caracteres por W
- k Mueve el cursor arriba
- d Abajo
- l Derecha
- h Izquierda
- ctrl+d Avanza media página (down)
- ctrl+u Retrocede media página (up)
- ctrl+f Avanza una página (forward)
- ctrl+b Retrocede una página (backward)
- :nn Ir a la línea n
- :w [nombre] Guarda el fichero, creándolo
- :w Guarda el fichero (el fichero ya existe y tiene nombre)
- :e Carga un fichero
- :q Sale, si el fichero está guardado y no hay nuevos cambios.
- :q! Sale, sin guardar
- :wq Guarda y sale
- :x Igual que :wq
- x Borra el carácter sobre el que está el cursor, indicando un n° antes de la x, se borra ese n° de caracteres (5x borra 5 caracteres)
- dw Borra una palabra, el n° delante indica el n° de palabras (5dw, 5 palabras)
- dd Borra línea, el n° delante indica el n° de líneas (5dd, 5 líneas)
- D Borra desde la posición actual del cursor hasta el final de línea
- <rango>d Borra un rango de líneas: 2,6d: desde la línea 2 a la 6. Podemos emplear algunos caracteres especiales: el carácter “.” representa la línea actual, el \$ el final del documento. Por ejemplo:

1,\$d

·,-3d

- Y Copia la línea
- P Pega
- :rango;y Copia las líneas del rango
- :pu Inserta el contenido (reemplaza lo que haya)
- :nnpu Inserta despues de la línea nn
- :v Sólo en VIM, pasa a modo visual, podemos seleccionar texto con el raton.
- :/cadena Busca cadena hacia adelante
- :?cadena Busca hacia atras
- :/ Repite ultima busqueda
- :;rang;s/cad1/cad2 Reemplaza la 1ª aparición de cad1 por cad2 en la linea actual
- :;rang;s/cad1/cad2/g Reemplaza todas las apariciones de cad1 en la linea actual
- :;rang;s/cad1/cad2/G Reemplaza todas las apariciones en el documento
- := Cuenta el número de lineas desde el cursor hasta el final del fichero
- := Cuenta el numero de lineas del fichero
- ctrl+g Muestra el número de lineas actual y total de lineas del fichero

Ejemplos:

:-5y copia las 5 lineas anteriores a la linea actual

:12pu pega en la linea 12

:1,\$s/hola/adios/g

reemplaza (con rango) todas las ocurrencias de hola por adios en cada linea, desde el comienzo del fichero hasta el final. Equivale a:

:s/hla/adios/G

`:g/^$/d` elimina las líneas en blanco

con los comandos `set` podemos configurar:

- `:Set [no]number` muestra / oculta números de línea
- `:Set [no]autoindent` indentación de líneas

Para editar la configuración de Vim, usamos el archivo `vimrc`, solo que está oculto, por lo tanto comienza por un punto (`.vimrc`). Hay que guardarlo en la carpeta `/home` del usuario (`/.vimrc`), o si queremos que aplique la configuración a todos los usuarios, eliminar las copias que pueda haber en la carpeta de usuario y modificar el archivo `/etc/vim/vimrc`

`~` representa el directorio personal (`/home/usuario/`)

1.5.2. Nano

Nano es un editor de texto sencillo, que para las tareas básicas puede resultar mucho más sencillo que VI. Más adelante añadiremos algo sobre él.

1.6. Uso de comandos

1.6.1. Ayuda en GNU/Linux

`man et all`

1.6.2. Comandos básicos

`ls -l` largo, `-l` formato humano (gigas, megas); `-a` ocultos; `-s` orden alfabético; `-i` n° de inodo...

`cd ~` directorio personal del usuario

`pwd` ruta hasta el directorio actual

`mkdir` crea directorios, con `-p` crea los intermedios que aparezcan y no existan

`rmdir` borra directorios (vacíos), con `-p` borra los de toda la ruta.

`mv` mueve o renombra. `-i` interactivo; `-u` (upgrade), actualiza el destino, sólo si el fuente es más moderno que el destino.

cp copiar. -R recursivo; -i interactiva; -l crea enlaces fuertes en lugar de copiar; acepta comodines.

rm borrar. -r|-R borra recursivo; -f fuerza el borrado; -i interactivo.

lpr imprime. -E encripta la conexión, -P destino: imprime en la impresora especificada; -# copias: establece el nº de copias (1-100).

ln enlazar. Sin opciones crea un enlace duro; -s crea un enlace blando (simbólico).

cat imprime en salida estándar (conCATenate); si no se le pasa argumento, lee de entrada estándar; si solo se le da origen (PE un archivo) lo saca en pantalla. Si solo se le da salida, lee de entrada estándar y guarda en salida. PE *cat > fichero.txt* lee desde teclado y guarda en fichero.txt; *cat fichero.txt* lo muestra en pantalla. para que deje de leer desde entrada estándar, *ctrl + d*.

PE: *cat < origen > destino*: origen explícito, pero se puede omitir el <

more muestra pantalla por pantalla. Se usa en conjunción con el pipe |, para redirigir la salida al more. PE *cat archivo | more*. More no permite el uso de cursores, y solo se puede avanzar en pantalla, no retroceder.

less como more, más potente, permite desplazamiento

1.6.3. Expresiones Regulares

Una expresión regular es un patrón de caracteres que se usa para representar a dichos caracteres en una operación de búsqueda.

Vienen encerradas entre barras: */exp_reg/*

las alternativas se separan con |:

/verdes|azules/;1|2|3/

Cuantificación : indican cuantos caracteres puede haber. Los más comunes son +,?,*

+ indica que el caracter que le precede puede aparecer una o más veces:

/eco+lógico/

busca eco, seguido de todas las o que sean, seguido de lógico: ecologico, ecoooloico...

? Indica que el caracter que le precede puede aparecer una vez o ninguna:

/ob?scuro/

coincide con oscuro y obscuro

* el caracter que le precede puede aparecer 0, 1, ó más veces.

Agrupación : usamos paréntesis para definir el ámbito y precedencia de los operadores:

```
/hij(o|a)/ equivale a /hijo|hija/
/(des)?cubierto/ genera descubierto y cubierto
/H(ae?|`a)ndel/
```

. cualquier caracter, pero sólo una ocurrencia del mismo.

PE, /algo./ es algo seguido de un caracter. /algo.*/ es algo seguido de cero ó más apariciones de un caracter /c.s/ en la cadena: el caso de la costilla provoco el cese, caes conmigo. Genera caso, costilla, cese, pero no "caes" porque tiene dos caracteres. Si fuera /c.*s/ si saldría.

barra invertida (no /) protege el caracter que le sigue: pe: pra buscar un punto, le pongo delante una barra invertida.

Corchetes se usan para definir conjuntos: para los numeros de 1 al 6, podemos poner [123456] ó [1-6] PE, los DNI serían

```
/[0-9]{8}([A-Z]|[a-z])/
```

Notar que usamos las llave para indicar el número de aparición de los números (8 elementos). Los metacaracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes. Si queremos que sigan siendo metacaracteres hay que protegerlos PE:

```
/[\d.]/
```

Busca dígitos con un punto decimal, el punto ya no actua como metacaracter, sino como literal. El d busca dígitos, como está protegido por la barra sigue actuando como dígito, no como letra d.

\$ Representa el final de la cadena de caracteres o de la línera, si se utiliza el modo multilínea. o representa un caracter en concreto, sino una posición:

```
/\.$/
```

Busca las líneas que terminan en .

```
/[0,9]$/
```

líneas que acaban en dígito

acento circunflejo representa el inicio de la cadena, pero ojo que tiene más usos, como la negación en los conjuntos dentro de los corchetes..

```
/^[a-z]/
```

líneas que comienzan por minúscula.

Entre corchetes:

```
/[^(a-z)]/
```

Cualquier caracter excepto minúsculas

```
/^\d$/
```

Se asegura que comience y termine por un dígito

llaves indican numero de repeticiones del carcarter (conjunto) que le precede. Un solo numerp inidica en numero. Si pongo entre comas, indico un rango:

```
/[0,9]{3,7}/
```

entre 3 y 7 dígitos

barra menor que: indica inicio de palabra

barra mayor que: indica final de palabra

```
/\<[A-Z].*\.\>/
```

palabra comienza en mayúsculas, sigue cualquier caracter, termina en punto

Repeticion nº de veces Repetcion del caracter x un numero entre m y n (n es opcional).

```
A\{2,5\}
```

busca entre 2 hasta 5 repeticiones de la A

1.6.4. find. Buscador de archivos

Con Find podemos localizar ficheros en el sistema, empleando para ello expresiones regulares, propietarios, grupos, fechas de acceso y modificación (de archivos, permisos, ...), tamaños. Permite búsquedas muy potentes, incluyendo quien ha hecho que, etc. Además permite que se ejecute una orden con cada archivo que encuentre (PE para renombrar, eliminar, grepear, volcarlos a cualquier otro sitio)..

Podemos buscar por:

-name expresion permite comodines *. PE:

```
find / -maxdepth 2 -name '*.conf'
```

busca todos los archivos .conf que existen desde la barra en un máximo de dos niveles de directorio

-type indica el tipo de fichero a buscar: b, bloques; c, caracter; d,directorio; f, regular; l, enlace

-size tamaño máximo o mínimo que tenga: `-size +tam` es el tamaño mínimo, y el `-size -tam` es el máximo. Para un rango de tamaño, `-size +tam -size -tam`. PE:

```
find / -maxdepth 2 -size +1024 -size -2048
entre 1024 y 2048
```

-iname indica que no diferencie mayúsculas de minúsculas.

-atime +n último acceso producido en las últimas $n \times 24$ h (n serían los días).

-mtime +n modificación en las últimas $n \times 24$ h

-ctime +n mod de permisos

-user usuario propiedad de un usuario; tb existe la opción para los que NO son de un usuario (ver man)

-group grupo prop de un grupo concreto; idem (ver man)

-perm +/-modo ficheros con los permisos concretos.

-exec comando tras comando, `{}` se refiere al archivo y sustituye la ocurrencia de las llaves por el nombre del archivo, y debe acabar con `\;` (la barra protege al `;`) PE:

```
find -name 'da*' -exec echo Archivo encontrado es {} \;
```

busca todos los archivos que comienzan por da y escribe una línea que indica 'archivo encontrado' y pone la ruta al mismo. Podemos usar varias llaves `{}` Podemos poner varios comandos:

```
find -name 'da*' -exec echo El archivo {} encontrado es {} \; -exec cat {} \;
```

Ejemplos de find:

```
find /etc -name '*.conf'
```

```
find / -size +10240k -size -20480k
```

```
find -maxdepth 1 -perm +222 -type d
```

en el directorio actual, con permisos 222, tipo directorio

1.6.5. locate

localiza un archivo. Solo para root. localiza en todo el sistema de ficheros.

```
locate nombre_de_archivo
```

para actualizar la base de datos de locate, usamos uno de estos dos comandos:

```
updatedb  
locate -u
```

1.6.6. which

busca el archivo especificado sólo dentro del directorio personal del usuario actual (reconoce al usuario de manera automática).

```
which vi examen.final
```

localiza el archivo vi y el archivo examen.final, dentro de mi carpeta personal (recursiva)

1.6.7. cat. Redirecciones y algo más

```
< redireccion de entrada
```

```
> redireccion de salida
```

```
>> redireccion de añadir
```

```
| pipe (tuberia, canalizacion) el resultado de la parte izqd se pasa como  
  entrada para el comando de la parte derecha
```

ejemplo

```
ls -l ... | egrep -v '[^a]*a' > aaas
```

para numerar las lineas:

```
cat -n
```

1.6.8. wc: word count

cuenta palabras. Tiene de opciones:

```
-l cuenta lineas
```

- c cuenta caracteres
- w cuenta palabras
- L longitud de la línea mas larga

PE ¿cuantos archivos hay en un directorio?

```
ls -l|wc -l
```

1.6.9. grep

Globally Regular Expressions Pattern. Busca las líneas que contienen una coincidencia con la expresión regular iniciada.

