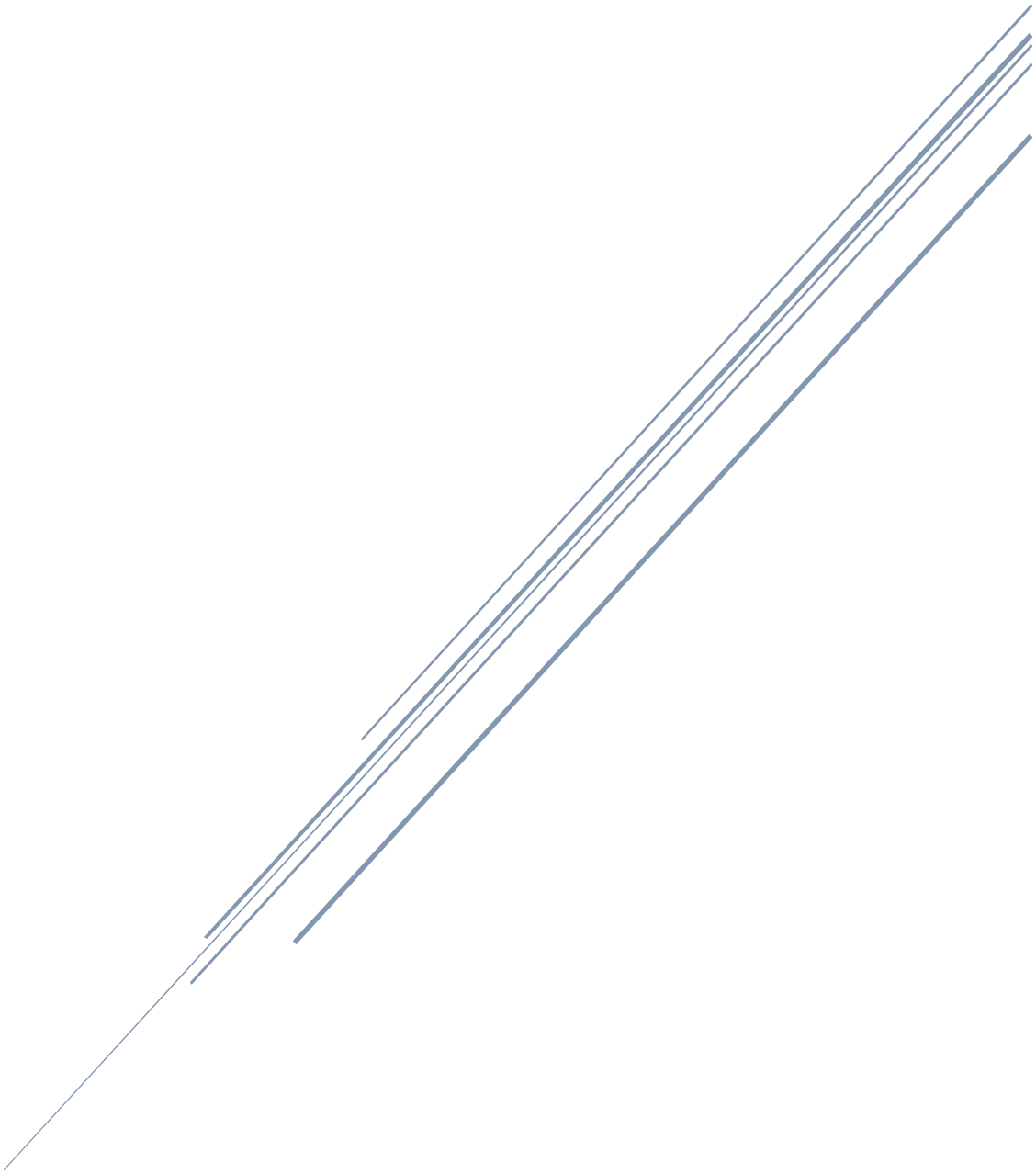
EJERCICOS DE PRACTICAS DE FUNDAMENTOS DEL SOFTWARE

FS PRACT



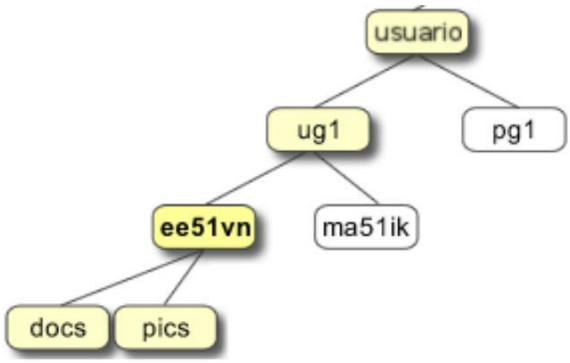
José Antonio Padial Molina 1ºGII A.3
Curso 2015/2016 ETSIIT UGR

Indice

Sesion 2

EJERCICIO 1

Crea el siguiente árbol de directorios a partir de tu directorio de usuario (donde usuario representa tu nombre de usuario o login en el sistema operativo). Crea también los ficheros vacíos **`pg1`** y **`ma51ik`** en los directorios que se indican en la ilustración:




```
mkdir ug1
touch pg1
cd ug1
mkdir ee51vn
touch ma51ik cd
ee51vn mkdir
docs pics
```

EJERCICIO 2

Mueve el directorio **`pics`** dentro del directorio **`ug1`**. Renombra finalmente dicho directorio a **`imagenes`**. Copia el directorio **`docs`** dentro de **`ug1`**.

```
cd $HOME
mv ./ug1/ee51vn/pics ./ug1/imagenes
cp -r ./ug1/ee51vn/docs ./ug1
```

EJERCICIO 3

Liste los archivos que estén en su directorio actual y fíjese en alguno que no disponga de la fecha actualizada, es decir, 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 -lF
touch ejemplo.txt
```

```
ls -lF
(la fecha se ha actualizado)
```

EJERCICIO 4

La organización del espacio en directorios es fundamental para poder localizar fácilmente aquello que estemos buscando. En ese sentido, realice las siguientes acciones dentro de su directorio home (es el directorio por defecto sobre el que trabajamos al entrar en el sistema):

a) Obtener en nombre de camino absoluto (pathname absoluto) de tu directorio home. ¿Es el mismo que el de tu compañero/a?

```
echo $HOME (no es el mismo que el compañero)
```

b) Crear 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 FS CA ALEM FP FFT
mkdir FS/practica1 FS/pratica2 FS/practica3
```

c) Dentro del directorio de la asignatura fundamentos del software (llamado FS) y dentro del directorio creado para esta sesión, copiar los archivos host y passwd que se encuentran dentro del directorio /etc.

```
cp /etc/hosts $HOME/FS
cp /etc/passwd $HOME/FS
```

d) Mostrar el contenido de cada uno de los archivos.

```
cat FS/hosts
cat FS/passwd
```

EJERCICIO 5

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 -l ????
```

b)
Todos los archivos que comiencen por los caracteres **d**, **f**.

```
ls -l [df]*
```

c)
Todos los archivos que comiencen por las parejas de caracteres **sa**, **se**, **ad**.

```
ls -l {sa,se,ad}*
```

d)
Todos los archivos que comiencen por **t** y acaben en **r**.

```
ls -l t*r
```

EJERCICIO 6

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

a)
de tu directorio home

```
ls -l ~/tem*.{gz,zip}
```

b)
del directorio actual

```
ls -l ./tem*.{gz,zip}
```

c)
¿hay alguna diferencia en el resultado de su ejecución? Explique el por qué si o el por qué no.

Si, uno usa rutas absolutas(aunque abreviada) y el otro no

EJERCICIO 7

Muestre del contenido de un archivo regular que contenga texto:

a)
las 10 primeras líneas.

```
head -n 5 jamon.txt
```

b)
las 5 últimas líneas.

```
tail -n 10 jamon.txt
```

EJERCICIO 8

Ordene el contenido de un archivo de texto según diversos criterios de ordenación.

Por orden alfabetico:

```
sort jamon.txt
```

Por orden numerico:

```
sort -n jamon.txt
```

Al reves:

```
sort -r jamon.txt
```

Aleatoriamente:

```
sort -R jamon.txt
```

EJERCICIO 9

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

```
ls -l ./*.{txt,c}
```

Sesion 3

EJERCICIO 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+x jamon.txt`

- Dar permiso de escritura y ejecución al "grupo".

```
chmod g+wx jamon.txt
```

- Quitar el permiso de lectura al "grupo" y al "resto de usuarios".

```
chmod go-r jamon.txt
```

- Dar permiso de ejecución al "propietario" y permiso de escritura el "resto de usuarios".

```
chmod u+x,o+w jamon.txt
```

- 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

Utilizando solamente las órdenes de la sesión anterior y los metacaracteres de redirección de salida:
* Crear un archivo llamado ej31 , que contendrá el nombre de los archivos del directorio padre del directorio de trabajo.

```
ls $HOME/.. > ej31
```

* Crear un archivo llamado ej32 , que contendrá las dos últimas líneas del archivo creado en el ejercicio anterior.

```
tail -n 2 ej31 > ej32
```

* Añadir al final del archivo ej32 , el contenido del archivo ej31 .

```
cat ej31 >> ej32
```

EJERCICIO 3

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

* Mostrar por pantalla, el listado (en formato largo) de los últimos 6 archivos del directorio padre en el directorio de trabajo.

