vipac
www.wuolah.com/student/vipac

Documento-sin-titulo.pdf

TODAS LAS PRACTICAS DE FS RESUELTAS

1° Fundamentos del Software

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

EJERCICIOS RESUELTOS FS

Práctica 1.

Ejercicio 1.1. Cree el siguiente árbol de directorios a partir de un directorio de su cuenta de usuario. Indique también cómo sería posible crear toda esa estructura de directorios mediante una única orden (mire las opciones de la orden de creación de directorios mediante man mkdir). Posteriormente realice las siguientes acciones:

a) En Ejer1 cree los archivos arch100.txt, filetags.txt, practFS.ext y robet201.me.
mkdir Ejer1
touch arch100.txt touch filetags.txt touch practFS.ext touch robet201.me
b) En Ejer21 cree los archivos robet202.me, ejer11sol.txt y blue.me.
mkdir Ejer21
touch robet202.me touch ejer11sol.txt touch blue.me.
c) En Ejer2 cree los archivos ejer2arch.txt, ejer2filetags.txt y readme2.pdf.
mkdir Ejer2
touch ejer2arch.txt touch ejer2filetags.txt touch readme2.pdf
d) En Ejer3 cree los archivos ejer3arch.txt, ejer3filetags.txt y readme3.pdf.
mkdir Ejer3
touch ejer3arch.txt touch ejer3filetags.txt touch readme3.pdf.
e) ¿Podrían realizarse las acciones anteriores empleando una única orden? Indique cómo

Ejercicio 1.2. Situados en el directorio ejercicio1 creado anteriormente, realice las siguientes acciones:

a) Mueva el directorio Ejer21 al directorio Ejer2.
mv Ejer21 Ejer2
b) Copie los archivos de Ejer1 cuya extensión tenga una x al directorio Ejer3.
cp Ejer1/*x* Ejer3
c) Si estamos situado en el directorio Ejer2 y ejecutamos la orden ls -la ../Ejer3/*arch*, ¿qué archivo/s, en su caso, debería mostrar? los que contengan *arch*

Ejercicio 1.3. Si estamos situados en el directorio Ejer2, indique la orden necesaria para listar sólo los nombres de todos los archivos del directorio padre. cd Ejer2 ls ../

Ejercicio 2.4. Liste los archivos que estén en su directorio actual y fíjese en alguno que no disponga de la fecha y hora actualizada, es decir, la hora actual y el día de hoy. Ejecute la orden touch sobre dicho archivo y observe qué sucede sobre la fecha del citado archivo cuando se vuelva a listar.

ls -al y nos fijamos en los archivos que no esten actualizados, hacemos touch <archivo> y ejecutamos otra vez ls -al

Ejercicio 2.5. La organización del espacio en directorios es fundamental para poder localizar fácilmente aquello que estemos buscando. En ese sentido, realice las siguientes acciones

Primera academia especializada en estudios de la Facultad de Informática UCM

academia@mathsinformática.com

C/Andrés Mellado, 88 duplicado

www.mathsinformatica.com

academia.maths

91 399 45 49

615 29 80 22

Maths informática

dentro de su directorio home (es el directorio por defecto sobre el que trabajamos al entrar en el sistema): **cd ~**

a) Obtenga en nombre de camino absoluto (pathname absoluto) de su directorio home. ¿Es el mismo que el de su compañero/a? **pwd home**
b) Cree un directorio para cada asignatura en la que se van a realizar prácticas de laboratorio y, dentro de cada directorio, nuevos directorios para cada una de las prácticas realizadas hasta el momento. **mkdir <directorio> (cree los convenientes)**
c) Dentro del directorio de la asignatura fundamentos del software (llamado FS) y dentro del directorio creado para esta práctica, copie los archivos hosts y passwd que se encuentran dentro del directorio /etc. **cp -a /etc/hosts FS cp -a /etc/passwd FS**
d) Muestre el contenido de cada uno de los archivos. **cat <archivo>**

Ejercicio 1.6. Situados en algún lugar de su directorio principal de usuario (directorio HOME), cree los directorios siguientes: Sesion.1, Sesion.10, Sesion.2, Sesion.3, Sesion.4, Sesion.27, Prueba.1 y Sintaxis.2 y realice las siguientes tareas: **mkdir Sesion.1 Sesion.10 Sesion.2 Sesion.3 Sesion.4 Sesion.27 Prueba.1 Sintaxis.2**

a) Borre el directorio Sesion.4 **rmdir Sesion.4**
b) Liste todos aquellos directorios que empiecen por Sesion. y a continuación tenga un único carácter **ls Sesion.?**
c) Liste aquellos directorios cuyos nombres terminen en .1 **ls *.1**
d) Liste aquellos directorios cuyos nombres terminen en .1 o .2 **ls *.[12]**
e) Liste aquellos directorios cuyos nombres contengan los caracteres si **ls *[si]***
f) Liste aquellos directorios cuyos nombres contengan los caracteres si y terminen en .2 **ls *[si]*.2**

Ejercicio 1.7. Desplacémonos hasta el directorio /bin, genere los siguientes listados de archivos (siempre de la forma más compacta y utilizando los metacaracteres apropiados): **cd /bin**

a) Todos los archivos que contengan sólo cuatro caracteres en su nombre. **ls ????**
b) Todos los archivos que comiencen por los caracteres d, f. **ls [df]***
c) Todos los archivos que comiencen por las parejas de caracteres sa, se, ad. **ls sa*se*ad***
d) Todos los archivos que comiencen por t y acaben en r. **ls t*r**

Ejercicio 1.8. Liste todos los archivos que comiencen por tem y terminen por .gz o .zip : **ls tem*.gz ls tem*.zip**

a) De tu directorio HOME.
b) Del directorio actual.
c) ¿Hay alguna diferencia en el resultado de su ejecución? Razone la respuesta.

Ejercicio 1.9. Muestre del contenido de un archivo regular que contenga texto:

a) Las 10 primeras líneas. **head <archivo>**
b) Las 5 últimas líneas. **tail -5 <archivo>**

Ejercicio 1.10. Cree un archivo empleando para ello cualquier editor de textos y escriba en el mismo varias palabras en diferentes líneas. A continuación trate de mostrar su contenido de

manera ordenada empleando diversos criterios de ordenación. **touch <archivo> gedit <archivo>**

sort <archivo> (alfabeticamente)

sort -r <archivo> (inverso)

sort -R

Ejercicio 1.11. ¿Cómo podría ver el contenido de todos los archivos del directorio actual que terminen en .txt o .c? cat *.txt*.c

Practica 2.