Hay tres posibilidades:

- grep: algunas expresiones regulares no las entiende
- egrep: entiende todas las expresiones regulares. (Extended Grep)
- fgrep: más encaminado a búsquedas en determinados ficheros. Tb entiende todas las regexp.

```
grep [opciones] <patron> [ficheros] opciones
```

-c (count) devuelve sólo el nº de líneas coincidentes

-i no diferencia mayúsculas de minúsculas

-H muestras además de las líneas, el nombre del fichero. Si estamos buscando en más de un fichero, se pone como opción por defecto.

-l Cuando se busca en más de un fichero, sólo muestra los nombres de aquellos donde se ha encontrado el patrón, pero no muestra las coincidencias

-v devuelve las líneas que no coinciden

-r recursivo

-n imprime el número de línea donde se ha encontrado la coincidencia

Ejemplos:

```
$ grep -i '^s' /etc/passwd
```

busca las líneas que comienzan por s ó S en /etc/passwd. Es equivalente al comando:

```
$ cat /etc/passwd|grep -i '^s'
```

```
$grep linux /usr/share/doc
```

```
$grep root /et/passwd
```

```
#grep -n error var/log/messages
```

```
$grep -i antonio /etc/passwd
```

```
$grep -c root /etc/group
```

```
$grep -l -r -i img /var/www/html/
```

```
$ls -lia | grep "examen GNU/Linux"
```

1.6.10. sed

editor de flujo de texto. Tiene de entrada un fichero, lo lee, y lo analiza, mostrando la salida en pantalla o en otro archivo (mediante redirección). Nunca toca los archivos originales. Podemos predefinir un archivo de ordenes para que cada vez que se llama a set se ejecute con esas opciones en concreto de manera automática (una especie de script, no muy potente, pero bastante usable).

```
sed [-n] [-e'script'] [-f archivo] archivo1 ... opciones
```

por defecto, sed muestra las líneas que procesa y la línea resultado del proceso, con lo que a veces dificulta entender el resultado. podemos modificarlo con las opciones:

-n no muestra la línea que se está procesando

-e indica que se ejecute el script indicado

-f indica el archivo donde se encuentran los comandos a ejecutar

En resumen, las opciones más usuales de sed son:

```
sed -n [-e|-f]
```

-e inidica que a continuación, entre comillas simples, le vamos a dar un script directamente en la línea de comandos.

-f indica un fichero de comandos a ejecutar.

PE:

```
sed -n -e '2p' ficherosed
ejecuta la orden 2p
```

Si creo un fichero `ordensed` que contenga la línea 2p:

```
sed -n -f ordensed ficherosed
haría la orden 2p
```

La estructura de una orden de `sed` es:

```
[inicio[,fin]] función [argumentos]
```

`inicio` y `fin`: hace referencia a las líneas (nº de líneas) afectadas, (o intervalo de líneas). Si no pongo nada, se aplica a todas las líneas del documento. Si pongo un nº, indica la línea en concreto. Si pongo \$, indica la última línea. Si aparece una expresión regular, se aplica a las líneas que concuerdan con la `regexp`. Solo admite un intervalo.

`función`: indica el comando que debemos aplicar

`argumentos`: indica los argumentos necesarios para la función

Funciones posibles que se le pueden dar:

`p`: mostrar la línea que se está procesando

`d`: eliminar la línea que se está procesando

`s`: sustituye lo que concuerda con el patrón por una cadena, siguiendo esta sintaxis:

```
s /regexp/cadena/modificadores
```

Ejemplos de `sed` y sus órdenes:

```
$ sed 2p fichero_sed
```

procesa la segunda y las muestra todas. `p` es una orden (por eso no lleva -)

```
$ sed -n 2p fichero_sed
```

procesa la segunda y no muestra las que no procesa. `-n` es una opción, necesita el -

```
$ sed -n 2,4p ficherosed
```

desde la 2 hasta la 4, muestra solo esas

```
$sed -n 2,\$p ficherosed
```

procesa hasta el final del archivo (el \$). hay que proteger el \$ con una barra \

```
$ls -l | sed 's/--/KK/g'
```

en un `ls -l`, sustituye las dos rayas de los permisos que no estan definidos por KK. Con `/g`, se aplica a todas las apariciones en cada una de las lineas. Si no pongo `/g`, solo se hace en la primera ocurrencia en cada linea. Pero ojo, en todas las lineas!!!

```
$sed -n /^F/p ficherosed
```

las que comienzan por F mayuscula (entre `/.../` porque es regexp); la p imprime, igual que en los ejemplos de arriba

```
$sed -n '/^\$/p ficherosed
```

entre comillas simples porque lleva caracteres especiales, y si no se ponen interpreta el \$ mal. Muestra las lineas que comienzan por \$

```
$sed -n s'/5$/8/'p ficherosed
```

las lineas que acaban en 5, les cambia un 5 por un 8. La sustitución la indica la s. Si quito el \$, sería al primer 5. Si añado `/g` y quito el \$, a todas las ocurrencias del 5.

1.6.11. cut

OJO para hacerlo funcionar en MAC OSX, hay que combinarlo con “tr”

corta campos de una salida. Con esto, podemos extraer columnas de un ls, o quedarnos con un solo campo de un fichero.

`-c` corta columnas (carácter por carácter).

`-f` corta campos (indicando el nº del campo, y por qué están delimitados, mediante `-d`)

`-d` especifica un delimitador

```
cut -f9 -d' ' algo >>> el campo nueve, delimitado por espacios.
```

podemos especificar intervalos de columnas o campos;

`m-n`: campos o columnas entre m y n inclusive.

`m-`: campos o columnas desde m hasta el final

`m,n`: campos o columnas m y n (sin intervalo). Puedes poner varias.

1.6.12. tr

traduce, se usa para convertir un caracter en otro. PE mayusculas a minusculas, espacios a otro tipo de caracteres.

```
-d elimina caracteres que coinciden con el patrón
-s elimina repetidos
-c [rango] todo carácter que no este en un conjunto
```

PE:

```
ls -l | tr -s ' ' '#' cambia todos los espacios (y repeticiones) por #
```

PE en OSX, para que funcione bien el cut:

```
ls -l |tr -s ' ' "#" |cut -f5 -d '#'
```

para sacar la fecha/hora de mod de un archivo:

cambio los espacios por #, hago el cut, vuelvo a cambiar # por espacios

```
ls -l |tr -s ' ' "#" |cut -f6-8 -d '#' |tr -s '#' ' ' '
```

para pasar de mayusculas a minusculas:

```
tr [A-Z] [a-z]
```

Para quitar todas las mayusculas y poner en su lugar *:

```
tr [A-Z] '*'
```

1.6.13. tee

Bifurca la salida, de manera que se muestre en pantalla, y además se almacene en un fichero.

```
ls -l /etc |tee directc
```

me muestra en pantalla la salida del ls -l y ademas me la guarda en el archivo directc.

Tiene como opción el parámetro -a, que indica que añada al fichero, en lugar de sobrescribirlo.

1.6.14. pr

formatea la salida, se usa para formatear antes de mandar a la impresora (divide en pags, añade encabezados y pie, tipo de espacios...).

1.6.15. lp

envia a la impresora

```
ls -l / | lp
```

1.6.16. file

identifica el tipo de fichero. file archivo

1.6.17. sort

ordena un fichero por orden lexicográfico:

sort a secas ordena la salida por la primera columna que aparezca.

- -r ordena en orden inverso . Default es menor a mayor.
- -f non case sensitive
- -n ordenación numérica, para que cuando ordenemos 111 y 20, el 20 esté delante del 111 (si no indicamos - n, hace orden léxico, donde 1 va antes que 2)
- -k c[,d]numero del campo de ordenación, desde la c hasta la d. Hay otra opción que permite indicar subcampos de ordenación (pe, que ordene primero con la 4, luego con la 3, luego con la 1, luego con la 7). Buscar esta opción en los manuales.
- -t sep establece el carácter separador de campos
- -m mezcla archivos: sort -m fi1 fi2
- -u líneas únicas, no aparecen líneas repetidas más de una vez.
- otras...

1.6.18. uniq

elimina líneas repetidas.

-c cuenta, poniendo como prefijo cuantas repeticiones hay -d solo imprime las líneas duplicadas

1.6.19. tail head

ya se han visto, ver mas arriba

1.6.20. stat

muestra informacion de un fichero. en OSX, con -x

1.6.21. touch

si no existe, crea un archivo. Si existe, le pone la fecha actual, o la que le indique directamente, o toma la fecha de un archivo -a cambia fecha acceso -c no crea fichero -m cambia fecha modificacion -r archivo referencia -t fecha, en formato [[SS]AA]MMDDhhmm[.ss] (SS son centurias, 90, 00, 20...). Centuria, año, segundos son opcionales.

1.6.22. gzip y gunzip

comprime y descomprimen uno o varios ficheros.

-r recursivo -1 ... -9 tipo de compresion (1 rapida, 9 mejor compresion) -S permite añadir una extensión al fichero resultante. (por defecto es .gz)

1.6.23. bzip y bunzip

igual que gzip y gunzip, crean archivos bz2, mas pequeños que los gz.

1.7. Comandos avanzados

1.7.1. zip y unzip

formato zip. Compatible con ficheros zip de windows.

1.7.2. tar

no comprime, agrupa un conjunto de archivos en uno solo (se le dice "tarear"). Luego se puede comprimir. Por eso, se usa mucho para backups, con tar y gzip.

1.8. Otros comandos útiles

Más comandos que pueden venir bien alguna vez¹⁰:

1.8.1. `df`

con `df` podemos ver el espacio libre de las particiones, o el tamaño total de un directorio.

```
df -h
```

nos muestra información de las particiones (dispositivo, tamaño, espacio libre y usado, donde está montado...)

si además añadimos un directorio:

```
df -h /home/user
```

nos dirá cuanto ocupa el directorio. Ojo, que parece que nos está dando datos sobre la partición, pero en la columna *Used* el espacio que muestra es el que ocupa el directorio (con todos sus subdirectorios), no el de la partición.

Más importante aún, es no confundir el comando *df* con *fd*, que es el dispositivo de disquetes en GNU/Linux.

1.8.2. `nano`

1.8.3. `file-roller`

1.8.4. `gnome-commander`

1.8.5. `apt-get`

1.8.6. `aptitude`

¹⁰Y que no han sido dados en clase, así que no creo que necesites estudiarlos

Capítulo 2

Administración del sistema. Ficheros en red

2.1. Administración de usuarios y grupos

2.1.1. Tabla de usuarios

Los usuarios del sistema se listan en el archivo `/etc/passwd`, uno por línea, en campos separados por “:” de la siguiente forma:

```
nombre:clave_cifrada:UID:GID:descripción_del_usuario:home:shell
```

Nombre es el nombre de usuario para logearse en el sistema.

En el segundo campo puede aparecer la clave encriptada directamente, o una `x`, indicando que la clave está almacenada en otro archivo, el `etc/shadow`¹.

Los UID (User ID) de usuarios reales son a partir del número 1000. 0 es root, y entre 2 y 999 para cuentas del sistema (`www`, `games`, ...).

El GID (Group ID) identifica al grupo primario. Un usuario puede pertenecer a muchos más grupos suplementarios.

La descripción no suele ponerse, es información extra sobre el usuario (PE, departamento, cargo, localización...)

El `home` indica el directorio personal del usuario, suele ser `/home/nombre-de-usuario`

Shell, el último campo, indica la shell (consola) que se utiliza. suele ser `bash`.

¹¿Por qué en dos archivos? Porque el `/etc/passwd` es accesible para cualquier usuario, mientras que el `shadow` es de root

2.1.2. Extensión de la tabla de usuarios

Hay mas datos de usuarios, que se encuentran en `/etc/shadow`

`nombre:clave-cifrada:fechas:fechas:fechas::::fechas:`

Si la clave no estaba en el archivo `passwd`, aquí si que aparece (cifrada, también). Las fechas indican caducidad, tiempos de cambio de clave, ...