```
ls -l $HOME|tail -n 6
```

* La orden wc muestra por pantalla el número de líneas, palabras y caracteres de un archivo (consulta la orden man para conocer más sobre ella). Utilizando dicha orden, mostrar 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 -m $HOME/[ef]*
```

EJERCICIO 4

Resuelva cada uno de los siguientes apartados.

a) Crear 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 “;”).

```
man chmod|tail -n 17 > ejercicio1
```

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

```
pwd >> ejercicio1
```

c) Usando la combinación de órdenes mediante paréntesis, crear un archivo llamado ejercicio3 que contendrá el listado de usuarios conectados al sistema (orden who) y la lista de archivos del directorio actual.

```
(who; ls -l) > ejercicio3
```

d) Añadir, 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 ejercicio1 &>> ejercicio3
```

e) Con una sola orden chmod, cambiar 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-w,ugo+x ejercicio{1,3}
```

Sesion 4

EJERCICIO 1

Escriba, al menos, cinco variables de entorno junto con el valor que tienen.

```
HOME=/home/superjes
USERNAME=superjes
LANGUAGE=es_ES:en
LOGNAME=superjes
DESKTOP_SESSION=gnome
```

EJERCICIO 2

Ejecute las órdenes del cuadro e indique qué ocurre y cómo puede resolver la situación para que la variable NOMBRE se reconozca en el shell hijo.

```
$ export NOMBRE=FS
$ echo $NOMBRE
$ FS
$ bash
$ echo $NOMBRE
$ FS
```

EJERCICIO 3

Compruebe qué ocurre en las expresiones del ejemplo anterior si se quitan las comillas dobles del final y se ponen después de los dos puntos. ¿Qué sucede si se sustituyen las comillas dobles por comillas simples?

-Si se ponen las dobles comillas después de los 2 puntos pasa que todo sale a continuación de lo anterior sin saltos de línea , con lo que no queda muy visual:

```
$ echo "Los archivos que hay en el directorio son:" $(ls -l)

Los archivos que hay en el directorio son: total 60 drwxr-xr-x 2 superjes superjes
4096 2011-10-11 08:50 Descargas drwxr-xr-x 2 superjes superjes 4096 2011-09-28
01:07 Documentos -rw-r--r-- 1 superjes superjes 9 2011-10-11 09:15 ej31 -rw-r--r--
1 [...]
```

-Si se sustituyen las comillas dobles por comillas simples pasa que se muestra literalmente lo que hay dentro de ellas, sin ejecutar ningún comando que contenga.

```
$ echo 'Los archivos que hay en el directorio son `ls -l`'
```

```
Los archivos que hay en el directorio son `ls -l`
```

EJERCICIO 4

Pruebe la siguiente asignación. ¿Qué ha ocurrido?

Ha ocurrido que como no hemos usado la orden "expr", no se reconoce el símbolo de la suma.

EJERCICIO 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"
```

EJERCICIO 6

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

```
#!/bin/bash
echo "Hola $@"
```

EJERCICIO 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\n" $VAR1 $VAR2 $VAR3
```

b) ¿Son variables locales o globales?

- *Son locales*

c) Borre la variable VAR2.

```
unset VAR2
```

d) Abra otra ventana de tipo terminal, ¿puede visualizar las dos variables restantes?

- *No, porque eran variables locales, sólo se podían ver en el terminal anterior.*

e) Cree una variable de tipo vector con los valores iniciales de las tres variables.

```
vector=(hola adios 14)
```

f) Muestre el segundo elemento del vector creado en el apartado e.

```
echo ${vector[1]}
```

EJERCICIO 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 .|wc -l'
```

Para que sea del directorio home del usuario que lo ejecuta:

```
alias ne='ls $HOME|wc -l'
```

EJERCICIO 9

Indique la línea de orden necesaria para buscar todos los archivos a partir del directorio 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
```

Para que se guarde en un archivo del directorio actual:

```
find /home -size -1 >> ./archivosP
```

EJERCICIO 10

Indique cómo buscaría todos aquellos archivos del directorio actual que contengan la palabra “ejemplo”.

```
find ./*ejemplo*
```

EJERCICIO 11

Complete la información de find y grep utilizando para ello la orden man.

```
man find:

-P      Never follow symbolic links.  This  is the  default  behaviour.
              When find examines or prints information a file, and the file is
              a symbolic link, the information used shall be  taken  from  the
              properties  of the symbolic link itself.


       -L Follow symbolic links. When find examines or prints information about
              files, the information used shall be taken from the prop- erties of
              the file to which the link points, not from the link itself (unless
              it is a broken symbolic link or find is unable to examine the file
              to which the link points). Use of this option implies -noleaf. If
              you later use the -P option, -noleaf will still be in effect. If -L
              is in effect and find discovers a symbolic link to a subdirectory
              during its search, the subdirec- tory pointed to by the symbolic
              link will be searched.

              When the -L option is in effect, the -type predicate will always
              match against the type of the file that a symbolic link points
              to rather than the link itself (unless the symbolic link is bro-
              ken). Using -L causes the -lname and -ilname predicates always
              to return false.


       -H Do not follow symbolic links, except while processing the com- mand
              line arguments. When find examines or prints information about
              files, the information used shall be taken from the prop- erties of
              the symbolic link itself. The only exception to this behaviour is
              when a file specified on the command line is a sym- bolic link, and
              the link can be resolved. For that situation, the information used
              is taken from whatever the link points to (that is, the link is
              followed). The information about the link itself is used as a
              fallback if the file pointed to by the sym- bolic link cannot be
              examined. If -H is in effect and one of the paths specified on the
              command line is a symbolic link to a directory, the contents of
              that directory will be examined (though of course -maxdepth 0 would
              prevent this).

-D debugoptions
              Print diagnostic information; this can be  helpful  to  diagnose
              problems  with why find is not doing what you want.   The list of
              debug options should be comma separated.  Compatibility  of   the
              debug  options  is not guaranteed between releases of  findutils.
              For a complete list of valid debug options, see   the output   of
              find -D help.  Valid debug options include

              help   Explain the debugging options

              tree   Show  the   expression  tree in its original and optimised
                     form.

              stat   Print messages as files are examined with  the  stat   and
```

lstat system calls. The find program tries to minimise such calls.

opt Prints diagnostic information relating to the optimisation of the expression tree; see the -O option.

rates Prints a summary indicating how often each predicate succeeded or failed.

-Olevel

Enables query optimisation. The find program reorders tests to speed up execution while preserving the overall effect; that is, predicates with side effects are not reordered relative to each other. The optimisations performed at each optimisation level are as follows.

0 Equivalent to optimisation level 1.

1 This is the default optimisation level and corresponds to the traditional behaviour. Expressions are reordered so that tests based only on the names of files (for example -name and -regex) are performed first.

2 Any -type or -xtype tests are performed after any tests based only on the names of files, but before any tests that require information from the inode. On many modern versions of Unix, file types are returned by readdir() and so these predicates are faster to evaluate than pred- icates which need to stat the file first.

3 At this optimisation level, the full cost-based query optimiser is enabled. The order of tests is modified so that cheap (i.e. fast) tests are performed first and more expensive ones are performed later, if necessary. Within each cost band, predicates are evaluated earlier or later according to whether they are likely to succeed or not. For -o, predicates which are likely to succeed are evalu- ated earlier, and for -a, predicates which are likely to fail are evaluated earlier.

-d A synonym for -depth, for compatibility with FreeBSD, NetBSD, MacOS X and OpenBSD.

-daystart

Measure times (for -amin, -atime, -cmin, -ctime, -mmin, and -mtime) from the beginning of today rather than from 24 hours ago. This option only affects tests which appear later on the command line.

-depth Process each directory's contents before the directory itself. The -delete action also implies -depth.

-follow

Deprecated; use the -L option instead. Dereference symbolic links. Implies -noleaf. The -follow option affects only those tests which appear after it on the command line. Unless the -H or -L option has been specified, the position of the -follow option changes the behaviour of the -newer predicate; any files listed as the argument of -newer will be dereferenced if they are symbolic links. The same consideration applies to -newerXY, -anewer and -cnewer. Similarly, the -type predicate will always match against the type of the file that a symbolic link points to rather than the link itself. Using -follow causes the -lname and -ilname predicates always to return false.

-help, --help
Print a summary of the command-line usage of find and exit.

-ignore_readdir_race
Normally, find will emit an error message when it fails to stat a file. If you give this option and a file is deleted between the time find reads the name of the file from the directory and the time it tries to stat the file, no error message will be issued. This also applies to files or directories whose names are given on the command line. This option takes effect at the time the command line is read, which means that you cannot search one part of the filesystem with this option on and part of it with this option off (if you need to do that, you will need to issue two find commands instead, one with the option and one without it).

-maxdepth levels
Descend at most levels (a non-negative integer) levels of directories below the command line arguments. -maxdepth 0 means only apply the tests and actions to the command line arguments.

-mindepth levels
Do not apply any tests or actions at levels less than levels (a non-negative integer). -mindepth 1 means process all files except the command line arguments.

-mount Don't descend directories on other filesystems. An alternate name for -xdev, for compatibility with some other versions of find.

-noignore_readdir_race
Turns off the effect of -ignore_readdir_race.

-noleaf
Do not optimize by assuming that directories contain 2 fewer subdirectories than their hard link count. This option is

needed when searching filesystems that do not follow the Unix directory-link convention, such as CD-ROM or MS-DOS filesystems or AFS volume mount points. Each directory on a normal Unix filesystem has at least 2 hard links: its name and its `.' entry. Additionally, its subdirectories (if any) each have a `..' entry linked to that directory. When find is examining a directory, after it has statted 2 fewer subdirectories than the directory's link count, it knows that the rest of the entries in the directory are non-directories (`leaf' files in the directory tree). If only the files' names need to be examined, there is no need to stat them; this gives a significant increase in search speed.

-regextype type
Changes the regular expression syntax understood by -regex and -iregex tests which occur later on the command line. Currently-implemented types are emacs (this is the default), posix-awk, posix-basic, posix-egrep and posix-extended.

-version, --version
Print the find version number and exit.

-warn, -nowarn
Turn warning messages on or off. These warnings apply only to the command line usage, not to any conditions that find might encounter when it searches directories. The default behaviour corresponds to -warn if standard input is a tty, and to -nowarn otherwise.