Ejercicio 2.1. Se debe utilizar solamente una vez la orden chmod en cada apartado. Los cambios se harán en un archivo concreto del directorio de trabajo (salvo que se indique otra cosa). Cambiaremos uno o varios permisos en cada apartado (independientemente de que el archivo ya tenga o no dichos permisos) y comprobaremos que funciona correctamente:

Dar permiso de ejecución al "resto de usuarios". chmod o+w Practicas

Dar permiso de escritura y ejecución al "grupo". chmod g+wx Practicas

Quitar el permiso de lectura al "grupo" y al "resto de usuarios". chmod go-r Practicas

Dar permiso de ejecución al "propietario" y permiso de escritura el "resto de usuarios". chmod u+x, o+w Practicas

Dar permiso de ejecución al "grupo" de todos los archivos cuyo nombre comience con la letra "e". Nota: Si no hay más de dos archivos que cumplan esa condición, se deberán crear archivos que empiecen con "e" y/o modificar el nombre de archivos ya existentes para que cumplan esa condición. chmod g+x e*

Ejercicio 2.2. Utilizando solamente las órdenes de la práctica anterior y los metacaracteres de redirección de salida:

- **Cree un archivo llamado ej31 , que contendrá el nombre de los archivos del directorio padre del directorio de trabajo. ls .. > ej31**
- **Cree un archivo llamado ej32 , que contendrá las dos últimas líneas del archivo creado en el ejercicio anterior. tail -2 ej31 > ej32**
- **Añada al final del archivo ej32 , el contenido del archivo ej31 . cat ej31 >> ej32**

Ejercicio 2.3. Utilizando el metacarácter de creación de cauces y sin utilizar la orden cd:

- Muestre por pantalla el listado (en formato largo) de los últimos 6 archivos del directorio /etc.
ls -l /etc | tail -6
- La orden wc muestra por pantalla el número de líneas, palabras y bytes de un archivo (consulta la orden man para conocer más sobre ella). Utilizando dicha orden, muestre por pantalla el número de caracteres (sólo ese número) de los archivos del directorio de

trabajo que comiencen por los caracteres “e” o “f”. **wc e*f***

Ejercicio 2.4. Resuelva cada uno de los siguientes apartados.

a) Cree un archivo llamado ejercicio1, que contenga las 17 últimas líneas del texto que proporciona la orden man para la orden chmod (se debe hacer en una única línea de órdenes y sin utilizar el metacarácter “;”). **touch ejercicio 1 man chmod | tail -17 > ejercicio1**

b) Al final del archivo ejercicio1, añada la ruta completa del directorio de trabajo actual.
pwd >> ejercicio1

c) Usando la combinación de órdenes mediante paréntesis, cree un archivo llamado ejercicio3 que contendrá el listado de usuarios conectados al sistema (orden who) y la lista de archivos del directorio actual. **touch ejercicio3 (who ; ls) > ejercicio3 o (who || ls) >ejercicio3**

d) Añada, al final del archivo ejercicio3, el número de líneas, palabras y caracteres del archivo ejercicio1. Asegúrese de que, por ejemplo, si no existiera ejercicio1, los mensajes de error también se añadieran al final de ejercicio3. **wc -mlw ejercicio1 >> ejercicio 3 o wc ejercicio1 >> ejercicio3**

e) Con una sola orden chmod, cambie los permisos de los archivos ejercicio1 y ejercicio3, de forma que se quite el permiso de lectura al “grupo” y se dé permiso de ejecución a las tres categorías de usuarios. **chmod g-r, ouga+x ejercicio1 ejercicio3**

Practica 3.

Ejercicio 3.5. Construya un guion que acepte como argumento una cadena de texto (por ejemplo, su nombre) y que visualice en pantalla la frase Hola y el nombre dado como argumento.

#!/bin/bash

echo “Hola $1”

./ejercicio3.5 <Nombre> bash ejercicio3.5 <nombre>

Ejercicio 3.6. Varíe el guion anterior para que admita una lista de nombres.

#!/bin/bash

echo “Hola $*”

./ejercicio3.5 <Nombre> bash ejercicio3.5 <nombre>

Ejercicio 3.7. Cree tres variables llamadas VAR1, VAR2 y VAR3 con los siguientes valores respectivamente "hola", "adios" y "14".

VAR1=HOLA

VAR2=ADIOS

VAR3=14

a) Imprima los valores de las tres variables en tres columnas con 15 caracteres de ancho.
printf "%-15s %-15s %-15d" $VAR1 $VAR2 $VAR3
b) ¿Son variables locales o globales? **locales**
c) Borre la variable VAR2. **unset VAR2**
d) Abra otra ventana de tipo terminal, ¿puede visualizar las dos variables restantes? **no**
e) Cree una variable de tipo vector con los valores iniciales de las tres variables. **VAR=(hola adios 14)**
f) Muestre el segundo elemento del vector creado en el apartado e. **echo ${VAR[1]}**

Ejercicio 3.8. Cree un alias que se llame ne (nombrado así para indicar el número de elementos) y que devuelva el número de archivos que existen en el directorio actual. ¿Qué cambiaría si queremos que haga lo mismo pero en el directorio home correspondiente al usuario que lo ejecuta? alias ne= ls -A | wc -l

Ejercicio 3.9. Indique la línea de orden necesaria para buscar todos los archivos a partir del directorio home de usuario ($HOME) que tengan un tamaño menor de un bloque. ¿Cómo la modificaría para que además imprima el resultado en un archivo que se cree dentro del directorio donde nos encontremos y que se llame archivosP? find $HOME -size-1 find $HOME -size-1 > ArchivosP

Ejercicio 3.10. Indique cómo buscaría todos aquellos archivos del directorio actual que contengan la palabra "ejemplo". find ./*ejemplo*

Ejercicio 3.11. Complete la información de find y grep utilizando para ello la orden man.
Ejercicio 3.12. Indique cómo buscaría si un usuario dispone de una cuenta en el sistema. cat /etc/passwd | grep <nombre usuario>

Ejercicio 3.13. Indique cómo contabilizar el número de ficheros de la propia cuenta de usuario que no tengan permiso de lectura para el resto de usuarios. find ~ ! -perm o+r | wc -l

Ejercicio 3.14. Modifique el ejercicio 8 de forma que, en vez de un alias, sea un guion llamado numE el que devuelva el número de archivos que existen en el directorio que se le pase como argumento.

