Apuntes FS

A tener en cuenta:

- Para mostrar los parámetros que toman las órdenes se usará [] para indicar que pueden ser un [directorio], un [archivo], etc...
- Cuando se pongan varias órdenes o parámetros seguidos de una barra /, significa que esas órdenes o parámetros o bien tienen el mismo efecto si son órdenes o bien son los posibles parámetros que puede tomar una orden.
- Recordemos que las opciones que puede tomar una orden comienzan con un guion `-opcion`
- En cualquier orden que implique el uso de opciones y el de argumentos, se escribirán primero las opciones y luego los argumentos. Ejemplo: `ls -l [directorio]`

Práctica 1: Órdenes básicas

1. Órdenes básicas

En esta práctica se ven las órdenes básicas más comunes.

La orden `man/info/help <orden>` muestra un mensaje de texto explicando el funcionamiento de la orden en cuestión.

A continuación una lista con las órdenes más frecuentes con **archivos** y **directorios**:

- `ls [directorio]` Lista los contenidos de un directorio. La opción `-a` lista los archivos del directorio actual, incluso los ocultos. La opción `-l` lista en formato largo, y la opción `-R` lista subdirectorios recursivamente, además del actual.
- `cd [directorio]` Cambia de directorio de trabajo. `.` hace referencia al directorio actual y `..` al directorio padre.
- `pwd`Hace referencia al camino absoluto del directorio actual (desde la carpeta home).
- `mkdir [nombre]` Crea un directorio llamado nombre.
- `rmdir [nombre]` Borra el directorio nombre (debe estar vacío)
- `cat [archivo(s)]` Puede hacer esto: mostrar contenido de archivos, concatenarlos con `[origen] > [destino]`, copiar archivos, ...
- `cp [archivo1] [archivo2]`Copia el contenido de archivo1 en archivo2. Si no existe, se crea.
- `mv [archivo(s)]/[directorio(s)] [destino]` Mueve el archivo o directorio al destino.
- `file [archivo(s)]` Muestra el tipo de archivo.
- `more [archivo(s)]` Muestra poco a poco el contenido del archivo.
- `rm [archivo(s)]/[directorio(s)]` Borra archivos o directorios con contenido.
- `touch [archivos(s)]`Crea los archivos en la ruta especificada. Si existen, se actualiza la fecha.

- `clear` Borra el contenido del terminal actual.
- `tail -numero_de_lineas [archivo(s)]` Muestra las últimas número_de_lineas líneas. Por defecto muestra 10.
- `head -numero_de_lineas [archivo(s)]` Igual que `tail` pero muestra las primeras lineas.
- `sort -criterio [archivos]` Ordena según criterio el contenido de los archivos dados como argumentos. Sin argumentos ordena alfabéticamente, usando `--reverse` ordena alfabéticamente al revés.

2. Permisos

Hay tres permisos, de lectura `(r)`, de lectura `(w)`, y de ejecución `(x)` (`-` indica ausencia de permiso). Con la orden `ls -l` se pueden ver los permisos de cada archivo. Mostrará 9 bits, donde el primero indica si es directorio `(d)` o un archivo `(-)`, y los siguientes 9 indican los permisos de lectura, escritura o ejecución en grupos de tres, empezando en el usuario, luego el grupo de usuarios y por último el resto.

3. Metacaracteres de archivo

Cuando aplicamos órdenes a varios archivos, es más fácil usar estos metacaracteres ya que nos facilitan el trabajo.

- `?` Representa cualquier carácter simple en la posición donde se inserte.
- `*` Representa cualquier secuencia de 0 o más caracteres.
- `[ab]` Representa cualquier carácter simple de los contenidos en la lista. también se puede especificar un rango de caracteres usando un guión entre el primero y el último.
- `{palabra1,palabra2}` Sustituyen las palabras que aparezcan entre medias. Hay que poner comas entre ellas. En el ejemplo se mostrarán todos los archivos que contengan palabra1 o palabra2.
- `~` Se usa para abreviar el camino absoluto del directorio del usuario /home/nombre_del_usuario.