-xdev Don't descend directories on other filesystems.

man grep:

OPTIONS

Generic Program Information

--help Print a usage message briefly summarizing these command-line options and the bug-reporting address, then exit.

-V, --version
Print the version number of grep to the standard output stream. This version number should be included in all bug reports (see below).

Matcher Selection

-E, --extended-regexp
Interpret PATTERN as an extended regular expression (ERE, see below). (-E is specified by POSIX.)

-F, --fixed-strings
Interpret PATTERN as a list of fixed strings, separated by newlines, any of which is to be matched. (-F is specified by POSIX.)

-G, --basic-regexp
Interpret PATTERN as a basic regular expression (BRE, see below). This is the default.

-P, --perl-regexp
Interpret PATTERN as a Perl regular expression. This is highly experimental and grep -P may warn of unimplemented features.

Matching Control
-e PATTERN, --regexp=PATTERN
Use PATTERN as the pattern. This can be used to specify multiple search patterns, or to protect a pattern beginning with a hyphen (-). (-e is specified by POSIX.)

-f FILE, --file=FILE
Obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing. (-f is specified by POSIX.)

-i, --ignore-case
Ignore case distinctions in both the PATTERN and the input files. (-i is specified by POSIX.)

-v, --invert-match
Invert the sense of matching, to select non-matching lines. (-v is specified by POSIX.)

-w, --word-regexp
Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.

-x, --line-regexp
Select only those matches that exactly match the whole line. (-x is specified by POSIX.)

-y Obsolete synonym for -i.

General Output Control
-c, --count
Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines. (-c is specified by POSIX.)

--color[=WHEN], --colour[=WHEN]
Surround the matched (non-empty) strings, matching lines,

context lines, file names, line numbers, byte offsets, and separators (for fields and groups of context lines) with escape sequences to display them in color on the terminal. The colors are defined by the environment variable GREP_COLORS. The deprecated environment variable GREP_COLOR is still supported, but its setting does not have priority. WHEN is never, always, or auto.

-L, --files-without-match
Suppress normal output; instead print the name of each input file from which no output would normally have been printed. The scanning will stop on the first match.

-l, --files-with-matches
Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match. (-l is specified by POSIX.)

-m NUM, --max-count=NUM
Stop reading a file after NUM matching lines. If the input is standard input from a regular file, and NUM matching lines are output, grep ensures that the standard input is positioned to just after the last matching line before exiting, regardless of the presence of trailing context lines. This enables a calling process to resume a search. When grep stops after NUM matching lines, it outputs any trailing context lines. When the -c or --count option is also used, grep does not output a count greater than NUM. When the -v or --invert-match option is also used, grep stops after outputting NUM non-matching lines.

-o, --only-matching
Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.

-q, --quiet, --silent
Quiet; do not write anything to standard output. Exit immediately with zero status if any match is found, even if an error was detected. Also see the -s or --no-messages option. (-q is specified by POSIX.)

-s, --no-messages
Suppress error messages about nonexistent or unreadable files. Portability note: unlike GNU grep, 7th Edition Unix grep did not conform to POSIX, because it lacked -q and its -s option behaved like GNU grep's -q option. USG-style grep also lacked -q but its -s option behaved like GNU grep. Portable shell scripts should avoid both -q and -s and should redirect standard and error output to /dev/null instead. (-s is specified by POSIX.)

Output Line Prefix Control
-b, --byte-offset
Print the 0-based byte offset within the input file before each line of output. If -o (--only-matching) is specified, print the offset of the matching part itself.

-H, --with-filename
Print the file name for each match. This is the default when there is more than one file to search.

-h, --no-filename
Suppress the prefixing of file names on output. This is the default when there is only one file (or only standard input) to search.

--label=LABEL
Display input actually coming from standard input as input coming from file LABEL. This is especially useful when implementing tools like zgrep, e.g., gzip -cd foo.gz | grep --label=foo -H something. See also the -H option.

-n, --line-number
Prefix each line of output with the 1-based line number within its input file. (-n is specified by POSIX.)

-T, --initial-tab
Make sure that the first character of actual line content lies on a tab stop, so that the alignment of tabs looks normal. This is useful with options that prefix their output to the actual content: -H,-n, and -b. In order to improve the probability that lines from a single file will all start at the same column, this also causes the line number and byte offset (if present) to be printed in a minimum size field width.

-u, --unix-byte-offsets
Report Unix-style byte offsets. This switch causes grep to report byte offsets as if the file were a Unix-style text file, i.e., with CR characters stripped off. This will produce results identical to running grep on a Unix machine. This option has no effect unless -b option is also used; it has no effect on platforms other than MS-DOS and MS-Windows.

-Z, --null
Output a zero byte (the ASCII NUL character) instead of the character that normally follows a file name. For example, grep -lZ outputs a zero byte after each file name instead of the usual newline. This option makes the output unambiguous, even in the presence of file names containing unusual characters like newlines. This option can be used with commands like find -print0, perl -0, sort -z, and xargs -0 to process arbitrary file names, even those that contain newline characters.

Context Line Control
-A NUM, --after-context=NUM
Print NUM lines of trailing context after matching lines. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.

-B NUM, --before-context=NUM
Print NUM lines of leading context before matching lines. Places a line containing a group separator (--) between

contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.

-C NUM, -NUM, --context=NUM

Print NUM lines of output context. Places a line containing a group separator (--) between contiguous groups of matches. With the -o or --only-matching option, this has no effect and a warning is given.

File and Directory

Selection -a, --text

Process a binary file as if it were text; this is equivalent to the --binary-files=text option.

--binary-files=TYPE

If the first few bytes of a file indicate that the file contains binary data, assume that the file is of type TYPE. By default, TYPE is binary, and grep normally outputs either a one-line message saying that a binary file matches, or no message if there is no match. If TYPE is without-match, grep assumes that a binary file does not match; this is equivalent to the -I option. If TYPE is text, grep processes a binary file as if it were text; this is equivalent to the -a option. Warning: grep --binary-files=text might output binary garbage, which can have nasty side effects if the output is a terminal and if the terminal driver interprets some of it as commands.

-D ACTION, --devices=ACTION

If an input file is a device, FIFO or socket, use ACTION to process it. By default, ACTION is read, which means that devices are read just as if they were ordinary files. If ACTION is skip, devices are silently skipped.

-d ACTION, --directories=ACTION

If an input file is a directory, use ACTION to process it. By default, ACTION is read, which means that directories are read just as if they were ordinary files. If ACTION is skip, directories are silently skipped. If ACTION is recurse, grep reads all files under each directory, recursively; this is equivalent to the -r option.

--exclude=GLOB

*Skip files whose base name matches GLOB (using wildcard matching). A file-name glob can use *, ?, and [...] as wildcards, and \ to quote a wildcard or backslash character literally.*

--exclude-from=FILE

Skip files whose base name matches any of the file-name globs read from FILE (using wildcard matching as described under --exclude).

--exclude-dir=DIR

Exclude directories matching the pattern DIR from recursive searches.

```
       -I     Process  a  binary  file as if it did not contain matching data;
              this is equivalent to the --binary-files=without-match option.

       --include=GLOB
              Search only files whose base name matches GLOB  (using  wildcard
              matching as described under --exclude).

       -R, -r, --recursive
              Read  all  files  under  each  directory,  recursively;  this is
              equivalent to the -d recurse option.

   Other Options
       --line-buffered
              Use line buffering on output.  This  can  cause  a  performance
              penalty.

       --mmap If  possible, use the mmap(2) system call to read input, instead
              of the default read(2) system call.  In some situations,  --mmap
              yields  better performance.  However, --mmap can cause undefined
              behavior (including core dumps) if an input file  shrinks  while
              grep is operating, or if an I/O error occurs.

       -U, --binary
              Treat  the  file(s) as binary.  By default, under MS-DOS and MS-
              Windows, grep guesses the file type by looking at  the  contents
              of  the first 32KB read from the file.  If grep decides  the file
              is a text file, it strips the CR characters  from  the  original
              file  contents  (to  make  regular expressions with ^ and $ work
              correctly).  Specifying -U overrules this guesswork, causing all
              files  to be read and passed to the matching mechanism verbatim;
              if the file is a text file with CR/LF pairs at the end  of  each
              line,  this  will  cause some regular expressions to fail.  This
              option has no effect on platforms  other  than  MS-DOS  and  MS-
              Windows.

       -z, --null-data
              Treat  the  input  as  a set of lines, each terminated by a zero
              byte (the ASCII NUL character) instead of a newline.  Like  the
              -Z  or --null option, this option can be used with commands like
              sort -z to process arbitrary file names.
```

EJERCICIO 12

Indique cómo buscaría si un usuario dispone de una cuenta en el sistema.

```
#!/bin/bash

cat /etc/passwd | cut -d ":" -f 1 | grep $1 &> /dev/null
var="$?"

if [ "$var" == "0" ]; then
      echo "El usuario $1 tiene cuenta en el sistema"

else
      echo "El usuario $1 NO tiene cuenta en el sistema"

fi
```

EJERCICIO 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.

```
#!/bin/bash
num_archivos=`ls -lR $HOME|cut -d " " -f 1|grep -e '-[-w][-x]$'|wc -l`

echo "Hay $num_archivos archivos que no tienen permiso de lectura para el resto de usuarios en todo tu arbol de directorios"
```

EJERCICIO 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
ls $1|wc -l
```

Sesion 5

EJERCICIO 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 $[…].

Primera forma:

```
$ anio=365
$ dia=`date +%j`
$ var=`expr $anio - $dia`
$ echo "Faltan `expr $var / 7` semanas hasta el fin de año"
```

Segunda forma:

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

Tercera forma:

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

Combinación de ellas:

```
$ echo "Faltan `expr $[$anio - $dia] / 7` semanas hasta el fin de año"
```

EJERCICIO 2

Realice las siguientes operaciones para conocer el funcionamiento del operador de incremento como sufijo y como prefijo. Razone el resultado obtenido en cada una de ellas:

```
$ v=1
$ echo $v          //vale 1
$ echo $((v++))    //vale 2 , pero aparece 1 en pantalla
$ echo $v          //vale 2
$ echo $((++v))    //vale 3, y aparece 3 en pantalla
$ echo $v          //vale 3
```

EJERCICIO 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=25
$ y=5
$ echo $[x /= $y]
$ echo $x          //valdrá 5
```

EJERCICIO 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   //muestra 1.20000000000
$ echo '6/5|bc  -l' //muestra literalmente lo que hay entrecomillado
$ echo "6/5|bc  -l" //muestra literalmente lo que hay entrecomillado
$ echo '6/5'|bc -l  //muestra  1.20000000000
$ echo  "6/5"|bc -l //muestra  1.20000000000
```

EJERCICIO 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      //Error sintactico
$ echo '(3-2)/5'|bc -l    //Muestra  .2000000000
$ echo "(3-2)/5"|bc -l    //Muestra  .2000000000
$ echo `(3-2)/5`|bc -l    //Error sintactico, lo entrecomillado no es un comando
```

EJERCICIO 6

Consulte la sintaxis completa de la orden **let** utilizando la orden de ayuda para las órdenes empotradas (**help let**) y tome nota de su sintaxis general.

```
$ help let
let: let arg [arg ...]
    Evaluate arithmetic expressions.

    Evaluate each ARG as an arithmetic expression. Evaluation is done in
    fixed-width integers with no check for overflow, though division by 0
    is trapped and flagged as an error. The following list of operators is
    grouped into levels of equal-precedence operators. The levels are listed
    in order of decreasing precedence.
```

```
      id++, id--   variable post-increment, post-decrement
      ++id, --id   variable pre-increment, pre-decrement
      -,  +        unary minus, plus
      !,  ~        logical and bitwise negation
      **           exponentiation
      *, /, %            multiplication, division, remainder
      +, -         addition, subtraction
      <<, >>       left and right bitwise shifts
      <=, >=, <, > comparison
      ==, !=       equality, inequality
      &            bitwise AND
      ^            bitwise XOR
      |            bitwise OR
      &&           logical AND
      ||           logical OR
      expr ? expr : expr
                   conditional operator
      =, *=, /=, %=,
      +=, -=, <<=, >>=,
      &=, ^=, |=   assignment

    Shell variables are allowed as operands. The name of the variable
    is replaced by its value (coerced to a fixed-width integer) within
    an expression. The variable need not have its integer attribute
    turned on to be used in an expression.

    Operators are evaluated in order of precedence. Sub-expressions in
    parentheses are evaluated first and may override the precedence
    rules above.

    Exit Status:
    If the last ARG evaluates to 0, let returns 1; let returns 0 otherwise.
```

EJERCICIO 7

Al realizar el ejercicio anterior habrá observado que la orden **let** admite asignaciones múltiples y 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.

(Se vera en la sesión siguiente)

EJERCICIO 8

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

if [ $1 -eq 1
] then
      echo "La primera expresion es cierta: $1"
else
      echo "La primera expresion es falsa: $1"
fi

if [ $2 -eq 1
] then
      echo "La segunda expresion es cierta: $2"
else
      echo "La segunda expresion es falsa: $2"
fi

if [ $1 -lt $2 ]
then
      echo "El primer parametro es mas pequeño que el
segundo" elif [ $1 -gt $2 ]
then
      echo "El primer parametro es mas grande que el segundo"
else
      echo "Son iguales"
fi
```

EJERCICIO 9

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

es_plano=`test -f $1 && test -x $1 && echo true || echo false`
es_enlace=`test -h $1 && echo true || echo false`

echo "El parametro pasado: "

if [ $es_plano == "true" ]
then
      #Dado el enunciado aqui no contemplamos el caso de que solo sea
      #verdad una de estas condiciones
      echo "Es un archivo plano y con permiso de ejecucion"
else

      echo "No es un archivo plano y no tiene permiso de ejecucion"
fi

if [ $es_enlace == "true" ]
then
      echo "Es un enlace simbolico"
else
      echo "No es un enlace simbolico"
fi
```

EJERCICIO 10

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.

```
test: test [expresión]
    Evaluate conditional expression.

    Exits with a status of 0 (true) or 1 (false) depending on
    the evaluation of EXPR. Expressions may be unary or binary. Unary
    expressions are often used to examine the status of a file. There
    are string operators and numeric comparison operators as well.

    The behavior of test depends on the number of arguments. Read the
    bash manual page for the complete specification.

    File operators:

      -a FILE        True if file exists.


      -b FILE        True if file is block special.
```

```
    -c FILE        True if file is character special.
    -d FILE        True if file is a directory.
    -e FILE        True if file exists.
    -f FILE        True if file exists and is a regular file.
    -g FILE        True if file is set-group-id.
    -h FILE        True if file is a symbolic link.
    -L FILE        True if file is a symbolic link.
    -k FILE        True if file has its `sticky' bit set.
    -p FILE        True if file is a named pipe.
    -r FILE        True if file is readable by you.
    -s FILE        True if file exists and is not empty.
    -S FILE        True if file is a socket.
    -t FD          True if FD is opened on a terminal.
    -u FILE        True if the file is set-user-id.
    -w FILE        True if the file is writable by you.
    -x FILE        True if the file is executable by you.
    -O FILE        True if the file is effectively owned by you.
    -G FILE        True if the file is effectively owned by your group.
    -N FILE        True if the file has been modified since it was last read.

    FILE1 -nt FILE2 True if file1 is newer than file2 (according to
                    modification date).

    FILE1 -ot FILE2  True if file1 is older than file2.

    FILE1 -ef FILE2  True if file1 is a hard link to file2.

  All file operators except -h and -L are acting on the target of a symbolic
  link, not on the symlink itself, if FILE is a symbolic link.

  String operators:

    -z STRING      True if string is empty.

    -n STRING
       STRING      True if string is not empty.

    STRING1 = STRING2
                   True if the strings are equal.
    STRING1 != STRING2
                   True if the strings are not equal.
    STRING1 < STRING2
                   True if STRING1 sorts before STRING2 lexicographically.
    STRING1 > STRING2
                   True if STRING1 sorts after STRING2 lexicographically.

  Other operators:

    -o OPTION      True if the shell option OPTION is enabled.
    -v VAR  True if the shell variable VAR is set
    ! EXPR         True if expr is false.
    EXPR1 -a EXPR2 True if both expr1 AND expr2 are true.
    EXPR1 -o EXPR2 True if either expr1 OR expr2 is true.

    arg1 OP arg2   Arithmetic tests.  OP is one of -eq, -ne,
```

```
                    -lt, -le, -gt, or -ge.

    Arithmetic binary operators return true if ARG1 is equal, not-equal,
    less-than, less-than-or-equal, greater-than, or greater-than-or-
    equal than ARG2.

    See the bash manual page bash(1) for the handling of parameters
    (i.e. missing parameters).

    Exit Status:
    Returns success if EXPR evaluates to true; fails if EXPR evaluates
    to false or an invalid argument is given.
```

```
Pruebas con cadenas:

$ test hola = hola
$ echo $?
$ 0                #Verdadero
$ test hola = adios
$ echo $?
$ 1                #Falso

Pruebas con expresiones numericas:

 $ test $[6-3] -eq $[10-7]
 $ echo $?
 $ 0                #Verdadero 3 = 3
 $ test $[6-3] -eq $[10+7]
 $ echo $?
 $ 1                #Falso 3 =/= 17
```

EJERCICIO 11

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
```

- Verifica si el archivo "sesion5.pdf" del directorio actual es un archivo plano, si es así imprime que el archivo existe.

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 && test -d /bin ; then printf "El archivo ./sesion5.pdf existe y /bin es un directorio"; elif test -f ./sesion5.pdf && ! test -d /bin ; then printf "El archivo ./sesion5.pdf existe y /bin no es un directorio"; elif ! test -f ./sesion5.pdf && test -d /bin ; then printf "No existe el archivo ./sesion5.pdf y /bin es un directorio"; else printf "Ni existe ./sesion5.pdf ni /bin es un directorio"; 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 "No existe el archivo $1 y $2 es un directorio\n"
else
      printf "Ni existe $1 ni $2 es un directorio\n"
fi
```

EJERCICIO 12

Construya un guion que admita como argumento el nombre de un usuario del sistema y que permita saber si ese usuario es el propietario del archivo /bin/ls y si tiene permiso de lectura sobre él.

```
#!/bin/bash

propietario=`ls -lF /bin/ls|cut -d " " -f 3`
lectura=`ls -lF /bin/ls|grep -e '^.r'|wc -l`


if [ $lectura -eq 1
] then
      printf "El usuario tiene permiso de lectura\n"
else
      printf "El usuario NO tiene permiso de lectura\n"
fi

if [ $1 == $propietario
] then
      printf "El usuario es propietario de /bin/ls\n"
else
      printf "El usuario NO es propietario de /bin/ls\n"
fi
```

EJERCICIO 13

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.

```
#!/bin/bash

if [[ $1 == "-h" ]]
then
	printf "Este es el manual\n"
else
	anio=365
	dia=`date +%j`
	restantes=`expr $anio - $dia`
	printf "Faltan $restantes dias hasta el fin de año\n"

	if [ `expr $restantes % 5` == 0 ]
	then
		printf "Ademas, ese numero es multiplo de 5\n"
	else
		printf "Pero ese numero no es multiplo de 5\n"
	fi
fi
```

EJERCICIO 14

¿Qué pasa en el ejemplo anterior si eliminamos la redirección de la orden **if**?