#!/bin/bash

printf "Numero de archivos que existen en el directorio: \n"

ls $1 | wc -l

echo

Practica 4.

Ejercicio 4.1: Utilizando una variable que contenga el valor entero 365 y otra que guarde el número del día actual del año en curso, realice la misma operación del ejemplo anterior usando cada una de las diversas formas de cálculo comentadas hasta el momento, es decir, utilizando expr, $((...)) y $[...].

```
anio=365
dia=`date + %j`
echo "Faltan `expr ($anio - $dia)/7` semanas"

echo "Faltan $[ ($anio-$dia) / 7 ] semanas hasta fin de año"

echo “Faltan $(( ($anio-$dia) / 7)) semanas hasta el fin de año”
```

Ejercicio 4.3: Utilizando el operador de división, ponga un caso concreto donde se aprecie que la asignación abreviada es equivalente a la asignación completa, es decir, que x/=y equivale a x=x/y.

```
x=6 y=3      echo $(( x=x/y))     echo $x     echo $(x/=y)   echo $x
```

Ejercicio 4.4: Compruebe qué ocurre si en el ejemplo anterior utiliza comillas dobles o simples para acotar todo lo que sigue a la orden echo. ¿Qué sucede si se acota entre comillas dobles solamente la expresión aritmética que se quiere calcular?, ¿y si se usan comillas simples?

```
echo 6/5|bc -l

echo "6/5|bc -l" -> te muestra todo como una cadena de texto

 echo '6/5|bc -l' -> lo mismo

echo "6/5"|bc -l -> muestra el resultado de la operación que está "
```

Ejercicio 4.5: Calcule con decimales el resultado de la expresión aritmética (3-2)/5. Escriba todas las expresiones que haya probado hasta dar con una respuesta válida. Utilizando una solución válida, compruebe qué sucede cuando la expresión aritmética se acota entre comillas dobles; ¿qué ocurre si se usan comillas simples?, ¿y si se ponen apóstrofos inversos?

```
echo ‘(3-2)/5’ | bc -l     echo “(3-2)/5” | bc -l
```

Ejercicio 4.7: Con la orden let es posible realizar asignaciones múltiples y utilizar operadores que nosotros no hemos mencionado anteriormente. Ponga un ejemplo de asignación múltiple

y, por otra parte, copie en un archivo el orden en el que se evalúan los operadores que admite. Apóyese a través de la ayuda que ofrece help let.

let x=y='3+2' echo xy

Ejercicio 4.8: Probad los siguientes ejemplos y escribir los resultados obtenidos con la evaluación de expresiones

a)no y 6

b)no y 5

c)ls, si y 5

d)no y 6

e)no y 5

f)ls, si y 5

g)0

h) 1

i)3 no es mayor que 5

Ejercicio 4.9: Haciendo uso de las órdenes conocidas hasta el momento, construya un guion que admita dos parámetros, que compare por separado si el primer parámetro que se le pasa es igual al segundo, o es menor, o es mayor, y que informe tanto del valor de cada uno de los parámetros como del resultado de cada una de las evaluaciones mostrando un 0 o un 1 según corresponda.

```
#!/bin/bash

declare -i A=$1
declare -i B=$1

esIgual=$[$A==$B]
esMenor=$[$A<$B]
esMayor=$[$A>$B]

printf "Los valores obtenidos son A=%s, B=%s.\n" $A $B
printf "Los valores son iguales %s\n" $esIgual
printf "El primer valor es menor que el segundo %s\n" $esMenor
printf "El primer valor es mayor que el segundo %s\n" $esMayor
```

Ejercicio 4.10: Usando test, construya un guion que admita como parámetro un nombre de archivo y realice las siguientes acciones: asignar a una variable el resultado de comprobar si el archivo dado como parámetro es plano y tiene permiso de ejecución sobre él; asignar a otra variable el resultado de comprobar si el archivo es un enlace simbólico; mostrar el valor de las

dos variables anteriores con un mensaje que aclare su significado. Pruebe el guion ejecutándolo con /bin/cat y también con /bin/rnano.

```
#!/bin/bash

declare -r v1=$1

esPlano=$`test -f '$1' -a -x ' $1' &&

        echo "El fichero '$1' existe, es plano y ejecutable" || echo "El fichero '$1' no existe, no es plano o no es ejecutable"

esSimbolico=$`test -h '$1' &&

        echo "El fichero '$1' es un enlace simbolico" || echo "El fichero '$1' no es un enlace simbolico"

echo

echo $esPlano

echo

echo $esSimbolico
```

Ejercicio 4.11: Ejecute help test y anote qué otros operadores se pueden utilizar con la orden test y para qué sirven. Ponga un ejemplo de uso de la orden test comparando dos expresiones aritméticas y otro comparando dos cadenas de caracteres.

```
help test
```

Ejercicio 4.12: Responda a los siguientes apartados:

a) Razone qué hace la siguiente orden:

```
if test -f ./sesion5.pdf ; then printf "El archivo ./sesion5.pdf existe\n"; fi
```

Imprimiría en pantalla si existiese el archivo sesion5 en el directorio actual el mensaje, pero al no existir,no imprime ningún resultado en pantalla.