Práctica 2 Permisos y redirecciones

1. Modificación de los permisos de acceso a archivos

Vamos a usar la orden `chmod`. Pondremos primero el grupo al que se le aplica el permiso, luego si se lo quitamos o añadimos (+ ó -), y por último el permiso en sí. Todo ello seguido (sin espacios) y luego el nombre del archivo
Los grupos de usuarios son:

- u : propietario
- g : grupo
- o : resto de usuarios
- a : todos los grupos
  Si se lo queremos aplicar a varios grupos podemos poner más letras seguidas. Por

ejemplo, para darle permiso de ejecución al propietario y al grupo sobre archivo, podemos usar:

```
chmod ug+x [archivo]
```

También podemos tomar los 8 bits que representan los permisos, siendo 1 estar activo y 0 no estarlo, y usar `chmod 777 [archivo]` con el número en octal que represente los permisos que queremos.

2. Metacarateres de redirección

Normalmente, la entrada y salida de Linux están asignadas por defecto (la entrada al teclado, la salida estándar y la de errores la pantalla). Sin embargo, puede pasar que queramos que esta salida u entrada se redirija hacia archivos. Para eso usamos los metacaracteres de dirección:

- `< [nombre]` Redirecciona la entrada de la orden para que la obtenga del archivo nombre.
- `>` [nombre]` Redirecciona la salida hacia el archivo nombre. **CUIDADO: lo sobreescribe**.
- `&> [nombre]` Igual que el anterior pero combina la salida estándar y el error. También lo sobreescribe.
- `>> [nombre]` Funciona igual que `>` pero no sobreeescribe (lo añade al final).
- `&>> [nombre]` Funciona igual que `&>` pero añadiéndolo al final del archivo.
- `|` Crea un cauce entre dos órdenes. La salida de la primera orden se usa como entrada para la siguiente.
- `|&` Hace lo mismo que la anterior pero también usa la salida de error.

3. Metacaracteres sintácticos

Sirven para combinar varias órdenes en una sola orden lógica.
-`;` Ejecuta la órdenes secuencialmente, de forma 'separada'.

- `( )` Se usa para separar órdenes al usar otros metacaracteres.
- `&&` Separador entre órdenes. La orden depués del metaracter sólo se ejecuta si la anterior ha tenido éxito.
- `||` La orden de después solo se ejecuta si la anterior ha fallado.

NOTA: En esta práctica se ve la orden `wc`, que muestra por pantalla en número de líneas, caracteres (`-m`) o palabras de un archivo.

Pŕactica 3: Variables, alias, órdenes de búsqueda y guiones

1. Variables

Hay dos tipos de variables, las de entorno, comunes a todos los shells, y las locales, visibles solo en el shell donde se definen. Para ver las variables locales creadas se usa `set`.

Para obtener ayudan del sistema sobre variables usamos `help variables`.

Las variables pueden contener cadenas, números, constantes y vectores o arrays.

Creación y visualización

Para crearlas, hacemos `[nombre]=5` para crear una variable nombre que valga 5. Para visualizar la variable, usaremos la orden `echo $[nombre]`. **Si no ponemos $ no impriremos el valor de la variable.**.

Para crear un array usaremos `[nombre]=(azul verde rojo)`. Para ver a la posición x, usamos `echo &{[nombre][x]}`.

Para eliminar una variable usaremos `unset [nombre]`. `declare [nombre]` creará una variable con determinados atributos (si usamos `-i` se crea un número, `-r` es solo lectura, `-a` se crea un vector, ...).

Exportar variables

Exportar variables sirve para que se puedan usar fuera del ámbito local del shell actual. Se pueden exportar usando `export [nombre]`, y se pueden exportar y declarar con `export [variable]=valor`.

Las comillas en las órdenes

En el bash se pueden sustituir órdenes, que permite ejecutar órdenes y que el resultado se pueda asignar a una variable.

Para realizar la sustitución, se puede usar `$(orden)` o bien ` `orden` `

Asignar resultados de órdenes a variables

Usando la sustituciñon de órdenes anterior, usando `[variable]=$(orden)`.

Para asignar a una variable el resultado de una operación, debemos usar `expr` seguida del nombre de la variable o de números.

