



# **Introducción al entorno de desarrollo**

Sistemas Operativos  
Práctica 1

# Índice



- ▶ Introducción a la máquina virtual
- ▶ Entorno de desarrollo C para GNU/Linux
- ▶ Práctica 1
- ▶ Extras
  - Páginas de manual
  - Shell
  - Introducción a Eclipse

## **Bibliografía:**

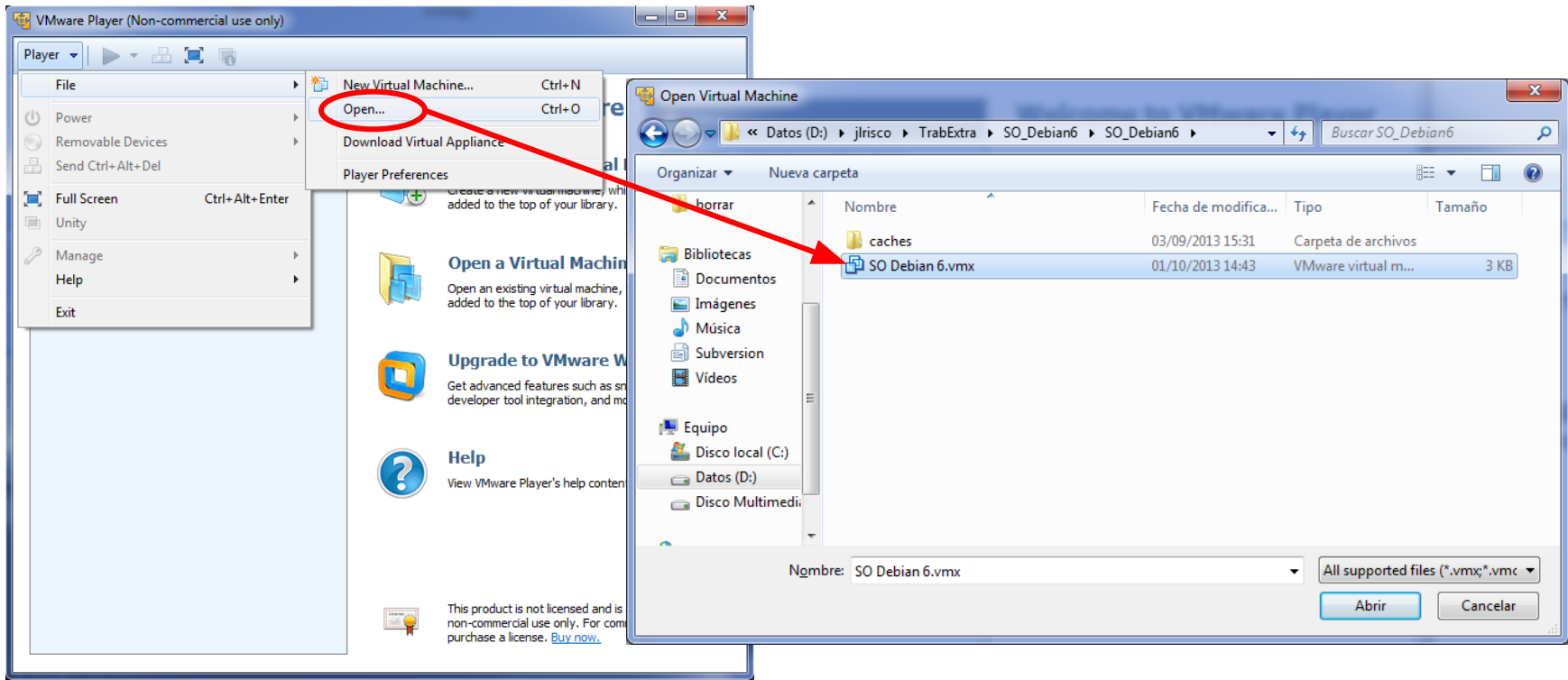
[The One Page Linux Manual](#)

[Entorno de desarrollo C para GNU/Linux, Christian Tenllado, 2014](#)

# Imagen VMWare



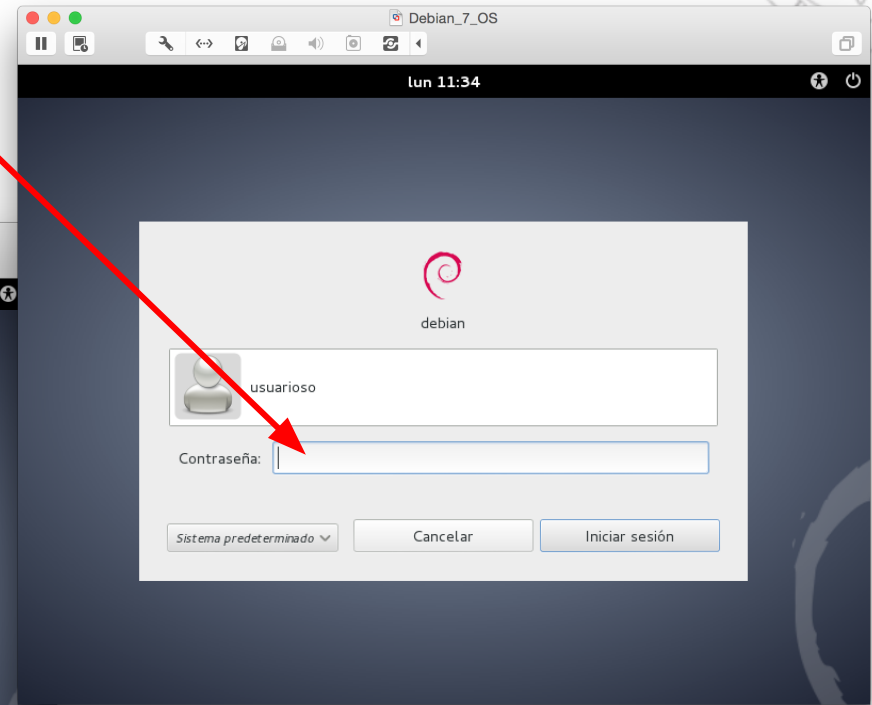
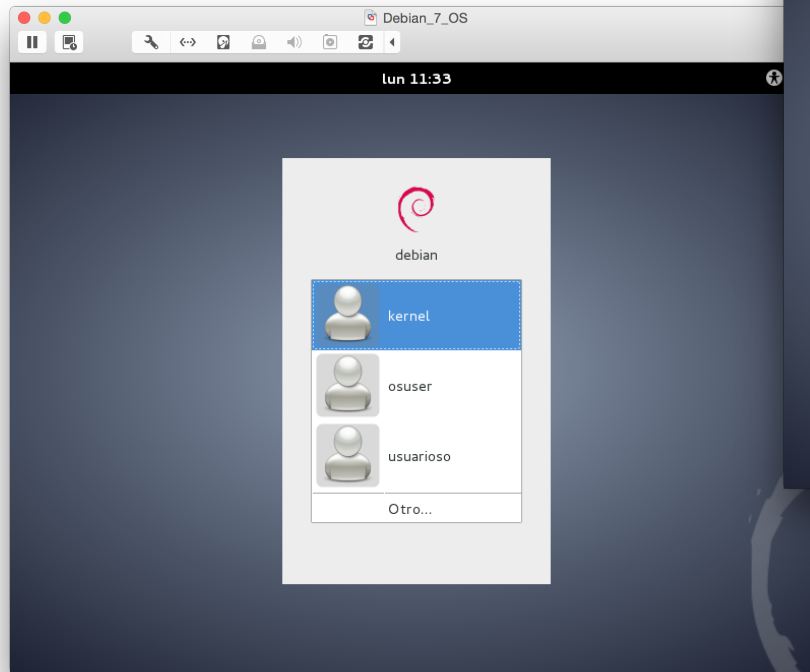
- Se proporciona una imagen de máquina virtual VMWare
  - ✓ Instalar [VMWare player](#) (Google it)
  - ✓ Descargar [Debian OS](#) (Curso 2015/16 es Debian 7)
    - Descomprimir (en Windows usando el programa [WinRAR](#))
- Ejecutar VMWare player y seleccionar la MV descomprimida