Estos campos de fecha se pueden modificar con diversos comandos, o siempre que sepamos exactamente que estamos haciendo, modificarlo con `vi` (ojo, puedes fastidiarla muy mucho).

Comandos que afectan al archivo `shadow` son: `passwd` (cambia la clave de un usuario) `chage` (modifica las fechas de los usuarios)

2.1.3. Tabla de grupos

La tabla se encuentra almacenada en

`/etc/group`

y en el archivo tenemos lineas de la forma:

`nombre:contraseña-grupo:GID:lista de usuarios`

los grupos de un usuario que son primarios se conocen como grupos suplementarios

2.1.4. Procedimientos para crear usuarios y grupos

`useradd` crea un usuario

`userdel` lo borra

`usermod` modifica los atributos

`groupadd` añade grupos

`groupdel` elimina grupos

`groupmod` modifica información sobre el grupo

`passwd` cambia contraseña a un usuario

`chage` cambia las fechas de caducidad

Si nosotros creáramos un usuario a mano, deberíamos añadirlo a `/etc/passwd`, a `/etc/shadow`, crear su directorio en `/home/usuario`, copiar el contenido de `/etc/skel` a `/home/usuario...`

Para que se haga todo de manera automática, se emplea `adduser usuario`, que nos crea todo de manera automática haciendo preguntas para completar los campos de información y crear los directorios.

2.1.5. Atributos de protección de procesos

Identificadores del propietario del proceso:

Existen 2, uno conocido como (r)UID (propietario real, quien lo lanzó), y el (e)UID (propietario efectivo, el que lo ejecuta en este momento).

También hay identificadores del grupo del propietario: (r)GID (grupo primario del propietario) y (e)GID (de la lista de grupos del usuario, cual de ellos esta ejecutando el proceso).

Cuando el proceso se crea, es cuando se le asigna la numeración, excepto en el primer proceso de todos, que se crea al logearse el usuario, que se le asigna específicamente en función del número de usuario, que tiene predeterminados unos privilegios que son los asignados al primer proceso. El resto de procesos se heredan y generan.

2.1.6. Atributos de protección de ficheros

ternas de atributos (propietario, grupo, otros) + la terna de bits especiales:

bits 11 10 9 - 8 7 6 - 5 4 3 - 2 1 0

11 10 9 son los especiales

8 7 6 son los de propietario

5 4 3 grupo

2 1 0 otros

recordar que, con ficheros y directorios, los permisos hacen cosas distintas. Si en un directorio el de x esta activado, permite usar el directorio en la ruta a un comando.

11 SETUID

10 SETGID

9 STICKY: si esta activado (1), en ficheros y directorios, únicamente lo podrá borrar su propietario, y nadie más, aunque se encuentre en una carpeta con permisos para todos.

Los especiales se heredan de directorio a directorio: si tengo un dir sticky, y creo otro dir dentro, este va a heredar sticky. OJO: Los ficheros no heredan automáticamente.

para modificar los tres bits especiales, al `chmod` hay que indicarle 4 dígitos octales: normalmente es con tres dígitos

```
chmod 666 file
```

para cambiar también los especiales, con 4 grupos:

```
chmod 7777 file
```

2.1.7. Protección básica

Cuando se lanza un proceso, GNU/Linux busca los permisos de forma secuencial, y la primera opción correcta que encuentre es donde se entra.

si (e)UID es 0, somos root, se da acceso automático.

si (e)UID == ID propietario, se da acceso

si (e)GID ó (lista de grupos suplementarios) == G propietario, se da acceso VIGILANDO los permisos de grupo (bits 5, 4, 3)

si no es ninguna, se miran los bits (2,1,0)

2.1.8. Cambio de los atributos de protección

Podemos cambiar:

- permisos (sólo pueden root, propietario)
(e)UID=0 si se permite (e)UID)=propietario si se permite
- propietario (Sólo root)
- grupo (root, y el propietario siempre que él también pertenezca al nuevo grupo)

2.1.9. Los bits SETUID y SETGID en ejecutables

cuando SETUID se activa, cuando un usuario normal [(r)UID] lo lanza, se ejecuta como si lo hubiera lanzado el propietario [(e)UID] del archivo. Es lo que pasa por ejemplo con el comando passwd, que normalmente solo puede ser lanzado por root. Es equivalente al runas (Ejecutar Como) del windows.

2.1.10. SETGID en directorios

Cuando se activa SETGID de directorios, cualquier archivo que se cree dentro tomará como identificador de propietario al propietario del directorio

2.1.11. Comando UMASK

máscara de creación de archivos y directorios. Por defecto, al crear un archivo o directorio, se le establecen una serie de permisos por defecto (definidos en `/etc/profile`). Si queremos cambiarlos para una sesión, se usa UMASK.

UMASK a secas me da 0022, el valor que tiene asignado en ese momento 6666, que resulta ser 6644, que son los permisos que le va a asignar por defecto) notar que no hace el complemento con 7777, para evitar que te cuele permisos de ejecución por la cara.

2.1.12. Grupos privados

Actualmente, cuando se crea un usuario, se le crea además un grupo para él mismo (se le asigna un nuevo GID, el nombre del grupo será el nombre del propio usuario).

2.1.13. Comandos relacionados

en el tema del moodle

2.2. Gestión de procesos

2.2.1. Introducción

un proceso es un programa en ejecución. Para que un programa se ejecute, el SO necesita asignarle un identificador, un espacio en memoria, etc. A partir de ese momento, el programa se convierte en un proceso. Cuando estamos en GNU/Linux, al crear un proceso necesitamos una serie de parametros que el SO tiene que almacenar. El primer proceso que se crea es *init*, cuyo ID es 1. A partir de ahí, todos los procesos son hijos de *init*, es decir, que *init* es el padre de todos los procesos del sistema. Se almacenan los siguientes datos de un proceso:

- Usuario que lo crea (UID)
- Hora de comienzo (para saber cuanto tiempo se lleva ejecutando)
- Comando que lo originó (más bien, la línea de comando con todos sus argumentos)
- Estado del proceso (sleep, running, zombie, stopped)
- Prioridad del proceso: permite que ese proceso se posicione en la cola de procesos respecto a los que se ejecutan concurrentemente con él. Por defecto, un nuevo proceso se crea con prioridad 10 (rango: -20 hasta 19). Lo más prioritario es -20, y sólo root puede dar prioridades por debajo de 0. La alta prioridad es exclusiva de root.

- Terminal desde la que se lanzó el proceso (si es que está relacionado con una terminal, si no lo está, no se almacena).

Comando PS

nos muestra los procesos que se están ejecutando en ese momento en la máquina. Si lo lanzamos sin parámetros, nos muestra:

- PID: ID del proceso
- TT: terminal en la cual se ejecuta
- TIME: tiempo que lleva ejecutándose
- COMMAND: orden que lanzó el proceso

Algunos modificadores de ps son:

- x: muestra todos los procesos del usuario actual
- a: todos los usuario
- f: jerarquia
- e: entorno
- l: largo
- u:usuario formato de presentacion

top

comando interactivo que muestra, en tiempo real, datos sobre los procesos. Cuando entramos en top, se queda ejecutándose hasta que salimos. podemos usar varios parámetros mientras usamos top:

- A: antigüedad
- M: cantidad de memoria
- P cpu
- N PID

Por ejemplo, podemos lanzar un proceso `sleep 1200`, pararlo con `ctrl+z`, ver con `jobs` que está ejecutándose, lanzar otro proceso en segundo plano (`sleep 300 &`), y ver con `jobs` como está el 1200 en `stopped`, y el 300 en `running`. para volver a lanzar en proceso con 1200, usamos `fg` (a secas, lanza el primero de la lista que tiene un `+`, si le indico el numero, lanza ese en concreto. De hecho, sin argumentos, lanza el primero de la lista en ORDEN DE PRIORIDAD DE LOS PROCESOS, que el más prioritario es el que tiene el `+` en la orden `jobs`).

2.2.2. Control de los procesos: `fg`, `bg`

usamos los comandos `fg` y `bg` (primer plano y segundo plano, respectivamente)

2.2.3. Comunicación con los procesos: `kill`, `killall`

Es enviarle una serie de señales. Con `kill -l` podemos ver la lista de señales que les podemos mandar: `PE`, la 9 es `sigkill` (muere ya y no protestes, mata todo lo que este relacionado contigo sin esperar a que acabe nada). 15 `sigterm` (muere, pero hazlo bien y cierra todo de manera correcta).

Un proceso que se está ejecutando se puede parar con `ctrl+c` y `ctrl+z` (con `z` se pasa a segundo plano, con `c` se mata directamente).

con `kill`:

```
kill -9 15357
```

mataría (`sigkill`) al proceso 15357. La PID la obtenemos con `ps` o `jobs`.

la diferencia entre `kill` y `killall` es que `kill` funciona con el identificador de proceso, y `killall` con el nombre de proceso. Ojo, engloba a todos los procesos con el mismo nombre (si tengo muchos procesos que se llaman igual, se mueren todos). PE: `killall -9 sleep` mata todos los procesos `sleep` que haya en el sistema.

2.2.4. Prioridades y ámbito: `nice`, `renice`, `nohup`

Con estos comandos cambiamos la prioridad de un proceso.

2.2.5. `pstree`

muestra los procesos en árbol jerárquico, si le indico el usuario los de ese usuario, si le indico el nº de proceso los que cuelgan de ese proceso.

2.3. Servidor NFS

Un servidor NFS nos permite compartir recursos con otras máquina cliente de la red. Luego podemos montar los recursos remotos con *mount* cuando los necesitemos, o haciendo que se monten de forma automática con el *fstab*.

Para montar el servidor NFS, necesitamos saber lo siguiente.

- Necesitamos comunicación TCP/IP (la red montada y los equipos configurados).
- El sistema de archivos que vamos a montar es de tipo NFS.
- Los permisos que se usan en la compartición son los de *otros*, por lo que nos tenemos que asegurar de que tenemos permisos de lectura para otros (como mínimo, los permisos adecuados serán los que necesite la máquina autorizada con mayores privilegios).
- Asegurarnos de que el cliente es capaz de montar sistemas NFS

2.3.1. Pasos a seguir (EN FEDORA):

1. Iniciar los siguientes daemons (demonios);

- `nsfd`
- `rpciod`
- `loctd`
- `rpc.mountd`
- `rpc.rquotad`

Todos los demonios se pueden iniciar con el comando:

```
/etc/init.d/nfs
```

Otra forma de lanzarlo (en lugar de con `init`), directamente desde el directorio donde está el script de inicio:

```
/etc/rc.d/rc5.d/S60nfs
```

2. Configurar el fichero `/etc/exports/`, que es donde se configuran las carpetas compartidas. La sintaxis para trabajar con ese fichero es:

```
/home pc1.bez.es (rw)  
carpeta_compartida IP/nombres_maquinas_con_conexion_al_recurso (permisos)
```

Aparte de las máquinas por IP o nombre, podemos indicar subredes:

```
192.168.5.1/255.255.255.0
```

incluso podemos utilizar el * para indicar máquinas de un mismo dominio o rangos de ip:

```
*.bez.es
192.168.1.*
```

Podemos indicar varias máquinas por línea:

```
/home pc1.bez.es (rw) pc2.bez.es (ro)
```

Las opciones disponibles para las máquinas son:

- rw: lectura y escritura
- ro: solo lectura
- sync: escritura sincronizada (por bloques, no continuo)
- secure: va a utilizar para la comunicación un puerto inferior al 1024 (del sistema).
- wdelay: retraso de escritura, para que use un almacenamiento intermedio de datos antes de llegar a la carpeta definitiva, para evitar el acceso continuo al recurso.
- root squash: los usuarios como root no pueden conectarse como tales, se conectan como usuarios anónimos (el usuario squash es anonimo).
- all squash: todos los usuarios se conectan como anónimos.
- anonuid=UID: mapea al usuario anonimo al usuario con UID. (PE anonuid=1005)
- anongid=GID: mapea al grupo anonimo al grupo con GID.