2. La orden empotrada printf

Se usa para imprimir un mensaje por pantalla con el formato especificado. La sintaxis es `printf formato [argumentos]`.

Estos son los caracteres de escape:

- `\b` Espacio atrás.
- `\n` Nueva línea.
- `\t` Tabulador.
- `\'` Comilla simple.
- `\\` Barra invertida.

- `\0n` n es un número en octal que representa un carácter ASCII.

Estos son algunos códigos de formato:

- `%` Número con signo.
- `%d` Número en coma flotante (decimal) sin notación exponencial.
- `%q` Entrecomilla una cadena.
- `%s` Muestra una cadena sin entrecomillar.
- `%x` Muestra un número en hexadecimal.
- `%o` Muestra un numero en octal.

Ejemplos útiles de esta orden:

```
$ printf “%10d\n” 25 # Imprime el número en una columna de 10 caracteres

$ printf “%-10d\n” 25 # Igual que el anterior pero justificado a la izquierda

$ printf "%10.3\n" 25.3 # En este caso la parte entera es el número de posiciones que ocupa el número y la parte decimal las cifras decimales con las que se quiere imprimir.
```

3. Alias

Los alias sirven para cambiar el nombre de una orden o cambiar el comportamiento de dicha orden.

Para crear un alias se usa `alias`, para eliminarlo `unalias`. Por ejemplo para cambiar el nombre de la orden `ls` por `dir` hacemos `alias dir='ls -l'`. Para ver los alias configurados usamos `alias` sin argumentos.

4. Órdenes de búsqueda: find y grep, egrep, fgrep

Orden Find

Se utiliza para buscar archivos que cumplan los requisitos especificados. Su sintaxis es `find [lista_directorios] expresiones`. Veamos algunos de estos criterios o expresiones:

- `-name [expresion]` Se usa para buscar por nombre. En expresion pueden haber expresiones con metacaracteres.
- `-atime [n_dias]` Si se usa con `7`, se muestran los archivos a los q se accedió hace 7 días. Si se usa con `+` o `-` delante del número muestra los archivos a los que se accedió hace más de x días o en los últimos x días, respectivamente.
- `-size [tamaño]` Se usa para buscar por tamaño, siendo tamaño un número, precedido con un `+` o `-` según se quiera el mismo o mayor tamaño o el mismo o menor tamaño,

respectivamente.
- Usando el operador `-o` entre dos criterios podemos buscar archivos que cumplan un cirterio u otro.
- Añadir `-exec [orden]` después de los criterios hará que se aplique la orden especificada a ese conjunto de archivos.

Órdenes grep, egrep y fgrep

La orden `grep` busca cadenas en archivos que toma como entrada e imprime en la salida las líneas de esos archivos donde se encuentren las cadenas.
Su sintaxis es `grep opciones patrón [archivos]`. `patrón` puede ser una cadena, pero también admite expresiones regulares (ver práctica 4).

Algunas opciones son:

- `-x` Localiza líneas que coincidan totalmente, desde el principio hasta el final de línea, con el patrón especificado.
- `-v` Selecciona todas las líneas que no contengan el patrón especificado.
- `-c` Produce solamente un recuento de las líneas coincidentes.
- `-i` ignora las distinciones entre mayúsculas y minúsculas.
- `-n` añade el número de línea en el archivo fuente a la salida de las coincidencias.
- `-l` selecciona sólo los nombres de aquellos archivos que coincidan con el patrón de búsqueda.
- `-e` especial para el uso de múltiples patrones e incluso si el patrón comienza por el carácter (-).

Hay dos variantes de `grep`: `fgrep`, que acepta solo cadenas simples y no expresiones regulares, y `egrep`, que permite expresiones regulares más complejas.

5. Guiones (Scripts)

Un guión o script es un archivo de texto que contiene órdenes que el sistema operativo puede leer y ejecutar. Esto simplifica procesos que requieran de muchas órdenes, pues no es necesario escribir en el terminal una orden tras otra.

De forma básica, se pueden escribir las órdenes en un fichero de texto y luego ejecutarlas en terminal con `bash [archivo]`, pero para facilitar el proceso se escribirá `#!/bin/bash` al inicio de cada script para indicar al sistema operativo que es un ejecutable. Posteriormente, se activan los permisos de ejecución con `chmod` y se ejecuta el programa con `./[archivo]`.