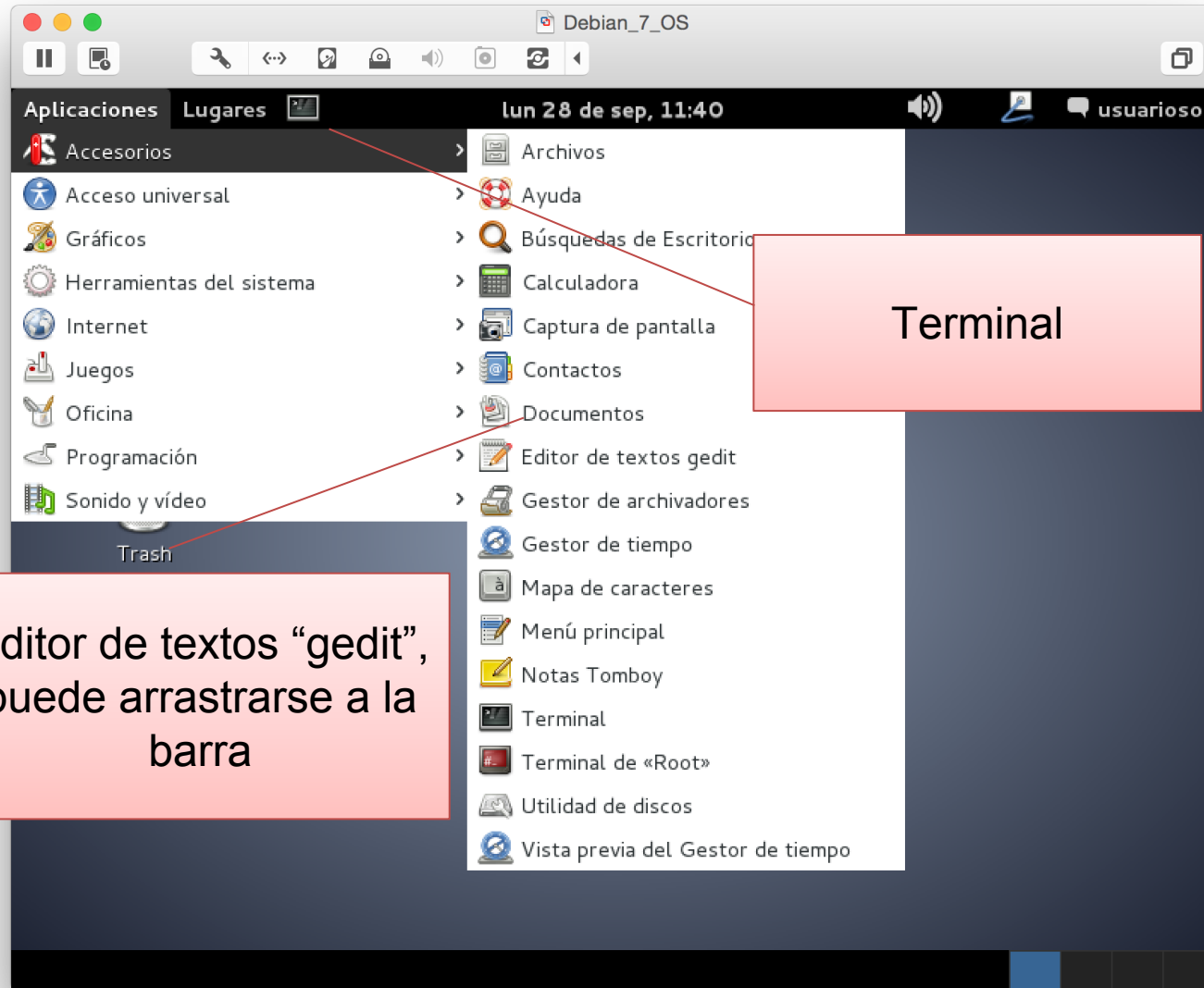
# Usuarios y contraseñas

## ■ Usuarios y contraseñas

- usuarios, usuario
- root: sudo su -



# Algunas utilidades



Editor de textos “gedit”,  
puede arrastrarse a la  
barra

Terminal



# Introducción

- ▶ Antes de nada, hay que saber [programar en C](#)
- ▶ Herramientas necesarias:
  - ▷ Editor de texto: **gedit**
  - ▷ Compilador: **gcc**
  - ▷ Automatización de proyectos: **make**
  - ▷ Depurador: **gdb**