b) Añada los cambios necesarios en la orden anterior para que también muestre un mensaje de aviso en caso de no existir el archivo. (Recuerde que, para escribir de forma legible una orden que ocupe más de una línea, puede utilizar el carácter "\" como final de cada línea que no sea la última.)

```
if test -f ./sesion5.pdf ; then printf "El archivo ./sesion5.pdf existe\n"; else printf "No existe"; fi
```

c) Sobre la solución anterior, añada un bloque elif para que, cuando no exista el archivo ./sesion5.pdf, compruebe si el archivo /bin es un directorio. Ponga los mensajes adecuados para conocer el resultado en cada caso posible.

```
if test -f ./sesion5.pdf ; then
```

```
    printf “El archivo./sesion5.pdf existe\n”;
elif test -d ./bin; then
    printf “El archivo ./bin es un directorio”;
else echo “No se cumplen los requisitos”
fi
```

d) Usando como base la solución del apartado anterior, construya un guion que sea capaz de hacer lo mismo pero admitiendo como parámetros la ruta relativa del primer archivo a buscar y la ruta absoluta del segundo. Pruébelo con los dos archivos del apartado anterior.

```
#!/bin/bash
if test -f $1 &&  test -d $2
then
    printf “El archivo $1 existe y $2 es un directorio\n”
elif test -f $1 && ! test -d $2
then
    printf “El archivo $1 existe y $2 no es un directorio\n”
elif ! test -f $1 && test -d $2
then
    printf “El archivo $1 no existe y $2 es un directorio\n”
else
    printf “El archivo $1 no existe y $2 no es un directorio\n”
fi
```

Ejercicio 4.13: Construya un guion que admita como argumento el nombre de un archivo o directorio y que permita saber si somos el propietario del archivo y si tenemos permiso de lectura sobre él.

```
#!/bin/bash
if test -r $1 && test -o $1
then
    printf  “Tenemos permiso de lectura y somos propietarios del archivo”
elif test -r $1 && ! test -o $1
```

```
    printf  “Tenemos permiso de lectura pero no somos propietarios”
else
    printf  “No tenemos permiso de lectura ni somos propietarios”
fi
```

Ejercicio 4.14: Escriba un guion que calcule si el número de días que faltan hasta fin de año es múltiplo de cinco o no, y que comunique el resultado de la evaluación. Modifique el guion anterior para que admita la opción -h de manera que, al ejecutarlo con esa opción, muestre información de para qué sirve el guion y cómo debe ejecutarse.

El siguiente guion de ejemplo se puede utilizar para borrar el archivo temporal que se le dé como argumento. Si rm devuelve 0, se muestra el mensaje de confirmación del borrado; en caso contrario, se muestra el código de error. Como se puede apreciar, hemos utilizado la variable $LINENO que indica la línea actualmente en ejecución dentro del guion.

```
#!/bin/bash
anio=365
dia=`date +%j`
dias_restantes=$[$anio - $date]
echo “Faltan $dias_restantes para acabar el anio\n”
esMultipo=$[$dias_restantes%5]
if [ esMultipo == 0 ]; then
     echo “Es multipo de 5”;
    else echo “NO es multipo de 5”;
fi
```

Ejercicio 4.16: Haciendo uso del mecanismo de cauces y de la orden echo, construya un guion que admita un argumento y que informe si el argumento dado es una única letra, en mayúsculas o en minúsculas, o es algo distinto de una única letra.

```
#!/bin/bash
if [[ $1 =~ [a-z] ]]
then
    echo “La letra es minuscula”
elif [[ $1 =~ [A-Z] ]]
```

```
then
    echo “La letra es mayuscula”
else
    echo “No es una letra”
fi
```

Ejercicio 4.17: Haciendo uso de expresiones regulares, escriba una orden que permita buscar en el árbol de directorios los nombres de los archivos que contengan al menos un dígito y la letra e. ¿Cómo sería la orden si quisiéramos obtener los nombres de los archivos que tengan la letra e y no contengan ni el 0 ni el 1?

```
ls *?*e*
```

Practica 5.

Ejercicio 5.1. Escriba un guion que acepte dos argumentos. El primero será el nombre de un directorio y el segundo será un valor entero. El funcionamiento del guion será el siguiente: deberán anotarse en un archivo denominado archivosSizN.txt aquellos archivos del directorio dado como argumento y que cumplan la condición de tener un tamaño menor al valor aportado en el segundo argumento. Se deben tener en cuenta las comprobaciones sobre los argumentos, es decir, debe haber dos argumentos, el primero deberá ser un directorio existente y el segundo un valor entero.

```
#!/bin/bash
if (( $# !=2 )); then
    echo “Introduce dos argumentos”
    exit 1
fi
if test -d $1 ; then
    echo “El archivo es un directorio”
else
    echo “No es un directorio”
fi
let esnum=$2
if (( $?==0 )) ; then
```

```
    echo “Es un valor numerico”
     exit
fi
for (( CONTADOR=0; CONTADOR<$2; CONTADOR++ )) ; do
find $1 -size $CONTADOR >> ./archivosSizN.txt
printf  “Archivos copiados”
done
```

Ejercicio 5.2. Escriba un guion que acepte el nombre de un directorio como argumento y muestre como resultado el nombre de todos y cada uno de los archivos del mismo y una leyenda que diga "Directorio", "Enlace" o "Archivo regular", según corresponda. Incluya la comprobación necesaria sobre el argumento, es decir, determine si el nombre aportado se trata de un directorio existente.

```
#Comprobamos que existe el argumento
if [ $# !=1 ]; then
   echo “Uso directorio”
   exit 1
fi
#Comprobamos si existe y tiene permiso de lectura
if ! test -d $1 && ! test -r $1; then
   echo “El directorio no existe o no tiene permiso de lectura”
   exit 1
fi
#Listamos los archivos
for ARCH in $1/* ;
  do
    if test -d $ARCH ; then
        printf “Directorio %35s\n” $ARCH
   elif test -f $ARCH; then
        printf “Archivo regular %35s\n” $ARCH
```

```
   elif test -h $ARCH; then
        printf “Enlace simbolico %35s\n” $ARCH
else
       printf “Archivo %35s\n” $ARCH
fi
done
```

Ejercicio 5.3. Escriba un guion en el que, a partir de la pulsación de una tecla, detecte la zona del teclado donde se encuentre. Las zonas vendrán determinadas por las filas. La fila de los números 1, 2, 3, 4, ... será la fila 1, las teclas donde se encuentra la Q, W, E, R, T, Y,... serán de la fila 2, las teclas de la A, S, D, F,...serán de la fila 3 y las teclas de laZ,X,C,V,...serán de la fila 4. La captura de la tecla se realizará mediante la orden read.

```
#!/bin/bash
read -p “Introduzca una letra: “ tecla
fila=-1
case “$letra” in
        1|2|3|4|5|6|7|8|9|0) fila=1;;
        q|w|e|r|t|y|u|i|o|p) fila=2;;
         a|s|d|f|g|h|j|k|l) fila=3;;
         z|x|c|v|b|n|m) fila=4;;
        *) printf “Error de tecla”;;
esac
if [  $fila != ”-1”  ]; then
printf “La letra ‘$tecla’ seleccionada pertenece a la fila “ $fila
fi
```