En los scripts se aplican todos los comandos y sintaxis antes vistos, pero además tenemos otros que sólo se usan dentro del script:

- `$0` Se refiere al nombre del script.
- `$1`, `$2`, Posibles argumentos que se le pasa al script (para más de 9 hay que usar `${n}`.
- `$*` Referencia a todos los argumentos. Con comillas equivale a “$1 $2 $3 ... $n”.
- `$#` Referencia a todos los argumentos. Con comillas equivale a “$1” “$2” ... “$n”.

- `${arg:-val}` Si el argumento tiene valor y es no nulo no lo modifica, en caso contrario le asigna `val`.
- `${arg:?val}` Si el argumento tiene valor y es no nulo, sustituye por `val`. En caso contrario imprime `val` y sale del guión. Si se omite val, imprime que el argumento es nulo o no está asignado.

Para depurar los fallos de nuestro script, podemos ejecutarlo desde la temrinal con `bash opciones [nombre]` empleando estas opciones:

- `-n` Chequea errores sintácticos pero sin ejecutar el guion.
- `-v` Visualiza cada orden del guion antes de ejecutarla.
- `-x` Actúa igual que –v sólo que sustituyendo, en su caso, las variables por los valores que tienen en ese instante.

Normas de estilo

En la asignatura de FS se requerirá de la siguiente plantilla para los guiones de prácticas:

```
#!/bin/bash
# Titulo: prueba
# Fecha: 5/10/2011
# Autor:Profesor de FS
# Versión: 1.0
# Descripción: Guion de prueba
# Opciones: Ninguna
# Uso: prueba directorio
```

Práctica 4: Expresiones con variables y expresiones regulares

1. Expresiones con variables

Para poder evaluar una expresión aritmética se vio el comando `expr()`, pero también podemos usar `$((...))` ó `$[...]`. Lo que haya dentro de los paréntesis se interpreta como una operación aritmética, y si se introducen variables se sustituyen directamente por su valor sin necesidad de poner `$`.

Operadores aritméticos

Los operadores aritméticos son parecidos a los de cualquier lenguaje de alto nivel. Cabe destacar que la división trunca decimales, que la potencia es `**`, y que si se pone `variable++` se hace primero lo que se solicite con la variable y luego se incrementa pero con `++variable` primero se incrementa y luego se hace lo que se desee.

Asignación y variables aritméticas

Para asignar el resultado de una expresión a una variable se puede usar el comando `let [variable]=expresion`. Poner `"[variable]=expresion"` no cambia nada. También es equivalente usar `[variable]=((expresion))`.

Operadores relacionales

Son exactamente iguales a los del C++, solo que el operador igual puede escribirse también como `=`.

Operadores de consulta de archivos

La orden `test expresion` permite evaluar una expresión lógica y devolver 0 si es verdadera o 1 si es falsa. Se puede usar `test opciones [archivo/directorio]` para conocer la naturaleza de un archivo o directorio. Las opciones son:

- `-b` Existe y es un dispositivo de bloques.
- `-c` Existe y es un dispositivo de caracteres.
- `-d` Existe y es un directorio.
- `-e` Existe.
- `-f` Existe y es un archivo plano o regular.
- `-G` Existe y es propiedad del grupo del usuario.
- `-h/L` Existe y es un enlace simbólico.
- `-O` Existe y es propiedad del usuario.
- `-r` Existe y el usuario tiene permiso de lectura sobre él.
- `-s` Existe y es no vacío.
- `-w` Existe y el ususario tiene permiso de escritura.
- `-x` Existe y el usuario tiene permiso de ejecución (si es un fichero, equivale a que tenga permiso de búsqueda).
- `[archivo1] -nt [archivo2]` Archivo1 más reciente que archivo2, o que el primero existe y el otro no.
- `[archivo1] -ot [archivo2]` Archivo1 más antiguo que archivo2, o que el primero existe y el otro no.
- `[archivo1] -ef [archivo2]` Archivo1 es un enlace duro al archivo2, es decir, si ambos se refieren a los mismos números de dispositivo e inode.