# gcc (apropos gcc, man file)



Sea el siguiente programa:

```
// saludo.c

#include <stdio.h>
int main(void) {
    char nombre[100];
    printf("Escribe tu nombre: ");
    if (scanf(" %s", nombre) != 1) {
        printf("Error o Fin\n");
        return 1;
    } else {
        printf("Hola %s\n", nombre);
        return 0;
    }
}
```

```
$ gedit saludo.c &
$ gcc -c saludo.c
$ file saludo.o
saludo.o: ELF 32-bit LSB
relocatable, Intel 80386, version 1
(SYSV), not stripped
$ gcc saludo.o -o saludo
$ file saludo
saludo: ELF 32-bit LSB executable,
Intel 80386, version 1 (SYSV), for
GNU/Linux 2.6.8, dynamically linked
(uses shared libs), not stripped
$ ./saludo
Escribe tu nombre: Fulano
Hola Fulano
```

# make (man rm, man make)



## # Makefile

```
all: saludo.o
    gcc saludo.o -o saludo

saludo.o: saludo.c
    gcc -Wall -g -c saludo.c

clean:
    rm -f saludo *.o
```

```
$ make clean
rm *.o
$ make
gcc -Wall -g -c saludo.c
gcc saludo.o -o saludo
$ ./saludo
Escribe tu nombre: Fulano
Hola Fulano
```



# make



```
// archi.c
```

```
#include <stdio.h>
#include "archi.h"

int main(void) {
    printf("Hola ...%d\n", VAR);
    return 0;
}
```

```
// archi.h
```

```
#define VAR 100
```

```
# Makefile
```

```
all: archi.o
    gcc archi.o -o archi

archi.o: archi.c
    gcc -Wall -g -c archi.c

clean:
    rm -f archi *.o
```

```
$ make clean
```

```
rm *.o
```

```
$ make
```

```
gcc -Wall -g -c saludo.c
```

```
gcc saludo.o -o saludo
```

```
$ ./saludo
```

```
Escribe tu nombre: Fulano
```

```
Hola Fulano
```

# make



**// main.c**

```
#include <stdio.h>
#include "suma.h"
#include "prod.h"

int main(void) {
    int a = 5, b = 7;
    printf("%d + %d = %d\n", a, b, suma(a,b));
    printf("%d * %d = %d\n", a, b, prod(a,b));
    return 0;
}
```

**// suma.h**

```
int suma(int s1, int s2);
```

**// prod.h**

```
int prod(int p1, int p2);
```

**// suma.c**

```
#include "suma.h"

int suma(int s1, int s2) {
    return (s1+s2);
}
```

**// prod.c**

```
#include "prod.h"

int prod(int p1, int p2) {
    return (p1*p2);
}
```

**# Makefile**

```
all: main.o suma.o prod.o
    gcc main.o suma.o prod.o -o main

main.o: main.c
    gcc -Wall -g -c main.c

suma.o: suma.c
    gcc -Wall -g -c suma.c

...
```

# GDB



- ▶ La utilidad de depuración en ejecución de programas C en LINUX es, por antonomasia, el depurador gdb de GNU

```
// badsort.c
```

```
/* 1 */ #include <stdio.h>
/* 2 */ typedef struct {
/* 3 */     char data[4096];
/* 4 */     int key;
/* 5 */ } item;
/* 6 */
/* 7 */ item array[] = {
/* 8 */     {"bill", 3},
/* 9 */     {"neil", 4},
/* 10 */     {"john", 2},
/* 11 */     {"rick", 5},
/* 12 */     {"alex", 1},
/* 13 */ };
/* 14 */
```



# badsort.c

- La utilidad de depuración en ejecución de programas C en LINUX es, por antonomasia, el depurador gdb de GNU

// badsort.c

```
/* 15 */ void sort(item *a, int n) {
/* 16 */
/* 17 */     int i = 0, j = 0;
/* 18 */     int s = 1;
/* 19 */
/* 20 */     for(; i < n & s != 0; i++) {
/* 21 */         s = 0;
/* 22 */         for(j = 0; j < n; j++) {
/* 23 */             if(a[j].key > a[j+1].key) {
/* 24 */                 item t = a[j];
/* 25 */                 a[j] = a[j+1];
/* 26 */                 a[j+1] = t;
/* 27 */                 s++;
/* 28 */             }
/* 29 */         }
/* 30 */         n--;
/* 31 */     }
/* 32 */ }
```



# badsort.c

- La utilidad de depuración en ejecución de programas C en LINUX es, por antonomasia, el depurador gdb de GNU

```
// badsort.c

/* 33 */
/* 34 */ int main()
/* 35 */ {
/* 36 */     int i;
/* 37 */     sort(array,5);
/* 38 */     for(i = 0; i < 5; i++)
/* 39 */         printf("array[%d] = {%s, %d}\n",
/* 40 */             i, array[i].data, array[i].key);
/* 41 */     return 0;
/* 42 */ }
```

# Inicialización de gdb



```
$ gcc -Wall -g badsort.c -o badsort
badsort.c: In function 'sort':
badsort.c:20:17: warning: suggest parentheses around comparison in operand of '&'
[-Wparentheses]
```

```
/* 20 */ for(; i <n & s != 0; i++) {
```

```
/* 20 */ for(; (i <n) && (s != 0); i++) {
```



```
$ gcc -Wall -g badsort.c -o badsort
$ ./badsort
Violación de segmento
```

```
$ gdb ./badsort
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 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.
...
Reading symbols from /home/.../badsort...done.
(gdb)
```

# Comandos básicos



```
(gdb) run
```

```
Starting program: /home/.../badsort
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x0000000000400595 in sort (a=0x600c20, n=5) at badsort.c:23
```

```
23  /* 23 */                                if(a[j].key > a[j+1].key) {
```

```
(gdb)
```

```
(gdb) bt
```

```
#0 0x0000000000400595 in sort (a=0x600c20, n=5) at badsort.c:23
```

```
#1 0x000000000040078a in main () at badsort.c:37
```

```
(gdb) print j
```

```
$1 = 4
```

```
(gdb) info locals
```

```
i = 0
```

```
j = 4
```

```
s = 2
```

```
(gdb) list
```

```
22  /* 22 */
```

```
23  /* 23 */
```

```
24  /* 24 */
```

```
25  /* 25 */
```

```
26  /* 26 */
```

```
(gdb) quit
```

```
/*22*/ for(j = 0; j < n; j++) {
```

```
/*22*/ for(j = 0; j < (n-1); j++) {
```

```
for(j = 0; j < n; j++) {  
    if(a[j].key > a[j+1].key) {  
        item t = a[j];  
        a[j] = a[j+1];  
        a[j+1] = t;
```