3. Configurar */etc/hosts.allow* y */etc/hosts.deny*

Definen los servicios permitidos o denegados a las máquinas indicadas. En allow estan los servicios y máquinas son permitidos. En deny, los denegados. Cada línea de estos ficheros es:

```
servicio:hosts[IP,domino,...]
```

Por ejemplo, un hosts.deny con el máximo de restricciones sería:

```
portmap:ALL
```

esto deniega el proceso de mapeo de puertos a todo el mundo.

Una vez cerrado el portmap, podemos permitir a máquinas concretas el acceso mediante hosts.allow:

```
portmap:192.168.1.3 192.168.1.4
```

4. Ejecutar *exportfs*, lo que carga de nuevo las configuraciones del nfs. Toma los ficheros *export*, *allow* y *deny*, y se escribe de manera automática el fichero */var/lib/nfs/xtab*, con los accesos de máquinas individuales, y en el fichero */var/lib/nfs/etab*, con los accesos de equipos que forman parte de subredes.

Como opciones tenemos

- -a: añadir los nuevos equipos/redes a los que ya hay (sólo añadir)
- -r: actualiza las tablas a partir de */etc/export*
- -v: muestra la info que hay en ese momento en los ficheros

2.3.2. Clientes NFS

Como ya se ha dicho, en el lado de los clientes podemos acceder mediante *mount* o mediante *fstab*:

1. mediante *mount*:

```
mount -t nfs servidor:/home/compartida /mnt/comparte
```

2. en el archivo *fstab*:

```
servidor:/home/comparte /mnt/comparte nfs [opciones]
```

2.4. SAMBA

Samba es una serie de servicios que nos van a permitir compartir datos, impresoras, y autenticación de usuarios, con un modelo cliente servidor, por medio de los protocolos SMB/CIFS (Server Message Block / Common Interface File System)², ambos originalmente propiedad de Microsoft. Fue creado por Andrew Tridgell, y dado que el nombre SMB ya estaba registrado por MS, utilizó el diccionario del sistema y encontró la palabra *samba* (`grep -i 's.*m.*b.*'`), que fue la que se adoptó como nombre oficial para el proyecto.

Samba nos permite compartir ficheros entre máquinas windows y unix. Nos permite logearnos a un dominio unix, logearnos a un dominio windows. E incluso actuar como Controlador de Dominio Primario (PDC, Primary Domain Controller) Windows, o montar sólo mediante Samba un PCD y un SDC (secondary). A día de hoy, aún no se puede crear un SDC Samba para un PDC Windows.

Para usar Samba, tenemos que tenerlo instalado, y sus *daemons* tienen que estar corriendo. Los demonios son:

²Ambos son el mismo protocolo. SMB era el nombre antiguo, y CFS es el nombre que se le da actualmente

- `smbd`: comparte y autentifica
- `nmbd`: actúa como servidor WINS

Para iniciar los dos servicios a la vez, usamos (en Ubuntu):

```
/etc/init.d/samba start
```

Si trabajamos en Fedora:

```
/etc/init.d/smb start
```

En caso de necesitar reiniciar el servicio, podemos usar *restart*. Para detenerlo, *stop*.

La configuración de samba (en Ubuntu) se realiza en el archivo:

```
/etc/samba/smb.conf
```

Es un fichero de texto plano, que se compone de diferentes líneas. El fichero se divide en secciones, que se nombran entre corchetes [*seccion*], y en cada sección tendrá una serie de parámetros.

Los parámetros se indican de la siguiente forma:

```
parametro = valor
```

Samba no es *case sensitive*, EXCEPTO en el caso de indicar rutas³.

En los parámetros, los espacios en blanco en los extremos son ignorados, pero si son espacios ENTRE el valor (por ejemplo, `parametro = valor1 valor2 valor3`), SI se tienen en cuenta.

Para comentar una línea se indica con `;` (al pcpo de la línea o como explicación de la línea).

Para continuar una línea muy larga en la siguiente línea, usamos la barra invertida `\`.

Como valores booleanos (lógicos), podemos usar indistintamente una de estas tres formas:

- 1-0
- True-False
- Yes-No

³no es igual `/home/user/` que `/home/User/`

En el fichero tenemos una sección *[global]*, donde se configuran parámetros generales del servidor, otra llamada *[printers]* donde se configuran las impresoras, y otra llamada *[homes]*, donde se configuran los directorios personales de los usuarios. La sección *[global]* tiene que existir obligatoriamente.

Si queremos control sobre la autenticación de los usuarios, necesitamos añadir la sección *[netlogon]*.

Por último, para crear carpetas compartidas, crearemos una nueva sección por cada recurso compartido. Además, deberemos autorizar a los usuarios correspondientes en Samba. Esto es muy importante, porque los usuarios locales del servidor (como root) NO están autorizados por defecto.

Si dos parámetros de configuración se contradicen, tiene validez la última configuración que se encuentre (la que esté más abajo en el archivo).

Podemos configurar samba a mano desde línea de comandos, o mediante las herramientas *SWAT* (Samba Web Administration Tools)

Para facilitar la configuración, podemos usar variables macro, que tienen un valor pre-determinado (se indican con un signo de %). Tenemos las siguientes (atención a mayúsculas/minúsculas):

- %U El usuario que ha iniciado sesión
- %G el grupo primario del usuario que ha iniciado sesión
- %h nombre del host donde se ejecuta samba
- %m nombre de la máquina cliente (NETBIOS)
- %L nombre NETBIOS del servidor
- %M nombre de internet del cliente (por IP, o por DNS)
- %R nombre del protocolo con el que se establece la comunicación
- %d ID del proceso actual en el servidor
- %a arquitectura del cliente (el Sistema Operativo)
- %I la IP del cliente

Samba emplea el dominio *smbd* para compartir los archivos e impresión, autenticar los usuarios, bloquear los recursos. puerto 139 y 445 tcp.

comentarios en el archivo *smb.conf* solo si al pcpo de línea van con ; # |

2.4.1. Tipos de servidores SAMBA

En Samba podemos adatar la configuración para que se ajuste al entorno donde vamos a integrarla. Hay pocas cosas que no pueda hacer, entre ellas, ser un backup de PDC Windows (o viceversa, que windows sea backup de PDC Samba).

Podemos crear servidores independientes, que se ponen en red pero no son PDC ni parte del control de usuarios. Simplemente dan acceso a recursos. El nivel de seguridad que podemos poner depende del uso. Podemos emplear el nivel de seguridad `security = share` para lograr un acceso libre y anónimo, aunque esto puede crear ciertos problemas de acceso. Para evitar estos problemas, podemos emplear los parámetros `guest only = yes`, `guest account = <nombre cuenta invitado>`.

Si empleamos `security = user`, necesitamos logearnos al recurso con nombre y contraseña (que pueden estar almacenadas en samba, o ser consultadas mediante LDAP a otra máquina, ya sea GNU/Linux o Windows).

Además, podemos ser servidores miembros de un dominio con las opciones `security = domain` (Dominio de NT4), o `security = ads` (Dominio en Active Directory).

Otra opción es ser controladores de dominio, realizando nuestra máquina samba la validación de los usuarios de la red. Podemos emplear `tbdsm`, con lo que la base de datos de usuarios es la propia de Samba, o emplear LDAP (la mejor solución, la base de datos de usuarios es LDAP). Además, podemos ser PDC, y tener backups de PDC (como ya se ha dicho, sólo puedo hacer backups de otros PDC samba).

Cuando conviven diversos servidores en la red, podemos indicar la “prioridad” de cada servidor. Podemos definir la prioridad de un servidor mediante la opción `OSLevel`, desde 1 hasta 99, siendo 99 la mayor prioridad. Los sistemas Windows PDC tienen como prioridad 35, así que cualquier servidor samba que tenga mayor prioridad será capaz de “adueñarse” de la red donde esté.

Tener en cuenta que en la sección [Homes] no puede emplearse la opción `force user` ni `force group`, dado que el usuario se logea a su directorio personal del sistema GNU/Linux (siempre que este exista, cosa que debe ocurrir, dado que para logearse tiene que estar dado de alta). En el caso de usar Homes, hay que asegurarse de que las cuentas de máquina creadas en el sistema tienen todas el \$ al final de su nombre en el archivo `passwd`, para evitar que puedan entrar como si fueran usuarios. Además, podemos emplear el parámetro `invalid users = ¡lista de usuarios!` para evitar a usuarios concretos el acceso.

- Servidores Independientes: como ya hemos dicho, no se rigen por el dominio, solo comparten recursos. podemos tener diversos tipos
 - solo lectura: comparte de manera anonima y como solo lectura. En Global, especificamos grupo de trabajo, nombre netbios del samba, `security = share`. En la parte de recursos, creamos los recursos que deseamos compartir mediante su path local,

y como opciones `read only` y `guest only`. En caso de problemas, podemos especificar además la cuenta de `guest` que se va a emplear, con `guest account = nobody`. OJO, recordad que los permisos GNU/Linux el sistema prevalecen siempre sobre los recursos compartidos, si nuestra carpeta compartida tiene permisos `000`, nadie podrá acceder ni por local, ni mucho menos en un acceso remoto. Si queremos un poco más de seguridad, podemos afinar más con los permisos y propietarios/grupo de las carpetas, empleando `force user =` y `force group =`

- Lectura, escritura: ponemos `read only = no`, y aquí si deberemos emplear las opciones `force user` y `force group`, para evitar que se produzcan conflictos con los permisos de los usuarios y grupos al escribir en los nuevos archivos (si no se usa, cualquier usuario que pueda entrar creará archivos de su propiedad/grupo, y otros usuarios no podrán usarlos). Aunque se puede hacer, no es recomendable un servidor anónimo de solo lectura.
- Anónimo de Impresión: deberemos incluir la sección `printers`, y ahí definiremos que impresoras puede ver el cliente. Podemos integrar los drivers en el servidor, o podemos hacer que el cliente tenga que tener los drivers, mediante `use client driver = yes`. Hay que tener en cuenta que el recurso compartido tiene que tener `printable = yes` para poder imprimir en la impresora. Además, tener en cuenta que podemos tener más de una cola de impresión en una misma impresora (para diferentes prioridades, configuraciones de impresión).

Podemos poner seguridad mediante `security = user`, con lo que obligamos a autenticar usuarios para que puedan imprimir.

- Servidores Miembro de un dominio: si ya tenemos una red implementada mediante dominios, y deseamos añadir un nuevo servidor (por ejemplo, web, ficheros, impresión) en la red, sin tener que pagar las licencias de MS, podemos crear el servidor con samba. Podemos unirnos a dominios NT y Active Directory. En ambos casos deberemos unir la máquina Samba al dominio, y en caso de AD, deberemos tener instalado Kerberos, para permitir la correcta autenticación. Además, deberemos tener el parámetro `realm = EJEMPLO.COM` (el nombre del dominio en mayúsculas). La seguridad en estos casos será ADS (active directory), necesitamos la opción `encrypt passwords = yes`, y le debemos indicar el servidor kerberos de la red (`password server = maquina.kerberos.com`). Si solo tengo una máquina GNU/Linux en el dominio, esa máquina será la encargada del demonio kerberos (que habrá que configurar aparte). Si tengo más de una máquina GNU/Linux, una de ellas actúa de servidor kerberos y el resto le consultan a ella (`password server` sería la máquina con kerberos activo).

Los pasos concretos son:

- Configurar samba (`smb.conf`) en el nuevo servidor GNU/Linux que vamos a unir
- Configurar kerberos (en la máquina nueva, o en el servidor kerberos correspondiente)
- Crear la cuenta de máquina en el AD (como admin del AD)
- Asociar el servidor como miembro del dominio.