- *Si se borra la redirección pasa que el mensaje de error que emite el sistema operativo , por ejemplo si el archivo no existe, se muestra en pantalla además de nuestro mensaje. Si no lo eliminamos no se muestra, ya que se envía a /dev/null.*

EJERCICIO 15

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 [ `echo $1 | grep -e '^.$'` ]
then
	echo "Es un unico caracter"

	if [[ $1 =~ [A-Z] ]]
	then
		echo "Y es una letra mayuscula"
	elif [[ $1 =~ [a-z] ]]
	then
		echo "Y es una letra minuscula"
	else
		echo "No es una letra, es otro caracter"
	fi

else
	echo "No es un unico caracter"
fi
```

EJERCICIO 16

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?

```
find / -regex '.*\([0-9]+.*e\).*'
find . -regex '.*e.*' -and -not -regex '.*[01].*'
```

EJERCICIO 17

Utilizando la orden grep, exprese una forma alternativa de realizar lo mismo que con wc -l.

```
grep –c
```

EJERCICIOS S5_COMPLEMENTOS.TXT

```
grep    .           s5_datos    #Busca todas las lineas que contengan cualquier
cosa
grep    \.          s5_datos    #Igual que el anterior
grep  p.p.          s5_datos    #Busca las lineas que contengan palabras como
papa,pepa,pipi,etc..
grep  P.p.          s5_datos    #Busca las lineas que contengan palabras como
Papa,Pepa,Pipi,etc...
grep  pi            s5_datos    #Busca las lineas que contengan pi
grep  pi\.          s5_datos    #Igual que el anterior
grep  'l?'          s5_datos    #Busca las lineas que contengan l?
grep  l?            s5_datos    #Igual que el anterior
grep  l\?           s5_datos    #Igual que el anterior
grep  “\(ll\)\?”    s5_datos    #Busca las lineas en las que aparece ll o no, o
sea, todas
grep  'pa\?'        s5_datos    #Busca las lineas que contienen una p y puede o no
tiener una a
grep  'pa\?p'       s5_datos    #Busca las lineas que contienen una p, puede o no
tener una a y luego una p
grep  '\(pa\)+'     s5_datos    #Busca las lineas que contienen pa+
grep  pa*p          s5_datos    #Busca las lineas que contienen pa[lo que sea]p
grep  'pa\{2\}p'    s5_datos    #Busca las lineas que contienen la letra p, a dos
veces,seguido de una p
grep  [1-6]         s5_datos    #Busca las lineas que contienen algun numero del 1
al 6
grep  [6-0]         s5_datos    #Rango invalido
grep  [q-t]         s5_datos    #Busca las lineas que tienen una letra que va de
la q a la t
grep  [Q-T]         s5_datos
grep  [q-sy]        s5_datos    #Busca las lineas que tienen una letra que esta
entre la q y la s, o una y
grep  [^QT]         s5_datos
grep  ^P            s5_datos    #Busca las lineas que empiezan por P
grep  -v  ^[^p]     s5_datos    #Busca las lineas que empiezan por p
grep  ^[^p]         s5_datos    #Busca las lineas que no empiezan por p
grep  e$            s5_datos    #Busca las lineas que acaban en e
grep  'e\b'         s5_datos    #Busca las lineas que tienen una e al principio o
final de una palabra
grep  “e\>”         s5_datos    #Busca las lineas que tienen una e al final de una
palabra
grep  'e\B'         s5_datos    #Busca las lineas que tienen una e entre dos otros
caracteres
grep  '\<e'         s5_datos    #Busca las lineas que tienen una e al principio de
una palabra
grep  -E 'e$|s$'    s5_datos    #Busca las lineas que tienen alguna palabra que
acaba en e o s.

grep  -E 'e$ | s$'  s5_datos    #Si hay espacios no hace nada porque buscas
espacios literalmente.
```

Seguidamente se proponen diferentes ejercicios que deben resolverse haciendo uso de la orden grep sobre el archivo s5_datos. Obtenga las líneas que contienen:

1.- Algún punto (.).

```
$ grep '\.' s5_datos
```

2.- El carácter de la barra de escape (\ ).

```
$ grep '\\' s5_datos
```

3.- "pi" seguido de cualquier otro carácter.

```
$ grep pi\. s5_datos
```

4.- Palabras con la letra “p” seguida de otros tres caracteres.

```
$ grep p\... s5_datos
```

5.- Palabras con “pe.”.

```
$ grep 'pe\.' s5_datos
```

6.- Palabras que incluyen la letra ele al menos una vez.

```
$ grep 'l\+' s5_datos
```

7.- Palabras que incluyen una o varias letras “a” entre dos letras “p”.

```
$ grep 'pa\+p' s5_datos
```

8.- Palabras con “pa?”.

```
$ grep 'pa?' s5_datos
```

9.- Palabras con “pa\?p”.

```
$ grep 'pa\\?p' s5_datos
```

10.- Palabras que incluyen “pa” al menos una vez.

```
$ grep '\(pa\)\+' s5_datos
```

11.- Palabras que incluyen de dos a cinco letras “a” entre dos letras “p”.

```
$ grep 'pa\{2,5\}p' s5_datos
```

12.- Palabras que incluyen las letras “q” o “t”.

```
$ grep [qt] s5_datos
```

13.- Palabras que incluyen las letras “q” o “t” o una coma.

```
$ grep [qt\,] s5_datos
```

14.- Palabras que incluyen la letra “s” o un guion (-).

```
$ grep [s-] s5_datos
```

15.- El carácter de comentario o un asterisco como primer carácter de línea.

```
$ grep '^[#*]' s5_datos
```

16.- Palabras de cinco letras que comiencen con la letra “b” y terminen con la letra “o”.

```
$ grep '^b.\{3\}o$' s5_datos
```

17.- Palabras de cualquier longitud que comiencen con la letra “p” y terminen con la letra “a”.

```
$ grep '^p.*a$' s5_datos
```

18.- La comilla simple ('). ¿Y si se quieren omitir estas líneas?

```
$ grep "'" s5_datos
#para que sea alreves $
grep -v "'" s5_datos
```

19.- La comilla doble (“).

```
$ grep '"' s5_datos
```

20.- La letra “e” y la letra “i” en cualquier orden o posición.

```
$ grep '.*\(e\|i\).*' s5_datos
```

Sesion 6

EJERCICIO 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 es 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

es_directorio=`test -d $1 && echo true || echo false` es_entero=`echo
$2 | grep -q "[0-9]\+" && echo true || echo false`
#el modificador -q del grep de antes es para que no se muestre en pantalla
al hacerlo

if [ $# -eq 2
] then
	if [ $es_directorio == "true" ] && [ $es_entero == "true" ]
	then
		find $1 -size -$2 > archivosSizN.txt
		echo "Fichero archivosSizN.txt creado correctamente"
	else
		echo "El directorio no existe o el parametro segundo no es un entero"
	fi

else
	echo "Numero incorrecto de parametros"
fi
```

EJERCICIO 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.

```
#!/bin/bash

es_directorio=`test -d $1 && echo true || echo false`

if [ $# -eq 1
] then
	if [ $es_directorio == "true" ]
	then
		for i in `ls $1`
		do
			es_archivo=`test -f $i && echo true || echo false`
```

```
                es_enlace=`test -h $i && echo true || echo false`
                es_subdirectorio=`test -d $i && echo true || echo false`

                if [ $es_enlace == "true" ]
                then
                        echo "Enlace : $i"
                elif [ $es_archivo == "true" ]
                then
                        echo "Archivo regular : $i"
                elif [ $es_subdirectorio == "true" ]
                then
                        echo "Directorio : $i"
                fi
        done
    else
        echo "El directorio no existe"
    fi

else
    echo "Numero incorrecto de parametros"
fi
```

EJERCICIO 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 la Z, X, C, V, ... serán de la fila 4. La captura de la tecla se realizará mediante la orden read. (vea sección 6 en página 6).

```
#!/bin/bash
printf "Pulsa una tecla\n"
read tecla

case $tecla in
    [º0-9\'¡]) echo "Primera fila" ;;
    [qwertyuiop\`+]) echo "Segunda fila" ;;
    [asdfghjklñ´ç]) echo "Tercera fila" ;;
    [\<zxcvbnm,.-]) echo "Cuarta fila" ;;
    *) echo "Otra linea" ;;
esac
```

EJERCICIO 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
printf "Elige un mes del 1 al 12\n"
read mes
```

```
case $mes in
 1) echo "Enero" ;;
 2) echo "Febrero" ;;
 3) echo "Marzo" ;;
 4) echo "Abril" ;;
 5) echo "Mayo" ;;
 6) echo "Junio" ;;
 7) echo "Julio" ;;
 8) echo "Agosto" ;;
 9) echo "Septiembre" ;;
 10) echo "Octubre" ;;
 11) echo "Noviembre" ;;
 12) echo "Diciembre" ;;
 *) echo "Elige una opcion dentro del rango"
;; esac
```

EJERCICIO 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** (ver página 6).

```
#!/bin/bash

num=0

while [ $num -lt 1 ] || [ $num -gt 10
] do
      printf "Mete un numero del 1 al 10\n"
      read num
done
```

EJERCICIO 6

Copie este ejercicio y pruébelo en su sistema para ver su funcionamiento. ¿Qué podemos modificar para que el giro se vea más rápido o más lento? ¿Qué hace la opción -e de las órdenes echo del guion?

-Para que el giro sea más rápido solo hay que cambiar el valor de la variable INTERVAL, que vale 1, y poner un valor más pequeño.

-La orden -e de los echo es para poder interpretar las barras invertidas como caracteres.

EJERCICIO 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 y para poder filtrar la información que nos interesa nos será útil la orden siguiente:

```
cut -d':' -f1
```

que aplicada de forma encauzada con cualquier orden para mostrar el contenido del citado archivo, cortará por la columna 1 y hasta que aparezca el delimitador ":".