# Depurando línea a línea



```
$ gcc -Wall -g badsort.c -o badsort
$ ./badsort
array[0] = {john, 2}
array[1] = {alex, 1}
array[2] = {bill, 3}
array[3] = {neil, 4}
array[4] = {rick, 5}
```

```
(gdb) break badsort.c:sort
```

```
Breakpoint 1 at 0x400524: file badsort.c, line 17.
```

```
(gdb) run
```

```
Starting program: /home/.../badsort
```

```
Breakpoint 1, sort (a=0x600c20, n=5) at badsort.c:17
```

```
17  /* 17 */      int i = 0, j = 0;
```

```
(gdb) next
```

```
18  /* 18 */      int s = 1;
```

```
(gdb) print array[0]@5
```

```
$1 = {{data = "bill", '\000' <repeats 4091 times>, key = 3}, {
      data = "neil", '\000' <repeats 4091 times>, key = 4}, {
      data = "john", '\000' <repeats 4091 times>, key = 2}, {
      data = "rick", '\000' <repeats 4091 times>, key = 5}, {
      data = "alex", '\000' <repeats 4091 times>, key = 1}}
```

```
(gdb)
```



# Encontrando el error



```
$ gcc -Wall -g badsort.c -o badsort
$ ./badsort
array[0] = {john, 2}
array[1] = {alex, 1}
...
```

```
(gdb) print array[0]@5
$22 = {{data = "john", '\000' <repeats 4091 times>, key = 2}, {
      data = "alex", '\000' <repeats 4091 times>, key = 1}, {
      data = "bill", '\000' <repeats 4091 times>, key = 3}, {
```

```
...
(gdb) print j
$23 = 1
```

```
...
(gdb) n
30 /* 30 */ n--;
```

`/* 30 */ n--;`

```
(gdb) n
20 /* 20 */ for(; (i < n) && (s != 0); i++) {
```

```
...
(gdb) print n
$25 = 2
```

```
...
(gdb) print i
$26 = 3
```

```
(gdb)
```



# Algunos comandos adicionales

```
$ gcc -Wall -g badsort.c -o badsort
$ ./badsort
array[1] = {alex, 1}
array[0] = {john, 2}
array[2] = {bill, 3}
array[3] = {neil, 4}
array[4] = {rick, 5}
```

```
# Algunos comandos más de gdb:
# Añadir/Listado/Deshabilitar breakpoints
(gdb) break [file.c:]<function_name|line_number>
(gdb) info breakpoints
(gdb) disable break #Num_Breakpoint
# Añadir/Listado/Deshabilitar visualización de variables
(gdb) display <var_name>
(gdb) info display
(gdb) disable display #Num_Display
```

# Práctica 1: Programa “mtar”



- El guión de la práctica puede encontrarse en el CV

**Archivo mtar:** fichero binario que alberga múltiples ficheros en su interior

Número de ficheros (N)
Ruta fichero 1
Tamaño fichero 1
Ruta fichero 2
Tamaño fichero 2
...
Ruta fichero N
Tamaño fichero N
Datos fichero 1
Datos fichero 2
...
Datos fichero N



# Implementación

## ► Modo de uso

```
$ ./mitar -c|x -f archivo_mtar [fich1 fich2 ...]
```

- -c : Crear archivo mtar
  - Ejemplo: `$ ./mitar -c -f ejemplo.mtar a.txt b.txt`
- -x : Extraer archivo mtar
  - Ejemplo: `$ ./mitar -x -f ejemplo.mtar`

## ► El proyecto consta de los siguientes ficheros

- ▷ `makefile`
- ▷ `mitar.c` : función `main()` del programa
- ▷ `mitar.h` : declaraciones de tipos de datos y funciones
- ▷ `rut_mitar.c` : funciones de creación y extracción de ficheros mtar
  - ▶ **Único fichero a modificar**

# Implementación



```
// mitar.h

#ifndef _MITAR_H
#define _MITAR_H

#include <limits.h>

typedef enum{
    NONE,
    ERROR,
    CREATE,
    EXTRACT
} flags;

typedef struct {
    char *name;
    unsigned int size;
} stHeaderEntry;

int createTar(int nFiles, char *fileNames[], char tarName[]);
int extractTar(char tarName[]);

#endif /* _MITAR_H */
```



# Funciones a implementar

```
int createTar(int nFiles, char *fileNames[], char* tarName);
/* Crea un fichero mtar con nombre tarName incluyendo en él los ficheros cuya rutas
están especificadas en el array fileNames */

int extractTar(char* tarName);
/* Extrae el fichero mtar cuya ruta se pasa como parámetro */

int copynFile(FILE *origen, FILE *destino, int nBytes);
/* Transfiere nBytes del fichero origen al fichero destino */
/* La copia de datos finalizará cuando se transfieran nBytes o se llegue al fin del
fichero origen */
/* copynFile() devuelve el número de bytes que se han transferido realmente */

int loadstr(FILE *file, char **buf);
/* Carga una cadena de caracteres terminada en '\0' del fichero, y devuelve
* en buf la dirección de memoria del heap donde fue copiado*/

int readHeader(FILE *tarFile, stHeaderEntry **header, int *nFiles);
/* Lee la cabecera del fichero mtar tarFile y copia la metainformación en el array
header */
/* La función ha de reservar memoria para el array header (de ahí el doble puntero
→ puntero por referencia) */
/* Devuelve en nFiles (entero por referencia) el número de ficheros contenidos en
el mtar */
```

# Extracción del fichero mtar



- ▶ En la extracción del fichero **mtar** tenemos un problema
  - ▷ No sabemos cuánto ocupa cada nombre de fichero, tenemos que leer hasta encontrar '\0'