- Servidores miembro de dominio NT4

En este caso, la seguridad se pone como domain. Samba actua como pasarela con el servidor NT4

- Controlador de Dominio: en este caso, Samba gestiona los usuarios y permisos. Podemos emplear TDBSAM (se usa la propia base de usuarios de samba, con limitaciones sobre el nº de usuarios, y con problemas de compatibilidad con AD), o podemos usar LDAP, que ofrece compatibilidad total con AD. Las bases de datos de usuarios se demoniman SAM (securty account manager). Windows y Samba almacenan de diferente manera estas bases de datos, y es por esto por lo que un Samba y un Windows no pueden hacerse backups mutuos.

El parámetro que indica la base de datos a emplear es `passdb backend = tdbsam` (o `ldap`). El parámetro que indica que es controlador primario es `domain master = yes`

- Controlador primario con TDBSAM hay un guía en el moodle, con un `smb.conf` ya hecho.
- Controlador primario con LDAP.
Más complicado, puesto que hay que configurar LDAP entre otras cosas, antes de configurar Samba. Crear cuentas de administradores, usuarios, máquinas...
- Controlador secundario de domino con LDAP:
lo mejor en este caso es montar el secundario, y realizar réplicas de la base de datos del primario cada cierto tiempo, para que en caso de caer el primario esté disponible el secundario.

2.5. LDAP: Administración de dominios GNU/Linux usando LDAP

2.5.1. Introducción

Un dominio nos permite centralizar la gestión de recursos (usuarios, contraseñas, recursos de red...) en una red local. Un dominio en Windows 2003 se implementa mediante LDAP y DNS (formando lo que se conoce como Active Directory). En el caso de GNU/Linux, se logra con LDAP y Samba. LDAP es una implementación concreta del protocolo X.500, y se encarga de la autenticación, mientras que Samba se encarga de la gestión de los recursos. Con LDAP, podemos crear una gestión

2.5.2. Concepto de dominio

Un dominio es un conjunto de equipos interconectados que comparten una misma gestión centralizada.

2.5.3. Servicios de directorio y LDAP

La base de datos LDAP donde se almacenan las cuentas de usuarios no es una base de datos normal, sino que se encuentra optimizada para realizar búsquedas de usuarios y permisos. La estructura de la BBDD LDAP es de árbol invertido (parecida a la estructura del sistema de ficheros de GNU/Linux). Tenemos una raíz, de la que parten nodos, donde cuelgan las hojas de los objetos. Cada objeto del árbol tiene una serie de atributos, y cada atributo tiene un nombre, una serie de valores, y un tipo para los valores. Además, cada objeto tiene lo que se conoce como *nombre distinguido*. Un nombre distinguido nos permite identificar de manera única un objeto dentro del árbol, y va desde el objeto hasta la raíz. Este nombre se llama *dn*. La raíz del árbol se denomina *base* o *sufijo de la organización*. Por ejemplo, en el caso del instituto, tendríamos `dn='dc=bezmiliana,dc.org'`, siendo *org* la base. Otro ejemplo: `dn='uid=pepe,ou=People,dc=bezmiliana,dc=org'`. En este caso, *pepe* es un objeto usuario, dentro de la unidad organizativa *People*, dentro de *bezmiliana*, con la raíz *org*. En LDAP, podemos definir los atributos que necesitamos para los objetos. Todo este formato de nomenclatura se llama *formato LDIF* (LDAP Data Interchange Format). En una máquina Unix los usuarios del sistema se almacenan en la base de datos *passwd*, y existen herramientas automatizadas que toman esos datos y los convierten en objetos en formato LDAP, para permitir una transición sencilla a un entorno LDAP.

2.5.4. Implementar un dominio GNU/Linux con LDAP

1. Instalar LDAP: en Ubuntu necesitamos los paquetes `slapd` y `ldap-utils`, que podemos instalar con `apt-get`
2. Configurar las funciones básicas de LDAP (tales como el sufijo principal).
3. Comprobar el funcionamiento del cliente LDAP, asegurando que se ha instalado bien. Básicamente se trata de realizar una consulta al LDAP y comprobar la respuesta del mismo. Hay un comando de test que sirve para hacer esto de forma automática.
4. Migrar y actualizar la información de usuarios y grupos. Tendremos dos tipos de usuarios, que se migran de forma diferente:
 - Usuarios Unix de la máquina servidor, que se hace mediante la conversión a formato LDIF, para luego importar esos datos al LDAP.
 - Usuarios samba, para migrar de `tbdSAM` a `ldapsam`, mediante la herramienta `pdbedit` (como `root`). Permite añadir, eliminar y modificar usuarios a samba, así como exportar los usuarios de `tbdSAM` a `ldapsam`. Para hacer esto último, haremos (como `root`):

```
pdbedit -i tbdSAM -e ldapsam
```

Siendo `i` importar, y `e` exportar. Podemos poner en lugar de `tbdSAM` y `ldapsam` la ruta hasta los ficheros que queramos emplear.

5. Comprobar que la migración se ha llevado a cabo correctamente.
6. Configurar la autenticación mediante LDAP en el archivo *smb.conf* con las opciones adecuadas.

2.5.5. Instalación de LDAP

Capítulo 3

Scripts

3.1. Introducción a Shell Scripting en Bash

Son guiones de comandos del sistema, con tal potencia que se pueden llegar a considerar un verdadero lenguaje de programación. Sin embargo, un shell script es interpretado, es decir, son archivos de texto plano que se ejecutan línea a línea por medio de la shell del sistema. Dado que son ficheros que vamos a ejecutar, debemos asignarles permisos de ejecución para poder lanzarlos.

Son muy importantes porque permiten automatizar tareas del sistema muy complejas de manera cómoda (PE añadir/eliminar usuarios, realizar comprobaciones...)

Son archivos de texto plano, formados por comandos del sistema. Todo archivo de shell script comienza por una línea que indica que shell del sistema se va a encargar de interpretar el script, PE:

```
#!/bin/bash
```

Esta shell (bash) es la mas extendida del mundo *nix. Es una evolución más potente de la shell sh, y es la que vamos a emplear aquí.

Un script de ejemplo:

```
#!/bin/bash
echo "Hola Mundo"
```

y lo guardamos como *ejemplo.sh*. La extensión *sh* no es obligatoria, pero suele usarse para identificar los archivos script. Le asignamos al archivo permisos de ejecución:

```
chmod u+x ejemplo.sh
```

y lanzamos el script con el comando:

```
./ejemplo.sh
```

Este script se abrirá en una shell hija de la shell que lo lanzó, y cualquier acción que lleve a cabo estará dentro de esa subshell.

3.1.1. Diferencias entre ", ' y ‘

Lo que se incluya entre " (comillas dobles) se considera un literal (una cadena). Es decir, ponemos entre comillas dobles todo aquello que queramos que aparezca tal cual. Nos van a permitir llevar a cabo expansión de variables, es decir, podemos usar variables en el código del script que al ejecutarlo serán sustituidos por el valor que tienen esas variables en el sistema en ese momento.

Lo que se incluya entre ' (comillas simples) no realizan expansión de variables.

Lo que se incluya entre ‘ (acento grave) va a ser sustituido por variables que existan en la shell, es equivalente a emplear \$(variable o comando)

PE: las siguientes dos líneas son equivalentes

```
echo 'ls -l'
echo $(ls -l)
```

Dado que el símbolo \$ es usado por el sistema, para que aparezca debemos precederlo por la barra \ de la siguiente forma:

```
\$
```

Si no aparece la barra, le estamos indicando que lo que aparezca detrás del \$ es el nombre de una variable.

Ejemplo real:

```
moebius:~ hector$ a='ls -l'
moebius:~ hector$ echo $a
ls -l
moebius:~ hector$ echo '$a'
$a
moebius:~ hector$ echo "$a"
ls -l
moebius:~ hector$ echo '$a'
Muestra el resultado de hacer "ls -l" en la carpeta actual.
moebius:~ hector$ echo "el comando es: $a"
el comando es: ls -l
```

para asignar números a una variable, usamos la orden `let`:

```
moebius:~ hector$ let a=13
moebius:~ hector$ echo $a
13
moebius:~ hector$ let a="$a"+1
moebius:~ hector$ echo $a
14
moebius:~ hector$ let a=${a+1}
moebius:~ hector$ echo $a
15
```

3.1.2. Ejecución secuencial y condicional

Si en un script separamos dos comandos por `&&`, obligamos a que el segundo comando se ejecute sólo si el primer comando se ha completado satisfactoriamente.

Si los separamos con `||`, (or), el segundo comando se ejecuta sólo si el primero ha fallado.

3.2. Evolución de las shells

3.2.1. Bourne

La primera fue Shell Bourne, creada por Steven Bourne en 1974. Antes de esta shell sólo había un intérprete básico de comandos. Esta shell no fue incorporada hasta el año 1978, en la versión v7 del sistema Unix. Bourne permitía control de flujo, variables, ... Ya estaba enfocada a permitir programación mediante scripts. De hecho, todas las shell posteriores son mejoras de la Bourne (exceptuando CShell, que veremos posteriormente).

Características:

1. Interactividad: permite una ejecución interactiva de los comandos del usuario.
2. Incorpora el background, para poder ejecutar comandos en segundo plano.
3. Redireccionamiento de la entrada y salida de las órdenes.
4. Capacidad de empleo de pipes (—).
5. Permite el uso de comodines y autocompletar nombres.
6. Permite el uso del entrecomillado (simple, doble, grave).
7. Permite personalizar el prompt (para que aparezca la información que el usuario elija).
8. Admite perfiles de usuario.

9. Introduce el uso de variables.
10. Funciones.
11. Control de flujo.
12. Arrays.
13. Traps¹ y manejo de errores.

Después de la aparición de Bourne comenzaron a proliferar los scripts.

3.2.2. CShell

Aparece en 1978, y fue la primera alternativa a Bourne. Fue desarrollada por Bill Joy (creador de vi). Como su nombre indica, la sintaxis de la shell está tomada del lenguaje de programación C, para que los programadores de C sean capaces de crear scripts del sistema de manera rápida con los conocimientos de programación que ya poseían.

Ventajas:

1. Mayor interactividad.
2. Implementa el histórico de comandos.
3. Presenta incompatibilidad entre los scripts de CShell y Bourne. Los scripts existentes no funcionaban en la nueva shell, y pese a presentar numerosas ventajas y mejoras frente a Bourne, los programadores de scripts no querían traducir sus scripts al nuevo sistema.
4. Incorpora Alias (poder llamar a un comando con otro nombre).
5. Introduce el comando jobs para el tratamiento de las tareas.

Tuvo una aceptación relativa, debido a su incompatibilidad.

3.2.3. TCSH

Es una evolución muy reciente de C Shell.

¹Un comando que permite asociar un conjunto de órdenes a una señal del sistema, PE que una señal Kill lance un script de comandos

3.2.4. Shell Korn

Creada por David G. Korn, en 1983. Pretendía integrar Bourne y C Shell, para que fuese compatible con ambas versiones, y con retroactividad (es decir, que permitiese lanzar cualquier script o comando creado para cualquier versión de ambas shell). Debido a esta característica, esta shell es muy grande, pero es la más rápida en ejecución. Incorpora además una serie de mejoras en la sintaxis de Shell Scripts.

1. Case
2. Funcionalidades de C más avanzadas (mejoras en bucles y similares).

Además, llegó a hacer que la línea de comandos fuese compatible con las órdenes de Emacs y Vi de manera directa. Por desgracia, casi no se utiliza.

3.2.5. POSIX

Se trató de crear un estándar para Shell, asegurando la compatibilidad. Comenzó con POSIX.1 y actualmente empleamos POSIX.2 Para que una shell fuera compatible con POSIX debía cumplir unos requisitos publicados. POSIX tomó como base a la shell Korn. Fue creada por las empresas AT&T y HP junto con OSF².

