Comandos Prácticas Fundamentos del Software

- mkdri *nombre*: Comando para crear directorio
- rmdir *nombre*: Comando para eliminar directorio
- touch *nombre.extensión*: Comando para crear ficheros
- nano: Abrir fichero de texto. Recmiendo usar gedit admin://
- mv *fuente destino*: Mover la fuente al destino
- cp *archivo1 archivo2*: Copiar archivo 1 a 2
- ls: Muestra el contenido del directorio. Puede combinarse con los siguientes comandos

Comando	Función
-a	Lista los elmentos del directorio actual
-c	Muestra el contenido en multicolumna
-l	Lista el contenido en formato de permisos
-r	Muestra el contenido en orden inverso
-R	Lista subdirectorios consecutivos
-t	Lista por fecha de modificación

- cat *fichero*: Muestra el contenido del fichero
- ~: Muestra el camino absoluto del directorio actual y hace referencia al home/
- ?: Representa cualquier carácter
- *: Representa culaquier secuencia de caracteres
- []: Representa un rango de caracteres

```
#Ejemplo
Supongaos un directorio que contiene los ficheros Sesion.1 Sesion.2
Sesion.hola y Doc
    $ls -l Sesion.?                              muestra muestra no muestra
no muestra
    $ls -l Sesion.*                              muestra muestra   muestra
no muestra
    $ls -l [S]*                                  muestra muestra   muestra
no muestra

#[S]* es equivalente a decir muestra aquellos que contengan el caracter 'S'
```

- pwd: Camino absoluto
- head -*numero*: Muestra las n-ésimas primeras líneas
- tail -*x*: Muestra las n-ésimas últimas líneas
- chmod: Coando para permisos de archivos

Comando	Usuarios	Derechos	Permiso
	u (*usuario*)		r (*read*)
chmod	g (*grupo*)	+ (*Dar permisos*)	w (*write*)
	o (*otros*)	- (*Quitar permisos*)	x (*ejecutar*)
	a (*all*)		

- ' > ' *fichero*: Sobrescribir fichero
- ' >> ' *fichero*: Añadir al final del fichero
- ' < ' *fichero*: Redirecciona la entrada para que la obtenga el fichero
- |: Cauce de órdenes
- (): Prioridad
- &&: Separador de órdenes

```
orden1&&orden2 Significa que 1 implica 2
```

- ||: Separador de órdenes

```
orden1||orden2 Significa que 1 negado implica 2
```

- ;: Separador de órdenes para que se ejecuten secuencialmente

```
#Ejemplo2
$ ls -la | head -7  Muestra los 7 primeros archivos del directorio
```

- Veamos como crear variables:

```
$ numero=1
$ echo numero
1
```

- echo: Sirve para mostrar el contenido de una variable
- Veamos como crear un vector:

```
$ colores=(rojo azul verde)
$ echo ${colores[0]}
rojo
```

- Veamos como crear variables con determinados atributos:

Comando	Atributos	Fichero
	-i (*tipo numérico*)	
	-p (*ver propiedades*)	
$declare	-r (*solo lectura*)	*nombre variable*
	-a (*matriz, vector o lista*)	
	-x (*exportable*)	

- export *variable*: Exportar variable
- bash: Abrir nueva estancia, es decir, una nueva terminal
- echo "*frases*": Para imprimir por pantalla
- echo "*frase ... $comando*": Se ejecuta el comando
- echo "*frase ...* ``": Se ejecuta el comando
- Como asignar el resultado de una orden a una variable:

```
$variable = `orden`
#Por ejemplo: $listadearchivos = `ls`
```

- Como asignar el resultado de una operación:

```
$variable = $operacion
```

- La orden echo puede tener comportamientos distintos según el sistema UNIX que estemos utilizando, luego es más conveniente usar la orden printf:

Comando	Comandos	Comandos	Argumentos
	\b (*Espacio detrás*)	%d (*Número con signo*)	
	\n (*Nueva línea*)	%f (*Coma flotante*)	
printf	\t (*Tabulador*)	%q (*Entrecomillado*)	argumento
	\ ' (*Carácter comilla simple*)	%s (*Sin entrecomillar*)	
	\ \ (*Carácter barra invertida*)	%x (*Número en hexadecimal*)	
	\0n (*Número en octal*)	%0 (*Número en octal*)	

- Asignar y desasignar alias a un comando:

```
$alias variable = `comando`
$unalias variable
```

- find *directorio [expresiones]*: Buscar un directorio con determinadas condiciones. Formas de usar find:

- -name: Por nombre del archivo

```
$find -name nombredeseado
```

- -atime: Por fecha de acceso

```
$find -atime 7  #Hace 7 días
$find -atime -2 #Hace menos de dos días
$find -atime +2 #Hace más de dos días
```