# Estructura de readHeader

```
int readHeader(FILE *tarFile, stHeaderEntry **header, int *nFiles)
{
    int i,...;
    stHeaderEntry* p;

    ... Leemos el número de ficheros (N) del tarFile y lo copiamos en nFiles

    /* Reservamos memoria para el array */
    p = (stHeaderEntry *) malloc(sizeof (stHeaderEntry) * (*nFiles));

    for (i = 0; i < *nFiles; i++) {
        ... usamos loadstr para cargar el nombre el p[i].name (pasamos &p[i].name)
        ... comprobación y tratamiento de errores
        ... leemos el tamaño del fichero y lo almacenamos en p[i].size
    }

    return (EXIT_SUCCESS);
}
```





# Creación del fichero mtar

- ▶ En la creación del fichero **mtar** tenemos algunos problemas
  - ▷ No sabemos de antemano lo que va a ocupar la cabecera, depende de la suma de los tamaños de los nombre/rutas de los ficheros a introducir en el tar
  - ▷ No sabemos de antemano cuál es el tamaño en bytes de cada uno de los ficheros que hay que introducir en el **mtar**

# createTar: implementación simple



1. Abrimos el fichero mtar para escritura (fichero destino)
2. Reservamos memoria (con `malloc()`) para un array de `stHeaderEntry`
  - ▷ El array tendrá tantas posiciones como ficheros haya que introducir en el mtar
3. Copiamos la cabecera al fichero mtar
  - ▷ Copiamos nFiles
  - ▷ Por cada fichero (fileNames):
    - ▶ Copiamos el nombre terminado en '\0' al mtar
    - ▶ Calculamos el tamaño del fichero (fopen, fseek, fclose)
    - ▶ Copiamos el tamaño al mtar
4. Por cada fichero (fileNames):
  - ▷ Abrimos el fichero
  - ▷ Lo copiamos a disco con `copynFile`
  - ▷ Cerramos el fichero
5. Cerramos el fichero mtar

# Ejemplo: Creación de un fichero mtar



```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

# Ejemplo: Creación de un fichero mtar

```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Archivo test.mtar  
(en disco)

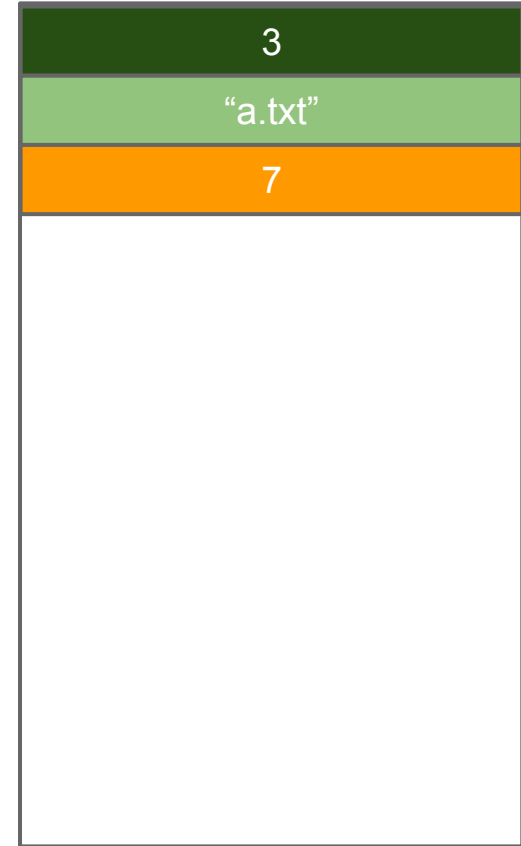
3

# Ejemplo: Creación de un fichero mtar



Archivo test.mtar  
(en disco)

```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

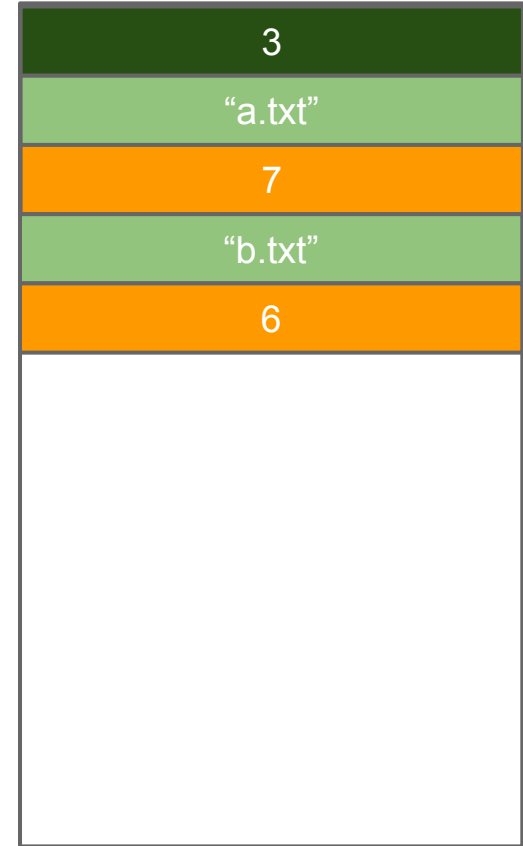


# Ejemplo: Creación de un fichero mtar



Archivo test.mtar  
(en disco)

```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

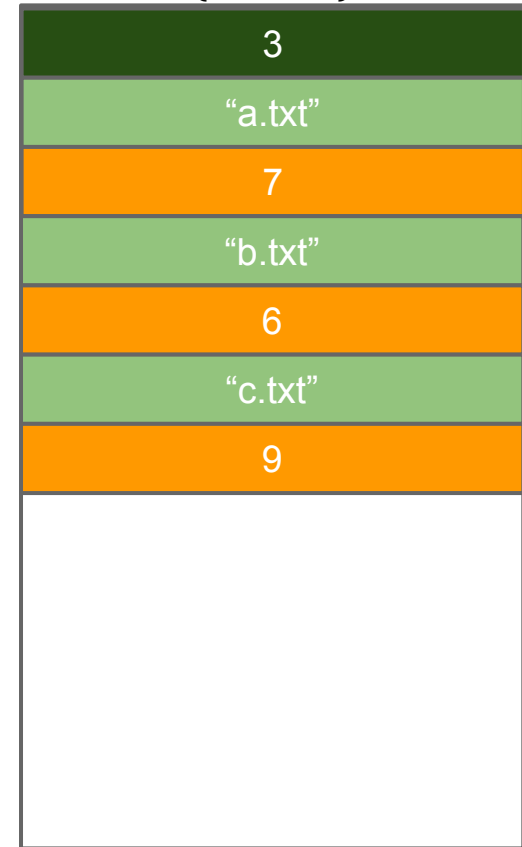


# Ejemplo: Creación de un fichero mtar



```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Archivo test.mtar  
(en disco)



# Ejemplo: Creación de un fichero mtar