3.2.6. Bash

Bourne Again Shell es la shell por defecto que se emplea en todos los sistemas actuales (GNU/Linux, BSD, ...). Tiene compatibilidad POSIX.2 total, y tomó mucha sintaxis de CShell (es parecida por tanto al lenguaje shell). Tiene una línea de comandos más amigable, permitiendo uso de históricos, usar arrays indexados no numéricos (se verá más adelante), múltiples funciones de cadenas (extraer partes de variables como cadenas...), aritmética en base 2, base 10, base 16. Expansión de variables, autocompletado de nombres, usuarios, máquinas...

Resulta una shell muy adecuada para el uso administrativo, por la enorme cantidad de recursos que se pone a disposición del usuario.

3.2.7. Identificar la shell actual

```
echo $SHELL
```

²Open Software Foundation

Devuelve la shell actual.

Podemos ver las shell disponibles en el sistema viendo el contenido del fichero `/etc/shells`. Podemos cambiar la shell asignada al usuario con el comando `chsh -s /bin/bash`, donde la `s` indica que se haga predeterminada, y por último indicamos la shell que deseamos emplear. Al indicar el `s`, se cambia la shell en el archivo `/etc/passwd`.

3.3. Variables en Scripts

Podemos definir una variable mediante un nombre y un valor. No es necesario (aunque se puede) especificar un tipo de dato. Por convenio, las variables se suelen nombrar con letras mayúsculas.

Ejemplo:

```
#!/bin/bash
DECIR="HOLA MUNDO"
echo $DECIR
NUMERO=4
echo $NUMERO +3
echo $(( $NUMERO+3 ))
let NUMERO=$NUMERO+3
echo $NUMERO
NOMBRE_FICHERO="Red"$(date +%d%m%y)
```

la salida de este script es:

```
HOLA MUNDO
4+3
7
7
Red260308
```

3.3.1. Operadores aritméticos para let

Let sirve para asignar exclusivamente valores numéricos. Podemos emplear `+`, `-`, `*`, `/`, `%`

3.3.2. Comparación de cadenas

La comparación de igualdad se hace con el operador =

```
cade1=cade2
```

PE³:

```
if [ cad1 = cad2 ]; then
  echo "Son iguales"
fi
```

Podemos usar otros operadores de comparación: != (distinto); \< (precedencia lexicográfica)⁴ (y al contrario, \>)⁵; -n cad1, que da verdadero si cad1 está definida y no es nula; -z cad1, que indica que está vacía o no definida.

PE:

```
#!/bin/bash
DECIR="HOLA MUNDO"
echo $DECIR
NUMERO=4
echo $NUMERO+3
echo $(( $NUMERO +3 ))
let NUMERO=$NUMERO+3
echo $NUMERO
NOMBRE_FICHERO="Red"$(date)
echo $NOMBRE_FICHERO
A="21"
B="99"
if [ $A \< $B ]; then
  echo "Uno"
fi
if [ $A -gt $B ]; then
  echo "Dos"
fi
let A=21
let B=5
if [ $A \< $B ]; then
  echo "Tres"
fi
if [ $A -gt $B ]; then
  echo "Cuatro"
fi
```

³OJO: hay que dejar espacios en blanco entre todos los elementos de la comparación (corchetes, cadenas, comparador) o nos dará un error de sintaxis.

⁴vamos, ordenación alfabética

⁵OJO, hay que proteger los < y > con porque sino los considera una redirección

3.3.3. Comparación de datos numéricos

Num1 `-eq` Num2 da verdadero si son iguales; Num1 `-ne` Num2 si son distintos; `-lt` para menor que; `-gt` para mayor que; `-le` menor o igual; `-ge` mayor o igual.

3.3.4. Comparación de ficheros

`-a` fichero es verdadero si fichero existe.

```
if [ -a /boot/grub/menu.lst ]; then
  cat /boot/grub/menu.lst
fi
```

`-d` fichero verdadero si existe y es de tipo directorio.

`-f` fichero existe y es un fichero regular.

`-r` fichero existe y tiene permiso de lectura.

`-w` fichero existe y tiene permiso de escritura.

`-x` fichero existe y tiene permiso de ejecución.

`-s` fichero existe y su tamaño es mayor que 0 bytes.

f1 `-nt` f2 si f1 es más nuevo que f2 (fechas de creación).

f1 `-ot` f2 si f1 es más viejo que f2 (fechas de creación).

3.3.5. Errores típicos

```
if [3 -eq 5]; then...
```

Falta un espacio delante del 3, y devuelve un error: `[3: command not found`.

```
if [ "J" -eq "J" ]; then...
```

Estamos usando un operador numérico con valores no numéricos.

```
if [ 3 = 4 ]; then...
```

Estamos usando un comparador de cadenas con números.

3.4. Estructuras condicionales

3.4.1. if

```
if condicion; then
  orden1
  ...
  ordenN
fi
```

En la condición, normalmente pondremos corchetes: `[condicion]`⁶. Para introducir otros casos con `if`, lo indicamos con `elif`, y lo concluimos con `else`. Para terminar el `if`, cerramos con `fi`:

```
if condicion; then
  orden1
elif condicion2; then
  orden2
else
  orden3
fi
```

Notar que puede haber múltiples `elif`, pero sólo un `else`. Un `if` simple con una sola rama sería:

```
if condición; then
  orden1
else
  orden2
fi
```

3.4.2. Case

Una estructura `case` se realiza de forma muy parecida a la estructura `switch...case` de C:

```
case $VARIABLE in
  VALOR1)
  orden
  orden
  ordenfinal;;
  VALOR2)
  orden
  orden
```

⁶También podemos emplear dobles paréntesis `((condicion))`

```
ordenfinal;
*)
ordendefault
ordendefault
ordendefaultfinal;;
esac
```

Se inicia con `case $var in`, y cada posible valor se pone delante de un paréntesis `)`, el caso por defecto (`default` en C) se indica con un `*`. Cada una de las órdenes en cada caso se termina directamente con un salto de línea, y cada sentencia final se termina con un `;;` (doble punto y coma, sería el equivalente al `break` en C). Si no ponemos los `;;` se sigue ejecutando hasta que se encuentre el siguiente `;;` (o el final del `case`).

3.4.3. for

Un bucle `for` se compone de las siguientes instrucciones:

```
for VARIABLE in conjunto; do
sentencias
done
```

Notar que `VARIABLE`:

- No lleva un `$` delante.
- No tiene porqué estar declarada antes, podemos declararla en el mismo `FOR`

Los elementos del conjunto van separados uno de otros por un espacio. Notar que en esto influye una variable de sistema llamada `IFS` (`internal field separator`).

IFS

`IFS` es donde el sistema almacena el separador de campos por defecto. Si en esta variable almacenamos otro valor, ese será el separador que se emplee al procesar campos. Esto es útil por ejemplo para procesar línea a línea un fichero, en lugar de palabra a palabra. Si dentro de un script modificamos el valor de `IFS`, ese nuevo valor sólo tendrá utilidad dentro del script, y en cuanto haya terminado el sistema volverá a tener su valor original de `IFS`. Ejemplos:

```
IFS=: ' establece el separador en ":"
IFS=$'\n' establece el separador en "salto de línea"
```

Notar que en el último ejemplo, cuando queremos establecer como separador un caracter de control (como en el caso del salto de línea), tenemos que añadir el `$` fuera de las comillas simples.

3.4.4. for (estilo C)

El `for` también se puede usar de la misma manera que en lenguaje C, de la manera:

```
for ((i=1;i<=10;i++)); do
    echo $i
done
```

Notar que la única diferencia respecto a C son los dobles paréntesis, y que los parámetros del `for` terminan en `;` (no se encierra el código del interior del bucle entre `{` y `}`). Notar que este bucle admite todas las posibilidades del bucle en lenguaje C, tales como bucles `for` anidados, múltiples condiciones de salida, etc...

3.4.5. seq

Si necesitamos emplear un bucle `for` del mismo tipo que en c (variable inicial, final, incremento), podemos emplear la orden `seq` de la siguiente manera:

```
seq inicial incremento final
```

con lo que podemos hacer bucles tal y cómo lo hacemos en C: PE:

```
SECUENCIA=$(echo seq 1 1 20)
for I in $SECUENCIA;do
    comando
done
```

este ejemplo ejecutaría `comando` 20 veces, que son los 20 valores que se almacenan en la variable `SECUENCIA` por medio de la instrucción `seq`, y el comando se ejecuta cada una veces por medio del comando `for`

3.4.6. while

Se usa de la forma:

```
while [ condicion ] ; do
    sentencias
done
```

en este bucle, cada vez que se va a entrar se evalúa la condición, y si esta es cierta, se accede al cuerpo del bucle. En caso contrario, se sigue adelante. Esto es muy útil, PE, para validar entradas de datos y asegurar que se está leyendo un valor válido:

```
#!/bin/bash
#Validación con while
clear
read -p 'Introduce una calificación entre 0 y 10:' NOTA
while [ $NOTA -lt 0 ] || [ $NOTA -gt 10 ] ; do
    echo "Nota $NOTA fuera del intervalo"
    read -p 'Introduce una calificación entre 0 y 10:' NOTA
    if [ $NOTA -lt 0 ] || [ ${#NOTA} -eq 0 ] ; then
        break
    fi
done
```

En este caso, estamos empleando una nueva función de la shell, las llaves `{}` para que nos devuelvan ciertas características de las variables. En este caso, la sentencia:

```
${#NOTA}
```

Nos devuelve la longitud de la cadena, que utilizamos para comprobar que hemos introducido algún valor en `NOTA`. El `break` no hace saltar a la siguiente instrucción tras el bucle (en este ejemplo, terminaría el programa).

3.4.7. until

Tenemos otro tipo de bucle (similar al `while`), que es el bucle `until`, donde se hace el bucle hasta que cumpla la condición. Notar que es justo al contrario que un `while`. PE:

```
until [ condicion ] ; do
    sentencias
done
```

Si queremos convertir un `while` en un `until`, sólo tenemos que dar la vuelta a la condición, PE, en el caso anterior (analizador de nota), la condición del `while` se convierte en esto:

```
until [ $NOTA -gt 0 ] && [ $NOTA -lt 10 ] ; do
    ...
```

3.4.8. select

Este tipo de bucle es nuevo, no lo hemos visto en ningún lenguaje de programación. La forma general es:

```
select OPCION in conjunto_de_opciones ; do
    sentencias
done
```

Cuando nuestro programa llega al `select`, en pantalla se muestra el conjunto de opciones numerado, y espera a que introduzcamos un valor. Veamos un ejemplo concreto:

```
#!/bin/bash
#Menu con select
clear
select OPCION in ElBreakindanse ElCrusaito ElMaikelyason ElRobocop ; do
    if [ $OPCION = "ElBreakindanse" ] ; then
        echo "El chikichiki se baila así: 1, ElBreakindanse"
    elif [ $OPCION = "ElCrusaito" ] ; then
        echo "2, ElCrusaito"
    elif [ $OPCION = "ElMaikelyason" ] ; then
        echo "3, ElMaikelyason"
    else
        echo "4, ElRobocop"
        echo "Baila el chikichiki, baila el chickichiki"
        echo "Lo bailan los jevis también los frikis"
        break
    fi
done
```

La salida es:

```
1 ElBreakindanse
2 ElCrusaito
3 ElMaikelyason
4 ElRobocop
#?
```

3.4.9. Funciones

Podemos definir funciones en el script, de la siguiente forma:

```
function nombre
{
    codigo
    codigo
    codigo
}
```

Una vez creada, podemos llamar a la función simplemente poniendo su nombre en el script (como si se tratara de un comando).

3.4.10. Consulta avanzada de variables

Con ciertos parámetros podemos obtener información extra de las variables, como su nombre, longitud, contenido.