- -type: Por tipo de archivo

```
$find -type *.txt
```

- -size: Por tamaño

```
$find -size 100     #Archivo de tamaño 100
$find -size +100    #Archivo de tamaño más de 100
$find -size -100    #Archivo de tamaño menos de 100
```

- -excec: Permite añadir una orden a los archivos localizados

```
#Ejemplo
$find -atime +100 -excec rm{}
```

- -print: Visualiza los nombres de camino a cada archivo lozalizado

- grep *sentencia*: Permite buscar cadenas de archivos utilizando patrones para especificar dichas cadenas. Se puede combinar con los siguientes comandos:

Comando	Función
-x	Localiza las líneas que coinciden totalmente
-v	Selecciona todas las líneas que no contengan el patrón especificado
-c	Recuento de las líneas coincidentes
-i	Ignora la distinción entre mayúscula y minúscula
-n	Añade el número de línea e el archivo

- Llamaremosscript o guión de shell a un archivo de texto que contiene órdenes de shell. Veamos algunos comandos de script:
  - $0: Nombre del guon
  - 1, . . . ,9: Argumentos que se pueden facilitar al llamar al guión
  - ${n}, n>9: Arguentos que se pueden facilitar al llamar al guión
  - $*: Nombre del guión y todos los argumentos dados
  - $@: Igual que el anterior
  - $#: Contiene el número de argumentos

    - ${arg: -val}: Si el argumento no es nulo continua con su valor, en caso contrario le asigna val
    - ${arg: ?val}: Si el argumento tiene valor y es no nulo, sustituye a su valor, en caso contrario imprime val y sale el guión
- Estructura:

```
#!/bin/bash
#Título
#Fecha
#Autor
#Versión
#Descripción
#Opiones
#Uso
--------------------------------Comandos---------------------------------
```

- $((...))$ o [...]: Situación aritmética. Lo que pongas en los puntos suspensivos será interpretado como una operación aritmética.
- Veamos los operadores aritméticos de shell:

Operador	Descripción
+ -	Suma y resta, o más unario y menos unario
* / %	Multiplicación, división(trucando decimales) y resto de la división
**	Potencia
++	Incremento en una unidad. Si se usa como prefijo (++variable) primero aumenta el valor de la variable y después se opera con ella. Si se usa como sufijo (variable++) primero se opera con la variable y luego se incrementa su valor
--	Decremento de la unidad. Su uso es análogo al de ++
( )	Agrupación para evaluar conjuntamente
,	Separador entre expresiones con evaluación secuencial
=	Asigna a una variable el resultado de evaluar una expresión (x=expresión)
+= -= *= /= %=	x*operacion*=y equivale a x=x*operacion*y

- Por lo general, bash trabaja con enteros, por lo que si queremos ver expresiones decimales debemos usar el comando bc con la opción -l:

```
#Ejemplo
$ echo (6/5) |bc -l
1.20000000000000
$ echo (6/5)
6/5
```

- let: Otra forma de asignar el valor de una expresión a una variable entera

```
$let variableEntera=expresión
```

- Veamos ahora los operadores relacionales:

Operador	Descripción
A = B A == B A -eq B	A es igual a B
A != B A -ne B	A es distinta de B
A < B A -lt B	A es menor que B
A > B A -gt B	A es mayor que B
A <= B A -le B	A es menor o igual que B
A >= B A -ge B	A es mayor o igual que B
! A	A es falsa. Representa el operador NOT
A && B	A es false y B es verdadera. Representa el operador AND
A || B	A es verdadera o B es verdadera. Representa el operdador OR

- test *expresión*: Evalua una expresión y devuelve 0 en caso de que sea verdadera y devuelve 1 en caso de que sea falsa. Se puede combinar con las siguientes expresiones:

Operador	Descripción: El resultado se evalua como verdadero si...
-b *archivo*	*archivo* existe y es un dispositivo de bloques
-c *archivo*	*archivo* existe y es un dispositivo de caracteres
-d *archivo*	*archivo* existe y es un directorio
-e *archivo*	*archivo* existe
-f *archivo*	*archivo* existe y es un archivo plano o regular
-G *archivo*	*archivo* existe y es propiedad del mismo grupo del usuario
-h *archivo*	*archivo* existe y es un enlace simbólico
-L *archivo*	Igual que anterior
-O *archivo*	*archivo* existe y es propiedad del usuario
-r *archivo*	*archivo* existe y el usuario tiene permiso de lectura sobre él
-s *archivo*	*archivo* existe y es no vacío
-w *archivo*	*archivo* existe y el usuario tiene permiso de escritura sobre él
-x *archivo*	*archivo* existe y el usuario tiene permiso de ejecución sobre él, o es un directorio y el usario tiene permiso de búsqueda sobre él
archivo1 -nt *archivo2*	*archivo1* ha sido modificado más recientemente que *archivo2* o si *archivo2* existe y *archivo1* no
archivo1 -ot *archivo2*	*archivo1* es más antiguo que *archivo2*, según fecha de modificación, o si *archivo2* existe y *archivo1* no
archivo1 -ef *archivo2*	*archivo1* es un enlace duro al *archivo2*, es decir, se refieren a los mismos números de dispositivos