```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Archivo test.mtar  
(en disco)





# Ejemplo: Creación de un fichero mtar



```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Archivo test.mtar  
(en disco)



# Ejemplo: Creación de un fichero mtar



```
$ ./mtar -c -f test.mtar a.txt b.txt c.txt
```

Archivo test.mtar  
(en disco)



# Ejemplo de ejecución



```
$ ls
a.txt b.txt c.txt makefile mitar.c mitar.h rut_mitar.c
$ du -b *.txt
7    a.txt
6    b.txt
9    c.txt
$ make
gcc -g -Wall -c mitar.c -o mitar.o
gcc -g -Wall -c rut_mitar.c -o rut_mitar.o
gcc -g -Wall -o mitar mitar.o rut_mitar.o
$ ./mitar -c -f test.mtar a.txt b.txt c.txt
Fichero mitar creado con exito
$ ls
a.txt c.txt mitar mitar.h rut_mitar.c test.mtar
b.txt makefile mitar.c mitar.o rut_mitar.o
```

# Ejemplo de ejecución



```
$ mkdir tmp
$ cd tmp/
$ ../mitar -x -f ../test.mtar
[0]: Creando fichero a.txt, tamaño 7 Bytes...Ok
[1]: Creando fichero b.txt, tamaño 6 Bytes...Ok
[2]: Creando fichero c.txt, tamaño 9 Bytes...Ok
$ ls
a.txt  b.txt  c.txt
$ diff a.txt ../a.txt
$ diff b.txt ../b.txt
$ diff c.txt ../c.txt
```



# Páginas de manual

- ▶ Muchos programas tienen una ayuda interna
  - ▷ `firefox --help`
- ▶ Si se necesitan más detalles, siempre se puede recurrir a las páginas de manual
  - ▷ `man man`

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda  man ls
LS(1)                                         LS(1)

NOMBRE
ls, dir, vdir - listan los contenidos de directorios

SINOPSIS
ls [opciones] [fichero...]
dir [fichero...]
vdir [fichero...]

Opciones de POSIX: [-CFRacdilqr tu l]

Opciones de GNU (en la forma más corta):
[-labcdfghiklmnopqrstuvwx ABCDFGHLNQRSUX] [-w cols] [-T cols] [-I
patrón] [--full-time] [--show-control-chars] [--block-size=tamaño]
[--format={long,verbose,commas,across,vertical,single-column}]
[--sort={none,time,size,extension}]
[--time={atime,access,use,ctime,status}] [--color[={none,auto,always}]]
[--help] [--version] [--]

DESCRIPCIÓN
El programa ls lista primero sus argumentos no directorios fichero, y
luego para cada argumento directorio todos los ficheros susceptibles de
Manual page ls(1) line 1 (press h for help or q to quit)
```



# Buscar en Man

- ▶ Una página de manual es una única página que contiene la documentación de referencia para el tópico consultado
- ▶ El comando `man` lee este documento en formato legible y lo muestra con el paginador por defecto, generalmente `less`
  - ▷ Teclas de dirección para movernos arriba/abajo
- ▶ Algunas página de manual son excesivamente largas
  - ▷ `man bash`
  - ▷ Se presiona “/” para buscar una palabra o “?” para buscar hacia atrás, seguido del término a buscar, a partir de entonces
    - ▶ Se presiona “n” para buscar la siguiente hacia adelante
    - ▶ Se presiona “N” para buscar la siguiente hacia atrás

# Secciones



## ► Las páginas de manual están divididas en secciones:

- (1) Comandos de usuario
- (2) Llamadas al sistema
- (3) Funciones en bibliotecas, especialmente la biblioteca estándar de C
- (4) Ficheros especiales (generalmente dispositivos, ubicados en /dev) y controladores (drivers)
- (5) Formatos de archivo y convenciones
- (6) Juegos y protectores de pantalla
- (7) Miscelánea
- (8) Comandos de administración del sistema y demonios
- (9) API del kernel (módulos y core kernel)

## ► Usuario normal: secciones 1,5 y 8

- ▷ A veces un mismo comando aparece documentado en varias secciones

```
$ man passwd  
$ man 1 passwd  
$ man 5 passwd
```



# Whatis y Apropos

- ▶ Man mantiene una base de datos sobre la que se puede buscar información
- ▶ Cada página tienen una sección llamada NAME, que incluye una breve sección del tópico
  - ▷ El comando **whatis** busca el término indicado en esta sección

```
$ whatis whatis
whatis (1)          - imprime descripciones de páginas de manual
```

- ▷ Para una búsqueda más genérica se usa **apropos**

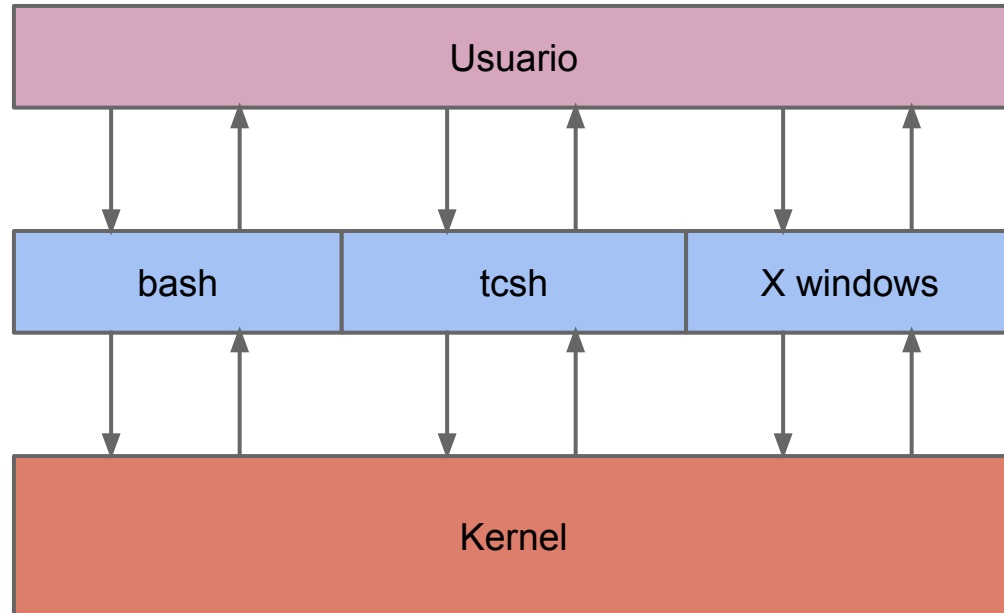
```
$ whatis png
png: nada apropiado.
$ apropos png
evince-thumbnailer (1) - create png thumbnails from PostScript and PDF documents
pdftocairo (1)        - Portable Document Format (PDF) to PNG/JPEG/PDF/PS/EPS/...
pdftohtml (1)         - program to convert PDF files into HTML, XML and PNG im...
pngtopnm (1)          - convert a Portable Network Graphics file into portable...
pnmtpng (1)           - convert a portable anymap into a Portable Network Grap...
xcursorgen (1)         - create an X cursor file from a collection of PNG images
```