Imagina que tenemos una variable V :

- V es el nombre de la variable
- $\$V$ es el contenido de la variable
- $\${V}$ es el contenido de la variable, indicado para usar con arrays
- $\#{V}$ es la longitud de la variable
- $\${V-C}$ si V está definida y tiene valor lo conserva, y si no está definida o no tiene valor, toma el valor C , pero sólo para esa línea concreta y de manera temporal.
- $\${V=C}$ si V está definida y tiene valor, lo conserva, si no está definida o no tiene valor, toma el valor de C de manera definitiva, y se mantiene con ese valor durante el resto de la ejecución del script.⁷
- $\${V+C}$ si V está definida le asigna el valor C (tanto si está definida como si ya tiene un valor asignado), y ese valor se mantiene para el resto del script. Si V no está definida, la define como cadena vacía.
- $\${V?C}$ si V está definida, muestra su valor; si no lo está, muestra C en la salida de errores y finaliza el script.
- $\${V:+C}$ este caso es equivalente a $\${V+C}$, pero V debe estar definida y tener un valor (distinto de vacío) para funcionar.
- $\${V:=C}$ idem
- $\${V:-C}$ idem
- $\${V:?C}$ idem
- $\${V#P}$ Elimina (por la izquierda) los caracteres iniciales de V que coinciden con el patrón P .
- $\${V%C}$ Elimina (por la derecha) los caracteres finales de V que coinciden con el patrón P .

Si V es un array:

- $\#{V[*]}$ es el nº de elementos que contiene el array

⁷lógicamente, el valor se mantiene mientras no volvamos a cambiarlo.

3.5. Introducción de datos en scripts

3.5.1. Read

con el comando `read` podemos hacer lectura de datos desde dentro del script, de la siguiente manera:

```
read VAR
```

de esta forma leería desde teclado una variable y la almacenaría en `VAR`. Tenemos 3 opciones para `read`:

1. `-p` muestra un mensaje al leer:

```
read -p "Dame un nº: " NUM
```

2. `-nX` lee tantos datos como le indiques en `X`:

```
read -n3 NUMERO
```

Este ejemplo leería 3 pulsaciones de teclado y almacenaría el dato sin tener que darle a Enter.

3. `-s` No muestra los caracteres mientras lo escribes:

```
read -s SECRETO
```

3.6. Parámetros

Cuando llamamos a un script, podemos pasarle parámetros en la llamada, para que los tome como variables. Estas variables se denominan parámetros reemplazables. Podemos tener tantos parámetros como necesitemos, y dentro del script los llamamos con `%0...%n`. El parámetro `%0` es el propio nombre del script, y el resto de parámetros van numerados desde el 1 hasta el `n`.

PE, si llamamos a un script de la siguiente manera:

```
./par.sh 5 fichero
```

tendríamos los siguientes parámetros:

1. `$0`: `par.sh` (sin el `./`). Además, incluirá la ruta completa al script.
2. `$1`: 5

3. \$2: fichero

tenemos otras variables especiales:

- `$#` devuelve el nº de parámetros que se han recibido sin contar el nombre del programa.
- `$*` lista completa de todos los parámetros.
- `@$` lista completa de todos los parámetros.
- `$?` devuelve el código de error de la última orden que se ha ejecutado (0 si ha ido bien, otro nº en caso de error).
- `$$` muestra el identificador del proceso que está lanzando el script.

3.6.1. Desplazamiento de parámetros: `shift`

Cuando trabajamos con parámetros, podemos pasar al script tantos como necesitemos, pero una vez que lo estemos ejecutando, sólo podemos acceder a los parámetros `$1` al `$9`. Para poder acceder al resto de parámetros (del `$10` en adelante), deberemos usar la orden `shift`, que sirve para eliminar los primeros parámetros y dejar hueco para los siguientes. Así, si tenemos dos parámetros `$0` y `$1`, y hacemos `shift`, el valor del parámetro `$1` desaparece y toma su lugar el valor de `$2`. Por defecto, `shift` elimina un sólo parámetro, pero podemos indicarle cuantos parámetros queremos desplazar: `shift n`, que desplaza `n` parámetros. Veamos un ejemplo:

```
#!/bin/bash
echo $1
echo $2
echo $3
echo "Desplazo"
shift
echo $1
echo $2
echo $3
```

y llamamos al script de la siguiente manera:

```
./param.sh uno dos tres
```

el resultado será:

```
uno
dos
tres
Desplazo
dos
tres
```

3.7. Arrays en scripts

Podemos emplear arrays en la ejecución de los scripts, pero en el caso de que una variable sea un array, deberemos encerrar su nombre entre llaves de la siguiente manera (supongamos un array llamado TABLA):

```
#{TABLA}
```

SI queremos acceder a un elemento concreto del array, lo hacemos mediante corchetes.

```
#{TABLA}[indice]
```

Notar que cuando empleábamos parámetros en un apartado anterior, sólo podíamos mostrar directamente los parámetros del 1 al 9. Pero al encerrar un valor entre llaves, nos permite protegerlo y usarlo directamente como valor, de forma que:

```
$9 ${10} ${22}
```

contienen los parámetros nº9, 11 y 22, sin necesidad de emplear `shift`.

Si queremos declarar un array con diferentes elementos, simplemente le asignamos un valor. No hay que declarar la variable previamente:

```
{TABLA}[0]="Malaga"  
{TABLA}[1]="Granada"  
...
```

Notar que estamos en asignación, y por lo tanto no ponemos un \$ delante: sólo empleamos el \$ para mostrar el contenido.

Por ejemplo, una lectura de el nombre de todas las provincias sería:

```
for ((i=0;i<=7;i++)); do  
  read-P "Dime una provincia: " PROV  
  {TABLA}[$i]=$PROV  
  echo 'TABLA['$i']= '${TABLA}[$i]  
done
```

Notar que en el último `echo` encierro algunos elementos entre comillas simples ' para evitar que se expanda el valor contenido dentro.

3.8. Ejercicios Scripts

1. Indicar si un nº es par

```
#!/bin/bash
read -p "Dame un numero: " NUM
let NUM2=$NUM%2
if [ $NUM2 -eq 0 ]; then
    echo "Es par"
else
    echo "Es impar"
fi
```

2. Ver si un n° está entre un máximo y un mínimo

```
#!/bin/bash
read -p "Minimo: " MIN
read -p "Maximo: " MAX
read -p "Numero: " NUM
if [ $NUM -ge $MIN ]; then
    if [ $NUM -le $MAX ]; then
        echo "$NUM está en el rango de $MIN y $MAX"
    else
        echo "$NUM es mayor que el maximo $MAX"
    fi
else
    echo "$NUM es menor que el minimo $MIN"
fi
```

Otra versión del mismo ejercicio, con las dos comprobaciones en un mismo if:

```
#!/bin/bash
read -p "Minimo: " MIN
read -p "Maximo: " MAX
read -p "Numero: " NUM
if [ $NUM -ge $MIN ] && [ $NUM -le $MAX ]; then
    echo "Esta entre $MIN y $MAX"
else
    echo "Esta fuera del rango $MIN y $MAX"
fi
```

3. Traducir una nota numérica a texto

```
#!/bin/bash
read -p "NOTA: " NOTA
#primero comprobamos si la nota es valida
if [ $NOTA -ge 0 ] && [ $NOTA -le 10 ]; then
    if [ $NOTA -ge 9 ]; then
        echo "$NOTA: Sobresaliente"
    else
        if [ $NOTA -ge 7 ]; then
            echo "$NOTA: Notable"
        else
            if [ $NOTA -ge 6 ]; then
                echo "$NOTA: Bien"
            else
```

```

        if [ $NOTA -ge 5 ]; then
            echo "$NOTA: Suficiente"
        else
            if [ $NOTA: -ge 3 ]; then
                echo "$NOTA: Insuficiente"
            else
                echo "$Nota: Muy deficiente"
            fi
        fi
    fi
fi
fi
fi
else
    echo "Nota no valida: $NOTA"
fi

```

4. Comprobar si un año es bisiesto Los años bisiestos son los múltiplos de 100 que no lo son de 400

```

#!/bin/bash
read -p "Año: " ANNO
let RESTO4=$ANNO%4
let RESTO100=$ANNO%100
let RESTO400=$ANNO%400
if [ $RESTO4 -eq 0 ]; then
    if [ $RESTO100 -eq 0 ] && [ $RESTO400 -ne 0 ];then
        echo "No es bisiesto"
    else
        echo "Bisiesto"
    fi
else
    echo "No es bisiesto"
fi

```

5. Crear un fichero `ejemplo5ddmmy.txt` (la fecha actual), con una lista de todos los archivos `*.txt` y `*.sh` de nuestro directorio personal que contenga Nombre:Tamaño ⁸

```

#!/bin/bash
IFS=$'\n'
FICHERO='lista' `date +%d%m%y`
if [ $# -eq 3 ]; then
    if [ -f $FICHERO ]; then
        rm $FICHERO
    fi
    if [ -d $1 ]; then
        FICHEROS=$(ls -l $1/*. $2 $1/*. $3)
        for i in $FICHEROS; do

```

⁸NOTA: en este ejemplo se ha empleado un `cut` con el campo 10 (opción `-f10`), que es el campo de nombre para un sistema OS X, en el caso de un GNU/Linux, el valor del campo será probablemente el 9 (opción `-f9`), pero sería mejor comprobarlo antes, o incluso usar expresiones regulares para localizar con exactitud el nombre en cada línea.

```

        NOMBRE=$(echo $i|tr -s ' ' ':' |cut -d ':' -f10)
        TAMA=$(echo $i|tr -s ' ' ':' |cut -d ':' -f5)
    echo $NOMBRE:$TAMA >> $FICHERO
    done
    clear
    cat $FICHERO
else
    echo "$1 no es un directorio"
fi
else
    echo "Sintaxis:$0 directorio extension1 extension2"
fi
fi

```

Por ejemplo en mi sistema, donde tengo varios archivos `sh` y ningún `txt`, la llamada y el resultado son (ojo, es la salida en un OS X, no en un GNU/Linux, puede que en tu máquina salga diferente):

```
./script.sh /Users/hector sh txt
```

genera un archivo `lista` con lo siguiente:

```

/Users/hector//ej1.sh:308
/Users/hector//ej2.sh:292
/Users/hector//ej3.sh:211
/Users/hector//ej4.sh:619
/Users/hector//ej5.sh:269
/Users/hector//ej6.sh:61
/Users/hector//ej7.sh:412
/Users/hector//ej8.sh:224
/Users/hector//param.sh:82
/Users/hector//param2.sh:167

```

6. Devuelve el nº de parámetros que hemos pasado al script

```
#!/bin/bash
echo He recibido $# parametros
```

7. Escribe el parámetro 3 y el 5 (y comprueba que el nº de parámetros es correcto)

```
#!/bin/bash
clear
if [ $# -gt 4 ]; then
    echo Parametro 3: $3
    echo Parametro 5: $5
else
    echo Me has dado $# parametros y me hacen falta al menos 5 parámetros
fi

```

8. Escribe la lista de todos los parámetros

```
#!/bin/bash
echo He recibido estos parametros: $*
```

9. Escribe la lista de todos los parámetros separados por :

```
#!/bin/bash
IFS=":"
echo "$@"
```

10. Escribe el parámetro 1 y el 11

```
#!/bin/bash
echo $1
shift 10
echo $1
```

Versión mejorada con comprobación de los parámetros:

```
#!/bin/bash
clear
if [ $# -gt 0 ]; then
    echo "El parámetro 1 es $1"
    shift 2
    if [ ${#9} -gt 0 ]; then
        echo "El parámetro 11 es $9"
    else
        echo "El parámetro 11 no existe"
    fi
else
    echo "No hay parámetros"
fi
```

11. Si el parámetro es un fichero escribe su propietario

```
#!/bin/bash
clear
if [ $# -gt 0 ]; then
    if [ -f $1 ]; then
        PROPIETARIO=$(ls -l $1|tr -s ' ' ':'|cut -d ':' -f3)
        echo "El propietario de $1 es $PROPIETARIO"
    else
        echo "$1 no es un fichero"
    fi
else
    echo "No hay parámetros"
fi
```