- La orden test *expresión* es equivalente a [*expresión*]
- if/else: Ejecuta una lista de declaraciones dependiendo de si se cumple cierta condición o no. Su sintaxis es:

```
if condición;
    then declaraciones
else
    declaraciones

elif condició;          #Se utiliza para varios bloques de condiciones
    then declaraciones
else
    declaraciones

fi
```

```
#Para la evaluar la condición usamos test o [ condición ]. Para
comparaciones aritméticas podemos usar doble paréntesis (( comparación
aritmética ))

#Si se va a realizar una comparación entre cadenas de caracteres es
conveniente colocar la cadena entre comillas

#Ejemplo
$valor="hola"
if [ $valor == "hola"]; then echo si; else echo no; fi
sí
```

- Veamos ahora los bucles:
  - for: Ejecuta una lista de declaraciones un número fijo de veces.
  - while: Ejecuta una lista de declaraciones mientras se cumpla una condición.
  - until: Ejecuta una lista de declaraciones hasta que se cumpla una condición.
  - case: Ejecuta una lista de declaraciones dependiendo del valor de una variable.
- Para hacer lecturas de teclado usamos la orden read [-ers], [-a array], [-i text], [-n nchars], [-p prompt]...

```
read -p "Introduzca nombre de archivo:" $Variable1
```

- Estructura bucle for:

```
for #nombre [in lista]
do
    #declaraciones
done

#También se puede declarar como un bucle clásico de C

for((contador=***; contador<***; contador++)); do
    #declaraciones
done
```

- Estructura bucle case (Compara una variable con un patrón. Si coinciden, se ejecutan las declaraciones):

```
case #expresion in
    patron1) declaracion;;
    .
    .
    .
    patronn) declaracion;;
    *) #Representa cualquier otro caso
esac
```

- Estructura while:

```
while #expresion;
do
    #declaraciones
done

#Se suele combinar con la orden read
```

- Se puede usar la orden break para salir del bucle. break n, sinedo n un natural, indica de cuantos bucles anidados salir
- Tambien se puede usar la orde continue para pasar a la siguiente iteración, saltándose el resto de declaraciones
- Estructura bucle until:

```
until #expresion;
do
    #declaraciones
done
```

- La orden sleep n, siendo n un natural, permite hacer que el sistema tarde un tiempo antes de pasar a la siguiente iteración.

- Una de las características de bash es la posibilidad de crear funciones. Las cuales se pueden invocar más rápido que los guiones. Su estructura es la siguiente:

```
fuction nombre_fn {
    declaraciones
}
#También podemos:

nombre_fn{
    declaraciones
}
```

- El comando fuction establece la función y con el comando unset -f *nombre* la borramos. También podemos ver qué funciones tenemos definidas junto con su declaración con el comando declare -f.
- Para invocar la función dentro de un guión solamente debemos escribir el nombre seguido de los argumentos correspondientes (en caso de que los hubiese).

- Es muy importante adaptar los guiones al sistema operativo en el que se usan. Una opción en usar el comando uname, el cual nos devuelve el nombre del sistema en uso. Por ejemplo:

```
#!bin/bash
SO = `uname`
case $SO in
    Linux) echo "Estamos en Linux";;
    SunOS) echo "Estamos en SunOS";;
esac
```

- Para realizar actividades en background usamos el metacarácter & :

```
orden &
```

Hablemos ahora de la depuración y control de trabajos en bash:

La shell bash no contiene ninguna orden específica para la depuración de guiones, pero contamos con diferentes herramientas que nos pueden ayudar a suplir esta función:

- Usar la orden echo en puntos críticos del guión para seguir el rastro de las variables más importantes.
- Usar las opciones -n, -v y -x de bash

Opción	Efecto	Órdenes equivalentes
-n	Chequea errores sintácticos sin ejecutar el guión	set -n set -o noexec
-v	Visualiza cada orden antes de ejecutarla	set -v set -o verbose
-x	Actúa igual que -v pero de forma más abreviada	set - x set -o xtrace

- Se pueden invocar pasándola como opciones o bien declarándolas dentro del guión. Se activa poniendo en una línea set -n y se desactica al encontrar en una línea set +n.