# Shell



- Programa que actúa como interfaz entre el usuario y el SO





# Intérprete de comandos bash

- ▶ Bash (Bourne-again shell) es una shell Unix escrita por Brian Fox para el proyecto GNU como una alternativa libre a la shell Bourne
- ▶ Bash es el intérprete predeterminado en muchos sistemas UNIX: la mayoría de sistemas GNU/Linux, Solaris y Mac OS X
- ▶ También se ha portado a Microsoft Windows (proyecto Cygwin)
- ▶ Otros intérpretes:
  - ▷ **sh**: Es el shell Bourne
  - ▷ **tcsh** o TENEX C shell: Derivado de csh, es un shell C
  - ▷ **ksh** o Korn shell: en ocasiones usado por usuarios con experiencia en UNIX

# Ejecución bash



## ► Comportamiento del shell:

- ▷ Cuando un shell interactivo que no es un login shell arranca, Bash lee y ejecuta órdenes desde `~/.bashrc`, si existiese.
- ▷ Dispone de prefijos o “prompts” (PS1 y PS2).
- ▷ Los mandatos se leen en línea (readline) y se ejecutan tras su lectura.
- ▷ La historia de mandatos se guarda en fichero (HISTFILE) y es posible realizar búsquedas en el historial (CTRL+R)
- ▷ Se permite la expansión de alias.
- ▷ Se pueden modificar los manejadores de señal (Ctrl+C).
- ▷ Se puede controlar la acción a tomar cuando el intérprete de comandos recibe un carácter EOF (ignoreeof.)

# Ejecutando comandos

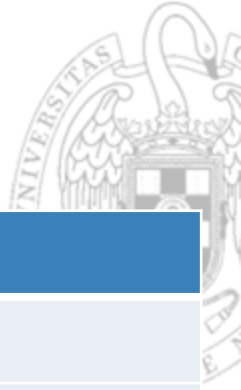


- ▶ Tipos de comandos
  - ▷ **Comandos internos** (built-in commands): Forman parte del repertorio del propio shell
  - ▷ **Comandos externos**: Programas externos al shell instalados en el sistema (ficheros binarios ejecutables o scripts)
- ▶ Las secuencias de comandos pueden incluirse en un fichero denominado **guión** o Script Bash
  - ▷ Cuando el programa es un guión, Bash creará un nuevo proceso usando `fork()`



# Comandos propios

- ▶ Bourne Shell built-ins ...
  - ▷ `:, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly, return, set, shift, test, [, times, trap, umask and unset.`
- ▶ + Bash built-in commands:
  - ▷ `alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit and unalias.`



# Comandos básicos

Comando	Descripción
<code>ls</code>	Lista los ficheros del directorio actual
<code>pwd</code>	Muestra en qué directorio nos encontramos
<code>cd directory</code>	Cambia el directorio
<code>man command</code>	Muestra la página de manual para el comando dado
<code>apropos string</code>	busca la cadena en la base de datos whatis
<code>file filename</code>	Muestra el tipo de fichero dado
<code>cat textfile</code>	Muestra el contenido del archivo en pantalla
<code>exit/logout</code>	Abandona la sesión
<code>grep</code>	Buscan en archivos líneas que contengan un patrón de búsqueda dado
<code>echo</code>	Muestra una línea de texto
<code>env</code>	Guarda información en el entorno
<code>export</code>	Cambia el valor de una variable de entorno



# Variables y operadores

## ► Variables

- ▷ `a=5` #asignación
- ▷ `echo $a` #expansión
- ▷ `b=$(( $a+3 ))` #aritmética entera
- ▷ `b=$(( $a<<1 ))` #operadores de bits

## ► Operadores aritméticos y de bits:

- ▷ `+ - / * % & | ^ << >>`



# Redirecciones

- ▶ Tres descriptores de ficheros predeterminados:
  - ▷ **stdin** (0) **stdout** (1) **stderr** (2)
- ▶ Redirección de la salida estándar:
  - ▷ orden > fichero
- ▶ Redirección de la salida de error:
  - ▷ orden 2> fichero
- ▶ Redirección de la entrada estándar:
  - ▷ orden < fichero





# Ejemplos

- ▶ Redirecciones:
  - ▷ `ls -l > listado`
  - ▷ `ls -l /etc >> listado`
  - ▷ `ls /bin/basha 2> error`
  - ▷ `find / -name 'lib*' -print > librerias 2>&1`

# Cauces, tuberías, pipes



- ▶ La salida estándar de una orden sirve como entrada estándar de otra:
  - ▷ `ls -l | more`
- ▶ Se combinan cauces y redirecciones:
  - ▷ `ps aux | grep -v root > ps.out`



# Listas de órdenes

- ▶ Variable **\$?**
  - ▷ status de la última orden ejecutada
- ▶ **orden1 ; orden2**
  - ▷ **orden2** se ejecuta cuando acaba **orden1**.
  - ▷ **\$?** es el status de **orden2**
- ▶ **orden1 && orden2**
  - ▷ **orden2** sólo se ejecuta si status de **orden1** == 0 (éxito)
- ▶ **orden1 || orden2**
  - ▷ **orden2** sólo se ejecuta si status de **orden1** != 0 (fallo)