```
#!/bin/bash

for linea in `cat /etc/passwd`
do
	usuario=`echo $linea|cut -d: -f1`
	shell=`echo $linea|cut -d: -f7`
	valor="/bin/$1"

	if [[ $valor == `echo $shell` ]]
	then
		echo $usuario >> temp.txt

	fi
done

cat temp.txt|sort
rm temp.txt
```

EJERCICIO 8

Dos órdenes frecuentes de Unix son tar y gzip. La orden tar permite almacenar/extraer varios archivos de otro archivo. Por ejemplo, podemos almacenar el contenido de un directorio en un archivo con

```
tar -cvf archivo.tar directorio
```

(la opción -x extrae los archivos de un archivo .tar).

```
#!/bin/bash
#Nombre del script: cpback.sh

destino="CopiasSeguridad"

if [[ `test -d $destino && echo 0 || echo 1` == 1 ]]
	then
		mkdir $destino
fi

if [ $# -ge 1
] then
	#Declaro una variable suma para saber si ha habido algun error, si uno
	#de los parametros no existe..suma ya no valdra 0.

	suma=0
	for i in $@
	do
		if [ `test -e $i && echo 0 || echo 1` == 1
		] then
			suma=$[$suma + 1]
		fi
	done

	#Solo entro al bucle si suma vale 0, es decir, si no hay errores
```

```
	if [ $suma -eq 0 ]
	then
		tar -cvf "./$destino/copia`date +"%Y%m%d"`.tar"
		$@ gzip "./$destino/copia`date +"%Y%m%d"`.tar"
	else
		echo "Alguno de los parametros dados no existe"
	fi
else
	echo "Numero de paramentros incorrecto"
fi
```

EJERCICIO 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 el 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 guión 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
#Nombre del fichero: newdirfiles

if [ $# -eq 3
] then
	#Comprueba que el numero,parametro 2, sea del 1 al 99
	num_files=$2
	if [ `echo $num_files|grep "^[1-9][0-9]\?$"` ]
	then

		#Creo el directorio si no existe
		dirname=$1

		if [[ `test -d $dirname && echo 0 || echo 1` == 1 ]]
```

```
                then
                        mkdir $dirname
        fi

        # y le doy permisos necesarios
        chmod u+rw $dirname

        basefilename=$3

        cd $dirname

        #Recorro del 1 hasta num_files
        for i in `seq 1 1 $num_files`
        do
                #Esto solo sirve para poner el 0 en los menores de 9. Ej: 01
                if [ $i -lt 10 ]
                then
                        touch $basefilename"0"$i
                else
                        touch $basefilename$i
                fi
        done

        echo "Archivo/s creado/s correctamente"

    else
        echo "El numero ha de ser del 1 al 99"
    fi
else
    echo "Numero incorrecto de parametros"
fi
```

Sesion 7

EJERCICIO 1

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

```
#!/bin/bash
# Uso: pathmas directorio [after]
if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)"
then
      if [[ $2 == "after" ]] #<<<<<<<<<<<<< Faltaban dobles corchetes
      then
            PATH=$PATH:$1
      else
            PATH=$1:$PATH
      fi
else
      echo "$1 ya está en el path"
fi
```

_____

Para ejecutar este guión , el cual modifica la variable PATH, se ha de usar el comando "source", ya que si no se pone la variable PATH cambiará en el shell hijo, no en el padre, y no notaremos cambio alguno.

Ej:

```
$ source ./pathmas.sh /dev/null after
```

EJERCICIO 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 sesión 6.

Los guiones de la sesión 6 están correctos, no ha hecho falta usar métodos de depuración,

en ciertos puntos donde he tenido dudas he puesto "echo" de determinadas variables para saber su valor en ese punto del script y me ha bastado.

EJERCICIO 3

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

```
#!/bin/bash

#Hago una lista de los procesos(solo 1 iteracion,no que lo mire constantemente)
y lo meto en un fichero para luego procesar datos
top -bn 1 > temporal

#Cuento las lineas del fichero anterior y creo otra variable que sea ese
numero restandole 8, que son las 8 primeras lineas inutiles
numero_lineas=`cat monoloco|wc -l`
lineas_validas=$[ $numero_lineas - 8 ]

#Hago un tail de esas n lineas y ordeno por el campo 10(que indica el consumo de
memoria) y lo meto todo a otro archivo temporal
cat temporal|tail -n $lineas_validas|sort -k 10 > archivo

#Cojo la ultima linea de este nuevo fichero, que como está ordenado contendrá el
proceso que más memoria ocupa...y corto el campo que contiene el nombre, no sin
antes indicar con 'tr -s " " " "' que ignore los espacios que hay entre los
campos y los convierta en uno sólo.
cat archivo|tail -n 1|tr -s " " " "|cut -d " " -f 13

#Borro los dos archivos temporales creados
rm temporal
rm archivo
```

EJERCICIO 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

i=1

while true
do
      printf "$i "
      sleep 1
      i=$[$i+1]
done
```

-Para parar este proceso desde otro terminal, se mira con "ps -aux" la lista de procesos completa, buscamos el PID del que queremos parar y ponemos:

```
$ kill -STOP PID_proceso
```

```
Ej:
```

```
$ kill -STOP ‘ps -A -o pid,cmd | grep nombre_guion | head -n 1 | cut -d " " -f 1‘
```

Para reanudarlo:

```
$ kill -CONT PID_proceso
```

```
Ej:
$ kill -CONT ‘ps -A -o pid,cmd | grep nombre_guion | head -n 1 | cut -d " " -f 1‘
```

Para matarlo:

```
$ kill -TERM PID_proceso
```

```
Ej:
$ kill -TERM ‘ps -A -o pid,cmd | grep nombre_guion | head -n 1 | cut -d " " -f 1‘
```

EJERCICIO 5

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

Sí, usando la orden "jobs" vemos un listado de los procesos en segundo plano y su estado.

Hacemos "kill -9 %n" ,donde "n" es el numero asignado al proceso y se finalizará.

También se podría hacer desde otro terminal buscando el PID del proceso y haciendo "kill -9 PID_proceso".

EJERCICIO 6

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

```
top -u nombre_usuario
```

Ejercicios adicionales

EJERCICIO 1

Construir un guion para que borre las líneas en blanco de un archivo de texto. El guion deberá comprobar:

a)Que se especifica un archivo. En caso contrario, se recuerda la sintaxis.

b)Si se especifica un nombre de archivo, comprobar que éste existe

```
#!/bin/bash

if [ $# -eq 1
] then
	es_archivo=`test -f $1 && echo true|| echo false`;
	if [ $es_archivo == "true" ]
	then

		cat $1 | grep -v "^[ ]*$" > temporal
		rm $1
		mv temporal $1
	else
		echo "No es un archivo regular"
	fi
else
	echo "Numero incorrecto de paramentros"
fi
```

EJERCICIO 2

Construir un guion que inserte una línea en blanco después de un párrafo (línea que finaliza en un ".". Hacer las comprobaciones pertinenes.

```
#!/bin/bash

if [ $# -eq 1
] then
	es_archivo=`test -f $1 && echo true || echo false`;
	if [ $es_archivo == "true" ]
	then
		while read linea
		do
			echo $linea >> temporal
			if [[ `echo $linea | grep '\.$'` ]]
```

```
                then
                        echo " " >> temporal
                fi
        done < $1

        mv temporal $1
    else
        echo "No es un archivo regular"
    fi
else
    echo "Numero incorrecto de paramentros"
fi
```

EJERCICIO 3

Construir un guion del shell que realice la función básica de la orden cat: muestre el contenido de uno o varios archivos que se pasan como argumento(s). Si no se pasa ningún argumento, muestra lo que se escriba en la entrada estándar.

```
#!/bin/bash

if [ $# -ge 1
] then
    for i in $@
    do
        es_archivo=`test -f $i && echo true || echo
        false`; if [ $es_archivo == "true" ]
        then
            printf "********************\n"
            printf "Archivo $i\n"
            printf "********************\n"

            while read linea
            do
                echo $linea
            done < $i
        else
            printf "\n\n********************\n"
            printf "Archivo $i\n"
            printf "********************\n"
            printf "\nERROR: Este archivo NO es un archivo regular\n\n\n"
        fi
    done
else
    while read linea
    do
        echo $linea
    done
fi
```

EJERCICIO 4

Centra en la pantalla las líneas de un archivo de texto que se pasa como argumento. Si no se pasa argumento, entonces ordena las líneas que se introducen por la entrada estándar.

```
#!/bin/bash



	es_archivo=`test -f $1 && echo true || echo false`;
	if [ $es_archivo == "true" ]
	then
		columnas=`tput cols`
		while read linea
		do
			ancho_linea=`echo $linea|wc -c`
			espacios=$[ $[ $columnas - $ancho_linea ] / 2
			] printf "%${espacios}s %s\n" "" "$linea"
		done < $1
	else
		echo "No es un archivo regular"
if [ $# -eq 1 ]
then









	fi
else
	while read linea
	do
		echo $linea >> temporal
	done

	echo "----------Después de ser ordenado----------"
	cat temporal|sort >> temporal2

	while read linea
	do
		echo $linea
	done < temporal2

	rm temporal
	rm temporal2
fi
```

Sesion 8

EJERCICIO 1

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

Sale éste error:

```
main.cpp: In function ‘int main()’:
main.cpp:7:17: error: ‘print_hello’ was not declared in this scope
main.cpp:9:52: error: ‘factorial’ was not declared in this scope
```

Indica que no sabe qúe hacen esas funciones dado que no se ha incluido ninguna librería que las contenga, porque hemos comentado la línea.

EJERCICIO 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?

Porque no hemos incluido los objetos .o de las funciones del seno, coseno y tangente, los cuales también se pueden incluir mediante una biblioteca, pero no la hemos incluido.

Hemos podido generar sin problemas el main.o porque tan sólo llama a las demás funciones.

EJERCICIO 3

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

Ha fallado porque hemos movido las librerías a otra carpeta, y la opción -L./ especifica que debe buscar las bibliotecas en la carpeta actual, y como ahí no están no encuentra la definición de ninguna función.

EJERCICIO 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?

Si no ha sido modificado ningún archivo, no se realiza ninguna operación.

Si se ha modificado alguno, se vuelve a generar el archivo objeto, o sea recompilación, de ese archivo cpp.

Se debe a que la orden make comprueba previamente si los archivos han sufrido modificaciones, o sea comprueba dependencias.

EJERCICIO 5

Usando el ejemplo anterior de archivo makefile, sustituya la línea de orden de la regla cuyo objetivo es programa2 por una línea en la que se use alguna de las variables especiales y que proporcione un resultado equivalente.