Ejercicio 5.4. Escriba un guion que acepte como argumento un parámetro en el que el usuario indica el mes que quiere ver, ya sea en formato numérico o usando las tres primeras letras del nombre del mes, y muestre el nombre completo del mes introducido. Si el número no está comprendido entre 1 y 12 o las letras no son significativas del nombre de un mes, el guion deberá mostrar el correspondiente mensaje de error.

```
#!/bin/bash
read -p “Introduce las tres primeras letras o el numero del mes “ mes
```

```
case “$mes” in
        1|'ene') echo "enero" ;;
         2|'feb') echo "febrero" ;;
         3|'mar') echo "marzo" ;;
         4|'abr') echo "abril" ;;
         5|'may') echo "mayo" ;;
         6|'jun') echo "junio" ;;
         7|'jul') echo "julio" ;;
         8|'ago') echo "agosto" ;;
          9|'sep') echo "septiembre" ;;
         10|'oct') echo "octubre" ;;
         11|'nov') echo "noviembre" ;;
          12|'dic') echo "diciembre" ;;
         *) echo "Error" ;;
         esac
```

Ejercicio 5.5. Escriba un guion que solicite un número hasta que su valor esté comprendido entre 1 y 10. Deberá usar la orden while y, para la captura del número, la orden read.

```
#!/bin/bash
valor=1
while [ $valor == 1 ];
do
echo
      read -p “Introduce un numero entre el 1 y el 10 “  num
     if [ $num -gt 1 ] && [ $num -lt 10 ]; then
     valor=0;
fi
```

```
done

echo

echo “Valor introducido correcto. Valor introducido: “ $num
```

Ejercicio 5.7. Escriba un guion que admita como argumento el nombre de un tipo de shell (por ejemplo, csh, sh, bash, tcsh, etc.) y nos dé un listado ordenado alfabéticamente de los usuarios que tienen dicho tipo de shell por defecto cuando abren un terminal. Dicha información del tipo de shell asignado a un usuario se puede encontrar en el archivo /etc/passwd, cuyo contenido está delimitado por ':'. Cada información situada entre esos delimitadores representa un campo y precisamente el campo que nos interesa se encuentra situado en primer lugar. En definitiva, para quedarnos con lo que aparece justo antes del primer delimitador será útil la orden siguiente:cut -d':' -f1 /etc/passwd

Donde la opción –d indica cuál es el delimitador utilizado y la opción –f1 representa a la secuencia de caracteres del primer campo. Realice, utilizando el mecanismo de cauces, el ejercicio pero usando la orden cat para mostrar el contenido de un archivo y encauzado con la orden cut para filtrar la información que aparece justo antes del delimitador ':'[3].

Realice también la comprobación de la validez del tipo de Shell que se introduce como argumento. Use para ello la información que encontrará en el archivo /etc/shells donde encontrará los tipos de Shell que se pueden utilizar en el sistema.

```
#!/bin/bash

#Comprobamos que aparece un argumento

clear

if [ $# != 1 ];  then

        echo “Uso: $0 <shells>”

         exit 1

fi

#Comprobamos que el argumento se corresponde con uno de los shells del sistema

cat /etc/shells | grep $1 > /dev/null

if [ $? == 1 ]; then

     echo “Debe introducir las siguientes shells: “

       cat /etc/shells

       echo

else
```

```
        echo “Usuarios con la shell $1: “

        echo

        cat /etc/passwd | grep $1 | cut -d”:“ -f1 | sort

       echo

fi
```

Ejercicio 5.9. Hacer un script en Bash denominado newdirfiles con los siguientes tres argumentos:

<dirname> Nombre del directorio que, en caso de no existir, se debe crear para alojar en él los archivos
que se han de crear.
<num_files> Número de archivos que se han de crear.
<basefilename> Será una cadena de caracteres que represente el nombre base de los archivos.
Ese guion debe realizar lo siguiente:

Comprobar que el número de argumentos es el correcto y que el segundo argumento tenga un valor comprendido entre 1 y 99.

Crear, en caso de no existir, el directorio dado en el primer argumento a partir del directorio donde se esté situado y que posea permisos de lectura y escritura para el usuario $USER.

Dentro del directorio dado en el primer argumento, crear archivos cuyos contenidos estarán vacíos y cuyos nombres lo formarán el nombre dado como tercer argumento y un número que irá desde 01 hasta el número dado en el segundo argumento.

```
#!/bin/bash

clear

#Comprobamos el numero de parametros

if [ $# != 3 ]; then

    echo “Use: $0 <dirname> <num_files> <basefilename>”

     exit 1

fi

if [ $2 -lt 1 ] || [ $2 -gt 99 ]; then

    echo “El valor del segundo argumento debe estar entre 1 y 99”

    exit 1

fi
```

```
if ! test -d $1 ; then
    echo “El directorio no existe”
     mkdir $1
else
     echo “El directorio ya existe”
fi
#Se procede a crear el numero de archivos solicitados atendiendo a la numeracion, desde 01 hasta el numero dado como argumento
for (( i=1; i<=$2; i++ )) ;
do
if [ $i -lt 10 ]; then
      touch $1/$3 ‘0’ $i
else
     touch $1/$3$i
fi
done
echo “Archivos creados correctamente”
```

Practica 6.

Ejercicio 6.1. Indique cuál ha sido el error introducido en el guion anterior y cómo se corregiría.

Falta un espacio `if [ "$2" = "after" ] ; then`

Ejercicio 6.2. Aplicar las herramientas de depuración vistas en la sección 2 para la detección de errores durante el desarrollo de los guiones propuestos como ejercicios en la práctica 5.