12. Comprueba si el parámetro es un directorio
13. Si no existe el directorio indicado como parámetro, lo crea
14. Calcula el horóscopo chino según el año de nacimiento

```
#!/bin/bash
clear
read -p "Año de nacimiento (4 cifras):" AN
let RESTO=$AN%12
case $RESTO in
  0) HOROS="Mono";;
  1) HOROS="Gallo";;
  2) HOROS="Perro";;
  3) HOROS="Cerdo";;
  4) HOROS="Rata";;
  5) HOROS="Buey";;
  6) HOROS="Tigre";;
  7) HOROS="Conejo";;
  8) HOROS="Dragón";;
  9) HOROS="Serpiente";;
  10) HOROS="Caballo";;
  11) HOROS="Cabra";;
  *) HOROS="Caso imposible";;
esac
echo "Tu horoscopo chino es $HOROS"
```

15. Ejemplo de bucle for

```
#!/bin/bash
for DIA in lunes martes miercoles jueves viernes; do
  echo Dia en proceso $DIA
done
```

cuya salida es:

```
Dia en proceso lunes
Dia en proceso martes
Dia en proceso miercoles
Dia en proceso jueves
Dia en proceso viernes
```

16. Buscar todas las canciones MP3 del disco y mostrar el nombre

```
#!/bin/bash
# Reproductor MP3
clear
CONJUNTO= $(find / -iname "*mp3" 2>/dev/null)
CONJUNTO=${CONJUNTO}Salir"
select MP3 in $CONJUNTO ; do
  if [ $MP3="Salir" ] ; then
    break
  fi
  echo "Reproduciendo...$MP3"
  mpg321 $MP3 &> /dev/null
done
```

3.9. AWK

AWK toma una entrada de datos, examina los campos de la entrada (internamente reconoce los campos de los datos) y procesa esos campos. AWK procesa línea a línea. Compara patrones con esas líneas y lleva a cabo acciones asociadas en las líneas cuyos patrones coincidan. Las líneas sin patrones coincidentes no sufren ninguna acción.

Por tanto, necesitamos un patrón y una acción.

Si no se especifica ningún archivo, toma la entrada por defecto.

awk es capaz de remplazar a otros comandos simples (como cut, tr, sed,...), pero es mucho más potente que todo esto, ya que permite realizar condiciones, bucles, tomas de decisión, etc, de manera similar a un lenguaje de programación.

3.9.1. sintaxis AWK

comando: `awk [opciones] [-Fs] regla-orden [-v var=valor] [archivo(s) a procesar]`

F indica que use el separador de campos indicado por s. v me permite definir variables e inicializarlas a un valor.

Si trabajamos desde línea de comandos, es recomendable encerrar todo entre comillas simples.

PE:

```
$ awk -F: /^-d/ {print $9}' fichero.txt
```

Esto nos establece la separación de datos en : y define el patrón como una regex (línea que comienza por -d), e imprime el campo 9 de las líneas coincidentes, de fichero.txt. Notar que los campos comienzan a contar en 1, y la variable \$0 indica toda la línea.

Si queremos indicar más órdenes, deberemos separar las mismas por ;

PE: Procesar la salida de un comando.

```
$ls -l /home/antonio |awk '-v s=0 {s+=$5; print s}'
```

esto toma la salida del ls, pone una variable s a 0, actúa sobre todas las líneas (porque no hay patrón), toma el tamaño de cada archivo (el campo 5), e imprime los tamaños de los ficheros acumulando.

Para evitar que salga el tamaño de cada fichero, y sólo imprima al final del todo, empleamos los patrones especiales BEGIN y END:

```
ls -l /home/antonio | awk 'BEGIN {s=0}; /^-r/ {s+=$5}; END {print s}'
```

la parte que corresponde a BEGIN (hasta el siguiente ;) se ejecuta una sola vez al comienzo, y la orden establece una variable `s` a 0. Luego procesa todas las líneas de ficheros regulares, y acumula el tamaño en `s`. Cuando ha terminado con todas las líneas, lanza la parte END, que imprime el valor de `s` (con los tamaños acumulados).

Para evitar tener que crear líneas muy largas, podemos indicar un fichero con las reglas predefinidas mediante la opción `-f`

```
awk -f fichero_ordenes
```

y el fichero de ordenes contiene las órdenes, cada una en una línea (un salto de línea equivaldría a un ; en la línea larga).

```
BEGIN {s=0}
/^-r/ {s+=$5}
END {print s}
```

Si guardo esto en `ej1.awk`, puedo lanzarlo como:

```
ls -ls /home/antonio | awk -f ej1.awk
```

Los comentarios en los archivos de comandos `awk` se ponen con `#`.

3.9.2. Tipos de patrones

- BEGIN se ejecuta una sola vez al comienzo del proceso
- END se ejecuta una sola vez al final de todos los procesos
- `/regexp/` se escribe entre barras y dentro lleva una expresión regular, que si coincide con la línea actual se usa dicha línea para aplicarle el proceso indicado.
- expresión relacional comprueba si alguno de los campos cumple una condición. (Podemos emplear `&&` y `||`). PE:

```
$5 > 5000 && $9 != 'hola.txt'
```

- concordancia con regexp Se emplea un operador especial `~`, y permite buscar una expresión regular en un campo o variable ya definida. PE:

```
$9 ~/^[A-Z]/ {print $9}
```

Busca los campos 9 que comienzan por A-Z y los imprime. OJO, no es la línea que comienza por A-Z.

- No concordancia con regexp: lo mismo, pero para los campos que NO concuerden con la condición. En este caso, el operador es !~

PE: buscar los directorios de un usuario que empiecen por mayúsculas.

```
$ ls -l /home/user | awk '$1 ~ /^d/ && $9 ~ /^[A-Z]/ {print $9}'
```

Las líneas cuyo campo 9 comience por d (directorio) y el campo 9 sea mayúscula, imprime las líneas.

3.9.3. Operadores

Operadores de asignación disponibles:

```
=, +=, -=, *=, /=, %=, ^=
```

Operador ?, que al igual que en c es equivalente a un if:

```
expr1 ? expr2 : expr3
```

En formato más legible:

```
{ if (expression)
  orden1
  else
    if (expression)
      orden2
  else
    .....
}
```

Si necesitamos más de una orden en cada rama, tenemos que ponerlas entre llaves. Otros operadores disponibles son:

```
&&, ||, !
~, !~
> < >= <= <>
+ - * / % ^
$
++ --
{,}
```

3.9.4. Variables internas de AWK

- FILENAME el nombre de fichero procesando actualmente. Si no estamos procesando un fichero (PE, entrada estandar), muestra un - (guión).
- FNR es el n° de línea de la entrada que se está procesando.
- FS Field Separator, almacena el separador de campos. Podemos manipularlo como queramos, PE en un BEGIN
- NF n° de campos de la línea que se está procesando. Si lanzo la orden:

```
{print $NF}
```

Me mostrará el contenido del último campo de la línea (lógico, dado que NF guarda el total).

- NR n° de líneas que han sido procesadas. PE:

```
{a[&NR]=$0}
```

en esta línea hemos creado un array donde cada elemento es cada una de las líneas que hemos procesado. Va incrementando en cada línea procesada.

- OFS Separador de campos de salida.
- ORS Separador de líneas a la salida
- OR Separador de líneas a la entrada
- \$0 Toda la línea actual
- \$n campo específico de la línea actual. Si el campo no existe, el campo estará vacío y no mostrará nada (pero no dará error).

3.9.5. Arrays y Matrices

Podemos crear y trabajar con arrays y matrices en AWK. Los arrays normales no necesitan mucha más explicación, trabajan igual que en C. No hace falta predefinir los tipos ni los tamaños de los array. Los array y matrices empiezan a contar en 0.

```
A[33]="Enero"
```

Pero tenemos otro tipo de array denominado asociativo, donde el índice del array no tiene que ser numerico. Podemos indexar por cadenas de texto.

```
Animales["Pajaro"]=25
Animales["Perro"]=3
```

OJO: cuando trabajamos con arrays con elementos de distintos tipos (entero, flotantes, texto...) cuando introducimos un elemento de otro tipo todos los elementos del array se convierten en el tipo "mas grande" (es decir, si tengo un array de enteros y almaceno un flotante, todos los elementos serán flotantes. Si tengo enteros y meto una cadena, todos los enteros serán cadenas.).

Podemos unir bucles y variables en los arrays:

```
usuario[$username]=...
```

3.9.6. Funciones internas

Veremos principalmente funciones matemáticas y de manejo de cadenas.

Matemáticas

1. `int(x)` devuelve la parte entera de `x`.
2. `rand` nos devuelve un n° aleatorio entre 0 y 1, pero nunca es 0 ni 1. Siempre será "0,algo". `itemsrand(x)` nos genera una nueva semilla para los n°s aleatorios, y en la `x` indicamos el n° de semilla que queremos emplear. Si usamos el mismo n° de semilla dos veces, repetiremos la secuencia de numeros aleatorios generados.
3. `sqrt(x)` devuelve la raiz cuadrada de `x`.

Por ejemplo:

```
n=int(rand()*49)+1
```

genera n°s aleatorios entre 1 y 49. (nunca llega a ser cero, porque sumamos 1).

```
nota=int(rand()*11)
```

genera notas entre 0 y 10 Da ceros, porque los valores de `rand` muy pequeños (inferiores a 0,0algo) al multiplacarlos por 10 siguen teniendo la parte entera 0. (nunca llega a 11, porque la nota mayor será 10.99999...).

3.9.7. Manejo de cadenas

1. `index(s,t)` devuelve la posición de la cadena "`t`" en la cadena "`s`". Si `t` no está en `s`, devuelve 0.

```
t=index("cadena","d")
```

2. `length(s)` devuelve la longitud de la cadena `s`.

```
{print length($1)}
```

3. `match(s,r)` devuelve la primera posición de la cadena `s` que verifica la regexp `r`, siendo válidas expresiones regulares extendidas de `egrep`. Si no encuentra, devuelve 0.

```
match("bEzmi",/[A-Z]/)
```

devuelve la posición 2, que es la primera mayúscula.

4. `split(s,a,r)` genera un array `.a` guardando en cada posición del array un elemento resultante de dividir la cadena `s` en tantas partes como resulte de aplicar el patrón `r` como divisor.

```
ruta="/home/usuario/awk/mis_ejemplos"
split($ruta,director,"/")
```

genera un array "director" que contiene lo siguiente:

- `director[0]=home`
- `director[1]=usuario`
- `director[2]=awk`
- `director[3]=mis_ejemplos`

5. `substr(s,i,n)` devuelve una cadena de `n` caracteres de la cadena `s`, comenzando a partir de la posición `s` (de la cadena `s`).

```
VAR="Examen de Linux"
E=substr($VAR,1,6)
L=substr($VAR,11,)
```

En `E` tendremos `.Examen`. En `L` tenemos toda la cadena desde la posición 11 hasta el final

3.9.8. Ejercicios

1. Ejemplo: buscar el mayor ID de usuario en `/etc/passwd`. En el fichero `ej2.awk` guardamos:

```
BEGIN {FS=::;x=0}
$3 > x {x = $3}
END {print x}
```

y lo lanzamos con:

```
awk -f ej2.awk /etc/passwd
```

y en un fichero `.sh` podemos poner:

```
#!/bin/bash
clear
awk -f ej2.awk /etc/passwd
```

2. Escribir un programa awk que nos informe sobre la cantidad en MB que está usando un usuario. Habrá que comprobar que el usuario exista
3. crear (a mano) un archivo con la siguiente estructura:

```
Nombre Ape1 Ape2 Nota
```

separando los campos por espacios o tabuladores, y siendo Nota numérica.

hacemos un programa en awk que tomando como entrada ese archivo me genere otro archivo ordenado por apellidos y nombre en el cual se ha sustituido la nota numérica por la nota escrita.

4. Programa que muestre el tamaño medio de los archivos regulares de todo el directorio personal de un usuario.

Capítulo 4

Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a

licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”,

“Dedications”, “Endorsements”, or “History”). To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.