```
# Nombre archivo: makefile
# Uso: make
# 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 las opciones que se van a pasar al
compilador CPPFLAGS= -Wall

# 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: 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 6

Busque la variable predefinida de make que almacena la utilidad del sistema que permite construir bibliotecas. Recuerde que la orden es ar. (use la orden grep). ¡No vaya a leer línea a línea toda la salida! A continuación, sustituya en el archivo makefile empleado en el ejercicio anterior la orden ar por su variable correspondiente.

La variable predefinida es AR.

```
# Nombre archivo: makefile
# Uso: make
# 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 las opciones que se van a pasar al
compilador CPPFLAGS= -Wall

# 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: 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

Explique las dependencias que existen y cada una de las líneas del siguiente archivo makefile. Enumere las órdenes que se van a ejecutar como consecuencia de ejecutar la utilidad make sobre él.

```
# Nombre archivo: makefile2
# Uso: make -f makefile2
# Descripción: Mantiene todas las dependencias entre los módulos que utiliza el
```

```
# programa1.

#Esta variable es el compilador a usar
CC=g++
#Variable que indica las opciones que se van a pasar al compilador, en este caso
con -Wall decimos que nos muestre las advertencias
CPPFLAGS=-Wall
#Esta es una variable de la shell, por si no tenemos el root y necesitamos cambiar
esta variable para el usuario en cuestion
LDFLAGS=
#Esta variable indica el listado de dependencias
SOURCE_MODULES=main.cpp factorial.cpp hello.cpp
#Esta variable hace referencia a los objetos .o que se crearan a partir de los
.cpp, en este caso los creamos con el mismo
nombre. OBJECT_MODULES=$(SOURCE_MODULES:.cpp=.o)
#Esta variable hace referencia al nombre que tomará el programa ejecutable
EXECUTABLE=programa1

#Aquí definimos una regla que hace referencia a todos los
objetos all: $(OBJECT_MODULES) $(EXECUTABLE)

#Aquí definimos las dependencias de programa1 y generamos el .o
correspondiente $(EXECUTABLE): $(OBJECT_MODULES)
      $(CC) $(LDFLAGS) $^ -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 8

Con la siguiente especificación de módulos escriba un archivo makefile que automatice el proceso de compilación del programa final.

El archivo programa.cpp usa funciones de ordenación de los elementos de un array incluidas en el archivo ordenacion.cpp. La declaración de estas funciones está en el archivo de cabeceras ordenacion.h. Además, programa.cpp utiliza funciones de manejo de arrays incluidas en array.cpp, cuyo archivo de cabecera es array.h. Y también utiliza funciones para imprimir los arrays, incluidas en impresion.cpp.

El archivo ordenacion.cpp utiliza funciones de manejo de arrays incluidas en el archivo array.cpp, cuya declaración se encuentra en array.h.

El archivo impresion.cpp se encarga de proporcionar utilidades de impresión generales para distintos tipos de datos, entre ellos la impresión de arrays. Las declaraciones de sus funciones se encuentran en utilities.h.

```
# Nombre archivo: makefile
```

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

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

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

# Variable que indica el directorio en donde se encuentran los archivos de
cabecera
INCLUDE_DIR= ./headers
SOURCE_MODULES=programa.cpp ordenacion.cpp array.cpp impresion.cpp
OBJECT_MODULES=$(SOURCE_MODULES:.cpp=.o)

#Variable con el nombre del ejecutable
EXECUTABLE=programa

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

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECT_MODULES)
	$(CC) $(CPPFLAGS) $^ -o programa #Si hubiera librerías irían aquí
ordenacion.o: ordenacion.cpp array.o
	$(CC) $(CPPFLAGS) -c $^
array.o: array.cpp
	$(CC) $(CPPFLAGS) -c $^
impresion.o: impresion.cpp
	$(CC) -I$(INCLUDE_DIR) $(CPPFLAGS)  -c $^
Clean:
	rm *.o $(EXECUTABLE)
```

Sesion 9

EJERCICIO 1

Compile los archivos main.cpp factorial.cpp hello.cpp y genere un ejecutable con el nombre ejemplo1. Lance gdb con dicho ejemplo y ejecútelo dentro del depurador. Describa la información que ofrece.

```
$ g++ -g main.cpp hello.cpp factorial.cpp -o ejemplo1

GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2 Copyright
(C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Para las instrucciones de informe de errores, vea:
<http://www.gnu.org/software/gdb/bugs/>...
Leyendo símbolos desde
/home/superjes/modulos/ejemplo1...hecho. (gdb)
```

EJERCICIO 2

Usando la orden list muestre el código del programa principal y el de la función factorial (para ello utilice la orden help list).

```
(gdb) l main //Muestra las lineas del programa principal
(gdb) l factorial //Muestra las lineas de la funcion factorial
```

EJERCICIO 3

Ponga varios puntos de ruptura en las zonas marcadas en el código fuente con /* break */. Muestre información de todas las variables que se estén usando en cada detención. Muestre la información del contador de programa y el de la pila (para ello utilice el valor del registro $sp en lugar del $cp).

```
$ g++ -g main.cpp -o ejemploprueba
$ gdb ejemploprueba

(gdb) break 14            //Creo un breakpoint en linea 14
Punto de interrupción 1 at 0x80485d3: file main.cpp, line 14.
(gdb) run
Starting program: /home/superjes/ejemploprueba

Breakpoint 1, cuenta (y=0) at main.cpp:17
17        return tmp;
(gdb) info locals  //Muestro el valor de las variables locales
tmp = 2
(gdb) p/x $pc      //Informacion del contador de programa
$2 = 0x80485d3
(gdb) p/x $ps      //Informacion del contador de pila
$3 = 0x200202



(gdb) break 30            //Creo un breakpoint en linea 30
Punto de interrupción 1 at 0x80485ee: file main.cpp, line 30.
(gdb) run
Starting program: /home/superjes/ejemploprueba

Breakpoint 1, multiplica (x=3, y=2) at main.cpp:31
31          final = final + y;
(gdb) info locals
final = 0
i = 0
(gdb) p/x $pc
$3 = 0x80485ee
(gdb) p/x $ps
$4 = 0x200202


(gdb) break 43            //Creo un breakpoint en linea 30
Punto de interrupción 1 at 0x8048613: file main.cpp, line 43.
(gdb) run
Starting program: /home/superjes/ejemploprueba

Breakpoint 1, main () at main.cpp:44
44 final1 = multiplica(3, 2); (gdb)
info locals
final1 = 134514411
final2 = 1174096
i = 3903476
(gdb) p/x $pc
$4 = 0x8048613
(gdb) p/x $ps
$5 = 0x200282

(gdb) break 47
Punto de interrupción 2 at 0x804862b: file main.cpp, line
47. (gdb) continue
Continuando.
```

```
Breakpoint 2, main () at main.cpp:47
47 for (i = 0; i < 100; i ++) (gdb)
info locals
final1 = 6
final2 = 1174096
i = 3903476
```

EJERCICIO 4

Imprima el valor de las variables final1 y final2 del programa anterior en los puntos de ruptura correspondientes. Copie el contenido del ensamblador de la zona depurada.

```
(gdb) break 43            //Creo un breakpoint en linea 30
Punto de interrupción 1 at 0x8048613: file main.cpp, line 43.
(gdb) run
Starting program: /home/superjes/ejemploprueba

Breakpoint 1, main () at main.cpp:44
44        final1 = multiplica(3, 2);

(gdb) print final1
$1 = 134514411
(gdb) print final2
$2 = 1174096
(gdb) disassemble
Dump of assembler code for function main():
   0x0804860a <+0>: push  %ebp
   0x0804860b <+1>: mov   %esp,%ebp
   0x0804860d <+3>: and   $0xfffffff0,%esp
   0x08048610 <+6>: sub   $0x20,%esp
=> 0x08048613 <+9>: movl  $0x2,0x4(%esp)
   0x0804861b <+17>:      movl   $0x3,(%esp)
   0x08048622 <+24>:      call   0x80485d8 <multiplica(int, int)>
   0x08048627 <+29>:      mov    %eax,0x18(%esp)
   0x0804862b <+33>:      movl   $0x0,0x1c(%esp)
   0x08048633 <+41>:      jmp    0x804864a <main()+64>
   0x08048635 <+43>:      mov    0x1c(%esp),%eax
   0x08048639 <+47>:      mov    %eax,(%esp)
   0x0804863c <+50>:      call   0x80485c4 <cuenta(int)>
   0x08048641 <+55>:      mov    %eax,0x14(%esp)
   0x08048645 <+59>:      addl   $0x1,0x1c(%esp)
   0x0804864a <+64>:      cmpl   $0x63,0x1c(%esp)
   0x0804864f <+69>:      setle  %al
   0x08048652 <+72>:      test   %al,%al
   0x08048654 <+74>:      jne    0x8048635 <main()+43>
   0x08048656 <+76>:      mov    0x18(%esp),%eax
   0x0804865a <+80>:      mov    %eax,0x4(%esp)
   0x0804865e <+84>:      movl   $0x804a040,(%esp)
---Type <return> to continue, or q <return> to quit---
   0x08048665 <+91>:      call   0x8048498 <_ZNSolsEi@plt>
   0x0804866a <+96>:      movl   $0x80487a0,0x4(%esp)
   0x08048672 <+104>:     mov    %eax,(%esp)
```

```
   0x08048675 <+107>: call 0x80484f8
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
   0x0804867a <+112>: mov $0x0,%eax
   0x0804867f <+117>: leave
   0x08048680 <+118>: ret
End of assembler dump.