Ejercicio 6.3. Escribir un guion que nos dé el nombre del proceso del sistema que consume más memoria.

```
#!/bin/bash
echo “El proceso del sistema que mas memoria consume es: “
ps -eo cmd,pmem -- sort -pmem | head -2
```

Ejercicio 6.4. Escribir un guion que escriba números desde el 1 en adelante en intervalos de un segundo ¿Cómo se podría, desde otro terminal, detener la ejecución de dicho proceso, reanudarlo y terminar definitivamente su ejecución?

```
#!/bin/bash

numero=1

while (( $numero > 0 ))

do

    printf "%d\n" $numero

    numero=`expr $numero + 1`

    sleep 2s

done
```

Ejercicio 6.5. ¿Se puede matar un proceso que se encuentra suspendido? En su caso, ¿cómo?

```
disown -a %(numero)
```

Ejercicio 6.6. ¿Qué debemos hacer a la orden top para que nos muestre sólo los procesos nuestros?

```
top -u <nombre usuario>
```

Practica 7.

Ejercicio 7.1. Pruebe a comentar en el archivo fuente main.cpp la directiva de procesamiento "#include "functions.h". La línea quedaría así: //#include "functions.h". Pruebe a generar ahora el módulo objeto con la orden de compilación mostrada anteriormente. ¿Qué ha ocurrido?

Aparece un archivo .o

Ejercicio 7.2. Explique por qué el enlazador no ha podido generar el programa archivo ejecutable programa2 del ejemplo anterior y, sin embargo, ¿por qué sí hemos podido generar el módulo main2.o?

El archivo .o si se general porque no comprueba las referencias externas. Se produce el error al enlacer por no haber indicado la libreria que queremos usar

Ejercicio 7.3. Explique por qué la orden g++ previa ha fallado. Explique los tipos de errores que ha encontrado.

No encuentra la libretia functions.h

Ejercicio 7.4. Copie el contenido del makefile previo a un archivo llamado makefileE ubicado en el mismo directorio en el que están los archivos de código fuente .cpp. Pruebe a modificar distintos archivos .cpp (puede hacerlo usando la orden touch sobre uno o varios de esos archivos) y compruebe la secuencia de instrucciones que se muestra en el terminal al ejecutarse la orden make. ¿Se genera siempre la misma secuencia de órdenes cuando los archivos han sido modificados que cuando no? ¿A qué cree puede deberse tal comportamiento?

Depende de si los archivos modificados son dependencias donde si se actualiza el makefileE compilado. En este caso, se actualizaría sí o sí ya que el programa2 es dependiente de todos los otros cpp.

Ejercicio 7.5. Obtener un nuevo makefileF a partir del makefile del ejercicio anterior que incluya además las dependencias sobre los archivos de cabecera. Pruebe a modificar cualquier archivo de cabecera (usando la orden touch) y compruebe la secuencia de instrucciones que se muestra en el terminal al ejecutarse la orden make.

Habria que agregarle el ./include/function.h y el -I./includes

Ejercicio 7.6. Usando como base el archivo makefileG, sustituya la línea de orden de la regla cuyo objetivo es programa2 por otra en la que se use alguna de las variables especiales y cuya ejecución sea equivalente.

```
# Nombre archivo: makefileG

# Uso: make –f makefileG

# Descripción: Mantiene todas las dependencias entre los módulos y la biblioteca que utiliza el programa2.

# Variable que indica el compilador que se va a utilizar

CC=g++

# Variable que indica el directorio en donde se encuentran los archivos de cabecera

INCLUDE_DIR= ./includes

# Variable que indica el directorio en donde se encuentran las bibliotecas

LIB_DIR= ./

#Variable que indica las opciones que se le va a pasar al compilador

CPPFLAGS= -Wall

programa2: main2.o factorial.o hello.o libmates.a

    $(CC) -L$(LIB_DIR) -o $@ main2.o factorial.o hello.o -lmates
```

```
main2.o: main2.cpp
	$(CC) -I$(INCLUDE_DIR) -c main2.cpp

factorial.o: factorial.cpp
	$(CC) -I$(INCLUDE_DIR) -c factorial.cpp

hello.o: hello.cpp
	$(CC) -I$(INCLUDE_DIR) -c hello.cpp

libmates.a: sin.o cos.o tan.o
	ar -rvs libmates.a sin.o cos.o tan.o

sin.o: sin.cpp
	$(CC) -I$(INCLUDE_DIR) -c sin.cpp

cos.o: cos.cpp
	$(CC) -I$(INCLUDE_DIR) -c cos.cpp

tan.o: tan.cpp
	$(CC) -I$(INCLUDE_DIR) -c tan.cpp
clean:
	rm *.o programa2
```

Ejercicio 7.7. Utilizando como base el archivo makefileG y los archivos fuente asociados, realice los cambios que considere oportunos para que, en la construcción de la biblioteca estática libmates.a, este archivo pase a estar en un subdirectorio denominado libs y se pueda enlazar correctamente con el resto de archivos objeto.

```
# Nombre archivo: makefileG
# Uso: make –f makefileG
```

```
# Descripción: Mantiene todas las dependencias entre los módulos y la biblioteca que utiliza el
programa2.

# Variable que indica el compilador que se va a utilizar
CC=g++

# Variable que indica el directorio en donde se encuentran los archivos de cabecera
INCLUDE_DIR= ./includes

# Variable que indica el directorio en donde se encuentran las bibliotecas
LIB_DIR= ./

#Variable que indica las opciones que se le va a pasar al compilador
CPPFLAGS= -Wall

programa2: main2.o factorial.o hello.o ./libs/libmates.a
    $(CC) -L$(LIB_DIR) -o $@ main2.o factorial.o hello.o -lmates
main2.o: main2.cpp
    $(CC) -I$(INCLUDE_DIR) -c main2.cpp
factorial.o: factorial.cpp
    $(CC) -I$(INCLUDE_DIR) -c factorial.cpp
hello.o: hello.cpp
    $(CC) -I$(INCLUDE_DIR) -c hello.cpp
libmates.a: sin.o cos.o tan.o
    ar -rvs libmates.a sin.o cos.o tan.o
sin.o: sin.cpp
    $(CC) -I$(INCLUDE_DIR) -c sin.cpp
cos.o: cos.cpp
    $(CC) -I$(INCLUDE_DIR) -c cos.cpp
tan.o: tan.cpp
    $(CC) -I$(INCLUDE_DIR) -c tan.cpp
clean:
    rm *.o programa2
```