- También podemos usar la orden trap, que sirve para especificar una acción a realizar cuando se recibe una señal (Mecanimos de comunicación entre procesos que permite la notificación de que ha ocurrido un determinado suceso a los procesos):

```
trap 'echo TRAZA --- contador= $contador lineaLectura= $lineaLectura' DEBUG
contador=1
while read lineaLectura;
do
    echo "Linea $contador: $lineaLectura"
    contador=$[$contador+1]
done < archivotest
echo "Finalizado el procesamiento del archivo"
```

- Al invocar el guión anterior con xtrace se obtiene:

```
+ trap 'echo TRAZA ---contador= $contador lineaLectura= $lineaLectura' DEBUG
++ echo TRAZA ---contador= lineaLectura=
TRAZA ---contador= lineaLectura=
+ contador=1
++ echo TRAZA ---contador= 1 lineaLectura=
TRAZA ---contador= 1 lineaLectura=
+ read lineaLectura
++ echo TRAZA ---contador= 1 lineaLectura= Esta es la linea 1 de archivotest
TRAZA ---contador= 1 lineaLectura= Esta es la linea 1 de archivotest

#etc
```

- Generalmente se utiliza también la señarl ERR para atrapar un error (Que se activa cuando una orden devuelve un código distinto de 0). Podemos contruir funciones para imprimir ese error. Por ejemplo:

```
function _atrapaerror{
    codigo_error=$? #Salvamos el código de error de la última orden
    echo "ERROR linea $1: la orden finalizó con estado $codigo_error"
}
```

- Por último, también podemos usar aserciones. Una función de aserción comprueba una variable o condición en un punto crítico del guión. Por ejemplo:

```
#!/bin/bash
#asercion.sh
#
_asercion()         # Si la condición es falsa, finaliza el guión
{                   # con un mensaje de error
    E_PARAMETRO_ERROR=98
    E_PATAMETRO_FALLIDA=99
    if [ -z "$2" ]; then        # Parámetros insuficientes pasados
a_asercion
        return $E_PARAMETRO_ERROR
    fi
    lineno=$2
    if [ ! $1 ]; then
        echo "Falla la aserción: \"$1\""
        echo “Archivo \”$0\”, línea$lineno” # da el nombre del guion y núm.
de línea
        exit$E_ASERCION_FALLIDA
        # else retorna y continúa con el guion
    fi
}
a=5
b=4
condicion=”$a -lt $b”       # Mensaje de error y salida del guion
                            # Ajustarla variable condicion a otra cosa y ver
qué pasa

_asercion “$condicion” $FILENO

# El resto del guion se ejecuta únicamente si la aserción no falla

echo “Si llega aquí es porque la aserción es correcta”
```

Por último hablaremos del control de trabajos en bash:

Las órdenes que se mandan a ejecutar en shell reciben el nombre de trabajos. Estas pueden estar en primer plano (foreground), en segundo plano (background) o suspendido (detenido).

- Para ejecutar una orden en segundo plano usamos &.

```
$ (sleep 5; echo "Fin de la siesta de 5 segs.")  &
[1] 10217
$ Fin de la siesta

[1]+  Hecho                   ( sleep 5; echo "Fin de la siesta" )
```

- El número entre corchetes indica el número de trabajo que acaba de ser lanzado en segundo plano y el número de al lado representa el identificador asociado al proceso.
- Los trabajos se pueden manipular usando órdenes de la shell. Estas órdenes permiten hacer referencia a un trabajo de varias formas. Una forma de hacerlo es mediante el carácter %:

Especificador	Trabajo que es denotado con dicho especificador
%	Trabajo actual (%+ y %% son sinónimos de este especificador)
%-	Trabajo previo al actual
%n	Trabajo número n
%	Trabajo cuya línea de órdenes comienza por
%?	Trabajo cuya línea de órdenes contiene

- La tabla siguiente recoge las órdenes más frecuentes de control de trabajos:

Órdenes	Descripción
jobs	Lista de trabajos activos bajo control del usuario Si se usa con -l permite visualizar la identificación del proceso
fg	Trae a primer plano un trabajo que se encuentra suspendido o en segundo plano Si se usa sin argumentos lleva el trabajo actual a primer plano
bg	Envía a segundo plano un trabajo Orden opuesta a fg y se usa exactamente igual
%	Permite cambiar el estado del trabajo
wait	Espera a la finalización de procesos en segundo plano
dosown	Suprime un trabajo de la lista de trabajos activos
kill	Envía una señal a un/os proceso/s. Por defecto, finaliza la ejecución del proceso A veces, la orden kill no puede finalizar un proceso debido a que este la ignore o realice una acción distinta a la especificada por defecto. Para ese caso se puede forzar la terminación de un procesoen estas circunstancias usando kill -9
ps	Muestra el estado de los procesos actuales del sistema. Puede usarse con: -e para mostrar la información de todos los procesos -l para mostrarlo en formato largo -u para mostrar el estado de los procesos del usuario -o para usar el formato definido por el usuario
top	Muestra los procesos en ejecución con actualización de su informa en tiempo real

Para pasar argumetnos usar orden %numero detrabajo.