La orden `test expresion` equivale a `[ expresion ]`, donde los huecos en blanco son necesarios.

2. Orden if/else

Esta orden permite ejecutar una orden u otra según se cumplan o no una serie de condiciones. La sintaxis es la siguiente:

```
if condición;
then
    declaraciones
```

```
[elif condición;
    then declaraciones ]...
[else
    declaraciones ]
fi
```

donde aquello que va entre corchete es opcional. La condición puede ser cualquier tipo de expresión condicional vista anteriormente, incluso una lista de ellas.

Para usar las comparaciones aritméticas, podemos usar cualquiera de los operadores vistos anteriormente, y además podemos tratar los valores numéricos como cadenas poniendo comillas.

Con los operadores `==` y `!=` se pueden comparar cadenas. Si estas se comparan con variables y además tienen espacios, hay que poner la variable entre comillas. Además se usan corchetes con espacios de por medio para este tipo de comparaciones.

```
if [ $valor != "adiós" ]; then echo sí; else echo no; fi
```

Al usar órdenes dentro de un `if`, si éstas se ejecutan correctamente se ejecutarán las declaraciones del `then`, en otro caso las del `else` o saltará a la siguiente instrucción.

Cuando se emplea el doble paréntesis, no es posible utilizar los operadores `-eq`, `-ne`, `-le`, `-lt`, `-ge` y `-gt`.
ATENCIÓN: cuando se use `=` con una variable y un número dentro de un if, se está asignando el valor, no comparando.

3. Expresiones regulares

Son patrones que permiten describir conjuntos grandes de cadenas y que se utilizan para búsqeudas dentro de una caedna o un archivo. A continuación se muestran una serie de patrones que se usan en estas expresiones. Cabe destacar que para comparar con el símbolo `*` hay que usar comillas simples. Para los siguientes caracteres, si se quiere que tengan efecto en la expresión regular tienen que ir precedidos por `\`.

- `\` Barra de escape. Si se quiere hacer referencia a este carácter, debe ir precedido de él mismo y entre comillas simples.
- `.` Cualquie carácter en esa posición si se utiliza entre otros patrones. Si se usa solo representa cualquier cadena.
- `(  )` Un grupo: los caracteres que se pongan entre medias serán considerados conjuntamente como si fueran uno solo.
- `?` El carácter o grupo al que sigue puede aparecer una vez o no aparecer ninguna.
- `*` El carácter o grupo al que sigue puede aparecer varias veces seguidas o no aparecer ninguna.

- `+` El carácter o grupo previo debe aparecer una o más veces seguidas.
- `{n}` El carácter o grupo previo debe aparecer exactamente n veces seguidas.
- `{n,m}` El caracter o grupo debe aparecer n veces como mínimo y m veces máximo.

En estos caracteres no hay que poner la barra de escape:

- `[ ]` Una lista de caracteres que se tratan uno a uno como caracteres simples. Si se pone `^` al principio, significa los caracteres que no estén en esa lista.
- `-` Rango de caracteres si el guion no es el primero o último de esa lista. Si el guion aparece primero o último de la lista, se trata como a él mismo. Los rangos de caracteres van alfabéticamente intercalando mayúsculas (AaBb...), los numéricos de menor a mayor. Se puede poner el guión al inicio o al final para que la lista no quede acotada (`[-m]` significa las letras de la 'a' a la 'm').
- `^` Indica inicio de una línea. Usado al inicio de lista representa los caracteres que no sean los que le siguen. Situando un caracter depués de `^`, filtrará las líneas que empiecen por ese carácter.
- `$` Indica final de línea. Situado un caracter antes, filtra las líneas que comiencen por ese carácter.

Los siguientes patrones deben usarse con comillas simples o dobles:

- `\b` Final de una palabra.
- `\B` No está al final de una palabra.
- `\<` Comienzo de una palabra.
- `\>` Final de una palabra.
- `\|` Operador OR para unir dos expresiones regulares, de forma que la resultante represetna a cualquier cadena que coincida con al menos una de las subexpresiones (si se usa con `grep`, hay que acompañarla de la opción `-E`).