Ejercicio 7.8. Busque la variable predefinida de make que almacena la utilidad del sistema que permite construir bibliotecas. Recuerde que la orden para construir una biblioteca estática a partir de una serie de archivos objeto es ar (puede usar la orden grep para filtrar el contenido; no vaya a leer línea a línea toda la salida). Usando el archivo makefileG, sustituya la orden ar por su variable correspondiente.

```
# Nombre archivo: makefileG
# Uso: make –f makefileG
# Descripción: Mantiene todas las dependencias entre los módulos y la
# biblioteca que utiliza el programa2.
# Variable que indica el compilador que se va a utilizar CC=g++
# Variable que indica el directorio en donde se encuentran los archivos de cabecera
INCLUDE_DIR= ./includes
# Variable que indica el directorio en donde se encuentran las bibliotecas LIB_DIR= ./
programa2: main2.o factorial.o hello.o libmates.a
$(CC) -L$(LIB_DIR) -o programa2 main2.o factorial.o hello.o -lmates
main2.o: main2.cpp
```

```
    $(CC) -I$(INCLUDE_DIR) -c main2.cpp
factorial.o: factorial.cpp
$(CC) -I$(INCLUDE_DIR) -c factorial.cpp
hello.o: hello.cpp
    $(CC) -I$(INCLUDE_DIR) -c hello.cpp
libmates.a: sin.o cos.o tan.o
    $(AR) -rvs libmates.a sin.o cos.o tan.o
sin.o: sin.cpp
    $(CC) -I$(INCLUDE_DIR) -c sin.cpp
cos.o: cos.cpp
    $(CC) -I$(INCLUDE_DIR) -c cos.cpp
tan.o: tan.cpp
    $(CC) -I$(INCLUDE_DIR) -c tan.cpp
```

Ejercicio 7.9. Dado el siguiente archivo makefile, explique las dependencias que existen y para qué sirve cada una de las líneas del mismo. Enumere las órdenes que se van a ejecutar a consecuencia de invocar la utilidad make sobre este archivo.

```
# Nombre archivo: makefileH
# Uso: make -f makefileH
# Descripción: Mantiene todas las dependencias entre los módulos que utiliza el
# programa1.
# La variable CC indica que vamos a compilar con c++
CC=g++
#La varialble CPPFLAGS nos indica nos indica todos los warning que se pueden dar
CPPFLAGS=-Wall –I./includes
#SOURCEMODUELS nos indica todos los archivos cpp
SOURCE_MODULES=main.cpp factorial.cpp hello.cpp
#OBJECTMODULES nos indica los archivos .o
OBJECT_MODULES=$(SOURCE_MODULES:.cpp=.o)
#EXECUTABLE nos indica el nombre del archivo ejecutable que se generara
EXECUTABLE=programa1
#all
all: $(OBJECT_MODULES) $(EXECUTABLE)
$(EXECUTABLE): $(OBJECT_MODULES)
     $(CC) $^ -o $@
# Regla para obtener los archivos objeto .o que dependerán de los archivos .cpp
# Aquí, $< y $@ tomarán valores respectivamente main.cpp y main.o y así sucesivamente
.o: .cpp
     $(CC) $(CPPFLAGS) $< -o $@
```

Ejercicio 7.10. Con la siguiente especificación de módulos escriba un archivo denominado makefilePolaca que automatice el proceso de compilación del programa final de acuerdo a la siguiente descripción:

Compilador: gcc o g++

Archivos cabecera: calc.h (ubicado en un subdirectorio denominado cabeceras) Archivos fuente: main.c stack.c getop.c getch.c

Nombre del programa ejecutable: calculadoraPolaca

Además, debe incluir una regla denominada borrar, sin dependencias, cuya funcionalidad sea la de eliminar los archivos objeto y el programa ejecutable.

```
CC=g++
INCLUDES=./cabeceras
SOURCES=main.cpp stack.cpp getop.cpp getch.cpp
OBJECT=main.o stack.o getop.o getch.o
EJECUTABLE=calculadoraPolaca

$(EJECUTABLE): $(OBJECT)
$(CC) -I$(INCLUDES) -O $@ $^

main.o: main.cpp $(INCLUDES)/calc.h
$(CC) -I$(INCLUDES) -c $<

stack.o: stack.cpp $(INCLUDES)/calc.h
$(CC) -I$(INCLUDES) -c $<

getop.o: getop.cpp $(INCLUDES)/calc.h
$(CC) -I$(INCLUDES) -c $<

getch.o: getch.cpp $(INCLUDES)/calc.h
$(CC) -I$(INCLUDES) -c $<

borrar:
rm $(OBJECT) $(EJECUTABLE)
```

Practica 8.

Ejercicio 8.1. Compile los archivos main.cpp factorial.cpp hello.cpp y genere un ejecutable con el nombre ejemplo9.1. Lance gdb con dicho ejemplo y ejecútelo dentro del depurador. Describa la información que ofrece.

g++ -g main.cpp factorial.cpp hello.cpp -o ejemplo9.1 gdb ejemplo9.1

Ejercicio 8.2. Usando la orden list muestre el código del programa principal y el de la función factorial utilizados en el ejercicio 9.1 (para ello utilice la orden help list).

list main list factorial

Ejercicio 8.3. Ponga un punto de ruptura asociado a cada línea del programa fuente mainsesion09a.cpp donde aparezca el comentario /* break */. Muestre información de todas las variables que se estén usando cada vez que en la depuración se detenga la ejecución. Muestre la información del contador de programa mediante $pc y el de la pila con $sp.

g++ -g mainsesion09a.cpp -o mainsesion09 gdb mainsesion09
break 15 break 29 break 42 break 47 (todo esto dentro del gdb)

Ejercicio 8.4. Indique las órdenes necesarias para ver el valor de las variables final1 y final2 del programa generado en el ejercicio anterior en los puntos de ruptura correspondientes tras un par de iteraciones en el bucle for. Indique la orden para obtener el código ensamblador de la zona depurada.

(gdb) break 47 (gdb) run (gdb) continue (gdb) continue (gdb) info locals (gdb) dis
(gdb) disassemble

Ejercicio 8.5. Considerando la depuración de los ejercicios anteriores, elimine todos los puntos de ruptura salvo el primero.