(gdb) continue
Continuando.

Breakpoint 2, main () at main.cpp:47
47        for (i = 0; i < 100; i ++)
(gdb) print final1
$3 = 6
(gdb) print final2
$4 = 1174096
(gdb) disassemble
Dump of assembler code for function main():
   0x0804860a <+0>: push  %ebp
   0x0804860b <+1>: mov   %esp,%ebp
   0x0804860d <+3>: and   $0xfffffff0,%esp
   0x08048610 <+6>: sub   $0x20,%esp
   0x08048613 <+9>: movl  $0x2,0x4(%esp)
   0x0804861b <+17>:      movl   $0x3,(%esp)
   0x08048622 <+24>:      call   0x80485d8 <multiplica(int, int)>
   0x08048627 <+29>:      mov    %eax,0x18(%esp)
=> 0x0804862b <+33>:      movl   $0x0,0x1c(%esp)
   0x08048633 <+41>:      jmp    0x804864a <main()+64>
   0x08048635 <+43>:      mov    0x1c(%esp),%eax
   0x08048639 <+47>:      mov    %eax,(%esp)
   0x0804863c <+50>:      call   0x80485c4 <cuenta(int)>
   0x08048641 <+55>:      mov    %eax,0x14(%esp)
   0x08048645 <+59>:      addl   $0x1,0x1c(%esp)
   0x0804864a <+64>:      cmpl   $0x63,0x1c(%esp)
   0x0804864f <+69>:      setle  %al
   0x08048652 <+72>:      test   %al,%al
   0x08048654 <+74>:      jne    0x8048635 <main()+43>
   0x08048656 <+76>:      mov    0x18(%esp),%eax
   0x0804865a <+80>:      mov    %eax,0x4(%esp)
   0x0804865e <+84>:      movl   $0x804a040,(%esp)
---Type <return> to continue, or q <return> to quit---
   0x08048665 <+91>:      call   0x8048498 <_ZNSolsEi@plt>
   0x0804866a <+96>:      movl   $0x80487a0,0x4(%esp)
   0x08048672 <+104>:     mov    %eax,(%esp)
   0x08048675 <+107>:     call   0x80484f8
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
   0x0804867a <+112>:     mov    $0x0,%eax
   0x0804867f <+117>:     leave
   0x08048680 <+118>:     ret
End of assembler dump.
```

EJERCICIO 5

Elimine todos los puntos de ruptura salvo el primero.

```
(gdb) delete breakpoint 2 3 4
```

EJERCICIO 6

Realice las acciones del ejercicio 3 y las del ejercicio 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.mdb*/
break 14
run
info locals
p/x $pc p/x
$ps break
30
n
info locals
p/x $pc p/x
$ps break
43
n
info locals
p/x $pc p/x
$ps break
47
n
info locals
p/x $pc p/x
$ps
delete breakpoint 2 3 4


gdb -x guion.gdb ejemplo1
```

La variable final2 vale 100 veces la función cuenta(), la cual en cada iteración suma 2 a la variable.

EJERCICIO 7

Realice la depuración del programa ejecutable obtenido a partir del archivo fuente ej1.cpp. Averigüe qué sucede y por qué no funciona. Intente arreglar el programa.

```
#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 */

float suma (float x, float y) //Han de ser float para que no se pierda
la precisión

{

   float tmp;



   tmp = x + y;



   return tmp;

}



/* Realiza la sumatoria de un vector */

int sumatoria (float vector[], int n)

{

   int i;

   float tmp; //Ha de ser float para que no se pierda la precisión



   tmp = 0;

   for (i = 0; i < n; i ++)

      tmp = suma(tmp, vector[i]); //faltaba guardar en la variable tmp el valor
devuelto de la función suma
```

```
   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 segundo parametro debe ser como máximo el
tamaño del vector


   return 0;

}
```

Sesion 10

EJERCICIO 1

Compile el programa main.cpp y genere un ejecutable con el nombre ejemplo1. Ejecute gdb con dicho ejemplo y realice una ejecución depurada mediante la orden run. Añada un punto de ruptura (breakpoint) en la función cuenta (tal y como se muestra en el ejemplo anterior). Realice 10 pasos de ejecución con step y otros 10 con next. Comente las diferencias.

```
(gdb) run
Starting program: /home/superjes/ejemplo1
6

Program exited normally.
(gdb) break cuenta
Punto de interrupción 1 at 0x80485ca: file main.cpp, line
13. (gdb) run
Starting program: /home/superjes/ejemplo1

Breakpoint 1, cuenta (y=0) at main.cpp:13
13 tmp = y + 2;
(gdb) step
17        return
tmp; (gdb) step
18    }
(gdb) step
main () at main.cpp:47
47 for (i = 0; i < 100; i ++) (gdb)
step
51 final2 = cuenta(i); (gdb)
step

Breakpoint 1, cuenta (y=1) at main.cpp:13
13 tmp = y + 2;
(gdb) step
17        return
tmp; (gdb) step
18    }
(gdb) step
main () at main.cpp:47
```

```
47 for (i = 0; i < 100; i ++) (gdb)
step
51 final2 = cuenta(i); (gdb)
step

Breakpoint 1, cuenta (y=2) at main.cpp:13
13 tmp = y + 2;
(gdb) next
17        return
tmp; (gdb) next
18     }
(gdb) next
main () at main.cpp:47
47 for (i = 0; i < 100; i ++) (gdb)
next
51 final2 = cuenta(i); (gdb)
next

Breakpoint 1, cuenta (y=3) at main.cpp:13
13 tmp = y + 2;
(gdb) next
17        return
tmp; (gdb) next
18     }
(gdb) next
main () at main.cpp:47
47 for (i = 0; i < 100; i ++) (gdb)
next
51 final2 = cuenta(i); (gdb)
next

Breakpoint 1, cuenta (y=4) at main.cpp:13
13 tmp = y + 2;
(gdb)
```

==== next (n) ====

Ejecuta la siguiente línea de programa. Si es una función, ejecuta la función entera (no entra en ella).

==== step (s) ====

Como next, pero si es una función entra en ella.

EJERCICIO 2

Depure el programa del ejercicio 1. Introduzca un punto de ruptura (breakpoint) dentro de la función cuenta. Muestre la información del marco actual y del marco superior, vuelva al marco inicial y compruebe si ha cambiado algo.

```
g++ -g main.cpp -o ejemplo1
```

```
bdg ejemplo1

(gdb) break cuenta
      Punto de interrupción 1 at 0x80485ca: file main.cpp, line 13.
(gdb) run
      Starting program: /home/javi/Escritorio/Sesion9y10/s10/ejemplo1

      Breakpoint 1, cuenta (y=0) at
      main.cpp:13 13 tmp = y + 2;
(gdb) info frame
      Stack level 0, frame at 0xbffff340:
       eip = 0x80485ca in cuenta (main.cpp:13); saved eip 0x8048641
       called by frame at 0xbffff370
       source language c++.
       Arglist at 0xbffff338, args: y=0
       Locals at 0xbffff338, Previous frame's sp is
       0xbffff340 Saved registers:
        ebp at 0xbffff338, eip at 0xbffff33c
(gdb) up
      #1 0x08048641 in main () at main.cpp:48
      48 final2 = cuenta(i);
(gdb) info frame
      Stack level 1, frame at 0xbffff370:
       eip = 0x8048641 in main (main.cpp:48); saved eip
       0x272e37 caller of frame at 0xbffff340
       source language c++. Arglist
       at 0xbffff368, args:
       Locals at 0xbffff368, Previous frame's sp is
       0xbffff370 Saved registers:
        ebp at 0xbffff368, eip at
0xbffff36c (gdb) down
      #0 cuenta (y=0) at main.cpp:13
      13 tmp = y + 2;
(gdb) info frame
      Stack level 0, frame at 0xbffff340:
       eip = 0x80485ca in cuenta (main.cpp:13); saved eip 0x8048641
       called by frame at 0xbffff370
       source language c++.
       Arglist at 0xbffff338, args: y=0
       Locals at 0xbffff338, Previous frame's sp is
       0xbffff340 Saved registers:
        ebp at 0xbffff338, eip at 0xbffff33c
```

No, no ha cambiado nada referente a su información.

EJERCICIO 3

Ponga un punto de ruptura en la línea 30 del programa (función multiplica) de tal forma que el programa se detenga cuando la variable final llegue a 8. Compruebe si se detiene o no y explique por qué.

```
(gbd) break 30 if final == 8
```

No se para nunca, ya que final no alcanza el valor de 8

EJERCICIO 4

Pruebe el ejemplo y ejecútelo después un continue, imprima el valor de la variable tmp. Explique por qué el valor no es 12.

La variable "tmp" es variable interna de la función cuenta(int y) y su valor es la asignación del valor de la varible "y" más dos, por lo que aunque cambiemos el valor de "tmp", en la siguiente llamada a la función se le asignará otro valor independiente al anterior.

EJERCICIO 5

Compile el programa ej2.cpp como ej2 añadiendo el flag de depuración. Ejecútelo en una shell. Mientras, en otra shell ejecute el depurador con el programa ej2 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. Escriba todo los pasos que haya realizado.

```
1. g++ -g ej1.cpp -o ej2
2. ./ej2
3.(En otro terminal) top   //Busco el PID correpondiente a ej2, en este caso 33252
4. gdb
5. attach 33252           //Así queda parado el programa, si salimos de la
depuración continuará ejecutándose
6. kill                   //matamos el proceso y se finaliza
```

EJERCICIO 6

Utilizando las herramientas de depuración de GDB, corrija todos los errores del programa anterior. Escriba todos los pasos que haya realizado.

(No hay programa)