(gdb) info breakpoints (gdb) delete breakpoints 2 3 4 (gdb) info breakpoints

Ejercicio 8.6. Realice las acciones del ejercicio 8.3 y las del ejercicio 8.5 en un guion y ejecútelas de nuevo mediante la opción -x de gdb. ¿Sabría decir qué hace este programa con la variable final2?

```
/* guion.gdb */

break 15
run
info locals
p/x $pc
x/i $pc
p/x $ps
x/i $ps
break 29
n
info locals
p/x $pc
x/i $pc
p/x $ps
x/i $ps
break 42
n
info locals
p/x $pc
x/i $pc
p/x $ps
x/i $ps
break 47
```

delete breakpoint 2 3 4

Ejercicio 8.7. Realice la depuración del programa ejecutable obtenido a partir del archivo fuente ejsesion09a.cpp. Utilizando gdb, trate de averiguar qué sucede y por qué no funciona. Intente arreglar el programa.

(gdb) list (gdb) break 27 (gdb) run (gdb) info locals (gdb) continue (gdb) info locals

(gdb) continue (gdb) info locals

Como podemos observar, la variable tmp permanece constante pues no se le asigna el valor devuelto por la función suma().

```
#include <stdlib.h>
#include <stdio.h>
/*
 Este programa trata de sumar una lista de números.
 La lista de números aparece en la variable "vector" y el resultado
 se almacena en la variable "final". */
/* Suma dos números entre sí */
int suma (int x, int y)
{
   int tmp;
   tmp = x + y;
   return tmp;
}
/* Realiza la sumatoria de un vector */
int sumatoria (float vector[], int n)
{
   int i;
   int tmp;
```

```
  tmp = 0;
  for (i = 0; i < n; i ++)
    tmp = suma(tmp, vector[i]); // Almacenamos en tmp
  printf ("Suma = %d\n", tmp);
  return tmp;
}
int main (void)
{
  float final;
  float vector[] = {0, 1, 2.3, 3.7, 4.10, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4};
  final = sumatoria(vector, 15); // El parámetro debe ser el tamaño del vector
  return 0;
}
```

Ejercicio 9.8. Compile el programa mainsesion09b.cpp y genere un ejecutable con el nombre ejemplo9.8. Ejecute gdb con dicho ejemplo y realice una ejecución depurada mediante la orden run. Añada un punto de ruptura (breakpoint) en la línea donde se invoca a la función cuenta (se puede realizar tal y como se muestra en el ejemplo anterior o mediante el número de línea donde aparezca la llamada a esa función). Realice 10 pasos de ejecución con step y otros 10 con next. Comente las diferencias.

(gdb) break 47 (gdb) run
(gdb) step (x10)
(gdb) n (x10)
Como se puede observar, la orden `next` ejecuta la siguiente linea del programa sin entrar en cada una de las líneas de una función mientras que `step` ejecuta una a una las instrucciones de la función.

Ejercicio 8.9. Depure el programa generado en el ejercicio anterior. Introduzca un punto de ruptura (breakpoint) dentro de la función cuenta. Usando la orden info frame, muestre la información del marco actual y del marco superior; vuelva al marco inicial y compruebe si ha cambiado algo

(gdb) break cuenta (gdb) run (gdb) info frame (gdb) up (gdb) info frame (gdb) down (gdb) info frame
No, no ha cambiado nada de información.

Ejercicio 8.10. Ponga un punto de ruptura en la línea 30 del programa utilizado en el ejercicio anterior (función multiplica) de tal forma que el programa se detenga cuando la variable final tenga como valor 8. Compruebe si se detiene o no y explique por qué.

(gdb) break if final == 8 No se detiene debido a que la función multiplica es llamada con argumentos `x=3` e `y=2`, y este es el que fija el número de repeticiones del bucle, y por tanto el valor máximo que alcanza`final` es 6.

Ejercicio 8.11. Pruebe el ejemplo anterior, ejecute después un continue y muestre el valor de la variable tmp. Todo haría indicar que el valor debiera ser 12 y sin embargo no es así, explique por qué.

(gdb) break 10 (gdb) run (gdb) print tmp (gdb) set variable tmp=10 (gdb) print tmp
(gdb) continue (gdb) print tmp

`tmp` es una variable local de la función cuenta. Por tanto, cada vez que se llama a la función se reinicia a `y+2` donde `y` es un parámetro de la función. Por tanto, aunque cambiemos el valor de la variable a 10, una nueva llamada a la función lo fija a `y+2`.

Ejercicio 8.12. Busque cualquier programa escrito en C++ que cumpla los requisitos para poderlo depurar utilizando la orden attach. Compílelo usando el flag de depuración, ejecútelo en una Shell en segundo plano y, en otra Shell, ejecute el depurador con el programa que se está ejecutando en estos momentos en la shell anterior. Utilice las órdenes de gdb para hacer que el programa que se está ejecutando se detenga en algún lugar y posteriormente se pueda continuar su ejecución. Escriba todos los pasos que haya realizado.

```console
$ g++ -g ejsesion10.cpp -o ej1
$ ./ej1 &
$ gdb
(gdb) attach 3619
$ kill
```

Ejercicio 8.13. Utilizando las órdenes de depuración de gdb, corrija el error del programa ecuacionSegundoGrado.cpp. Escriba todos los pasos que haya realizado. Pruebe a depurarlo usando attach.

```console
$ g++ -g ecuacionSegundoGrado.cpp -o ecuacionSegundoGrado
$ gdb ecuacionSegundoGrado
(gdb) run

Starting program:
/home/ricardo/Dropbox/dgiim/FS/Prácticas/practicas-fs/ArchivosModuloII/sesion09/ecuacionSegundoGrado

Vamos a resolver una ecuacion del tipo ax2+bx+c=0

Introduce el valor de a: 10
```

Introduce el valor de b: 1

Introduce el valor de c: 10

La primera solucion es: -nan
La segunda solucion es: -nan
[Inferior 1 (process 14680) exited normally]
$

Lo cual nos muestra que calcula ecuaciones de segundo
grado que no tienen soluciones reales.

Por tanto, modificamos la línea

```c++
if (ecuacionator(a, b, c) == 0) // Ya que si el discriminante es negativo, devuelve 0
```

```console
$ ./ecuacionSegundoGrado
Vamos a resolver una ecuacion del tipo ax2+bx+c=0

Introduce el valor de a: 32

Introduce el valor de b: 24

Introduce el valor de c: 11

La ecuacion no tiene solucion
```