# Ejecución en primer y segundo plano



## ► foreground y background

- ▷ En modo interactivo los procesos se ejecutan en primer plano (foreground): la shell no muestra el prompt hasta que no finaliza la ejecución de la última orden introducida.
- ▷ Si queremos dejar el proceso en segundo plano (background) se añade &:

```
$ xeyes &
```

```
[2] 7584
```

jobID      PID



# Comodines

- ▶ Permiten referirnos a un conjunto de ficheros con características comunes en sus nombres.
  - ▷ \* corresponde con cualquier conjunto de caracteres.
  - ▷ ? corresponde con cualquier carácter individual
  - ▷ **[conjunto]** corresponde con cualquier carácter dentro de conjunto.
- ▶ Ejemplo:
  - ▷ **?[a-c]\*.h** cualquier fichero cuyo nombre comience por un carácter cualquiera seguido de las letras **a**, **b** ó **c** y que acabe en **.h**



# Expansión de órdenes

- ▶ Podemos guardar en una variable la salida estándar de una orden o lista de órdenes.
- ▶ **Ejemplo:**
  - ▷ `num=$( ls a* | wc -w )`
- ▶ **Forma equivalente:**
  - ▷ `num=`ls a* | wc -w``



# Guiones Shell (1/2)

- ▶ Un guión shell es un fichero que contiene una secuencia de órdenes shell.
- ▶ Se crea un proceso shell que interpreta las líneas (subshell)
- ▶ Los comentarios comienzan por el carácter #
- ▶ Ejemplo:

```
#!/bin/bash  
mkdir tmp  
cd tmp  
touch hola  
cd ..
```

# Guiones Shell (2/2)



- ▶ Un guión es más versátil si su ejecución depende de parámetros.
- ▶ Los parámetros posicionales se denotan por \$1, \$2, \$3 ... \$9
- ▶ Pueden usarse como si fueran variables normales pero además:
  - ▷ \$# es el número total de parámetros.
  - ▷ shift desplaza a la izquierda los parámetros.



# Sentencias condicionales (1/2)



- ▶ Estructura **if-then-else**:

```
if condicion ; then
    bloque_then
else
    bloque_else
fi
```

- ▶ **Nota importante:** en la condición, 0 significa “verdadero”, otro valor significa “falso”

# Sentencias condicionales (2/2)



- ▶ Ejemplo:

```
if test -x /bin/bash ; then
    echo "/bin/bash es ejecutable"
else
    echo "/bin/bash no es ejecutable"
fi
```

- ▶ También:

```
if [ -x /bin/bash ] ; then...
```



# Condiciones (1/2)

## ► Cadenas:

- ▷ `cadena1 = cadena2` Verdadero si son iguales
- ▷ `cadena1 != cadena2` Verdadero si no son iguales
- ▷ `-n cadena` Verdadero cadena no nula
- ▷ `-z cadena` Verdadero si cadena nula

## ► Ficheros

- ▷ `-d fichero` es un directorio
- ▷ `-e fichero` existe
- ▷ `-f fichero` es un fichero regular
- ▷ `-r fichero` tiene permisos de lectura
- ▷ `-s fichero` tiene longitud  $> 0$
- ▷ `-w fichero` tiene permisos de escritura
- ▷ `-x fichero` tiene permisos de ejecución

# Condiciones (2/2)



## ► Aritméticas

- |                             |  |
|-----------------------------|--|
| ▷ expresión1 -eq expresión2 | ambas expresiones son iguales              |
| ▷ expresión1 -ne expresión2 | ambas expresiones no son iguales           |
| ▷ expresión1 -gt expresión2 | $\text{expresión1} > \text{expresión2}$    |
| ▷ expresión1 -ge expresión2 | $\text{expresión1} \geq \text{expresión2}$ |
| ▷ expresión1 -lt expresión2 | $\text{expresión1} < \text{expresión2}$    |
| ▷ expresión1 -le expresión2 | $\text{expresión1} \leq \text{expresión2}$ |
| ▷ ! expresión               | expresión es falsa                         |

# Bucles for



## Bucles:

- ▶ Bucle for (I):  

```
for variable in valores  
do  
    cuerpo del for  
done
```
- ▶ Ejemplo  

```
for i in `seq 0 1 9`  
do  
    echo $i  
done
```

## Bucles:

- ▶ Bucle for (II):  

```
for ((i=0; $i<10; i++))  
do  
    echo $i  
done
```

# Bucles while



- ▶ Bucle while:

```
while condición ; do  
    cuerpo del while  
done
```

- ▶ Ejemplo:

```
while [ $# -gt 0 ] ; do  
    echo $1 ; shift  
done
```

# Expresiones regulares (1/2)



- ▶ Son un mecanismo muy potente para la búsqueda de patrones en cadenas de caracteres.
- ▶ Bloques básicos:
  - ▷ carácter: coincide con un carácter concreto. P.e. a
  - ▷ . : (punto) coincide con cualquier carácter.
  - ▷ ^ : principio de línea.
  - ▷ \$ : final de línea.
  - ▷ [lista] : cualquier carácter dentro de lista
  - ▷ [^lista] : cualquier carácter fuera de lista

# Expresiones regulares (2/2)



- ▶ Operadores de repetición. El elemento precedente concuerda:
  - ▷ `?` : como mucho una vez (puede ser ninguna).
  - ▷ `*` : cero o más veces.
  - ▷ `{n}` : exactamente  $n$  veces.
  - ▷ `{n, }` :  $n$  o más veces.
  - ▷ `{, m}` : como mucho  $m$  veces.
  - ▷ `{n, m}` : al menos  $n$  veces y no más de  $m$ .



# Ejemplos



## ► Ejemplos

- ▷  $a$  : cualquier cadena que contenga al menos una  $a$ .
- ▷  $ab^*$  : cadena que contenga al menos una  $a$ , seguida o no de  $b$ 's
- ▷  $ab$  : cualquier cadena que contenga la subcadena "ab"
- ▷  $a.b$  : cualquier cadena que tenga una  $a$  y una  $b$ , separadas por un carácter cualquiera.
- ▷  $^[abc]$  : cualquier línea que comience por  $a$ ,  $b$  ó  $c$ .
- ▷  $[^abc]$  : cualquier cadena que contenga cualquier carácter distinto de  $a$ ,  $b$  ó  $c$ .

# Desarrollo de aplicaciones C en Linux



## ■ Tres alternativas

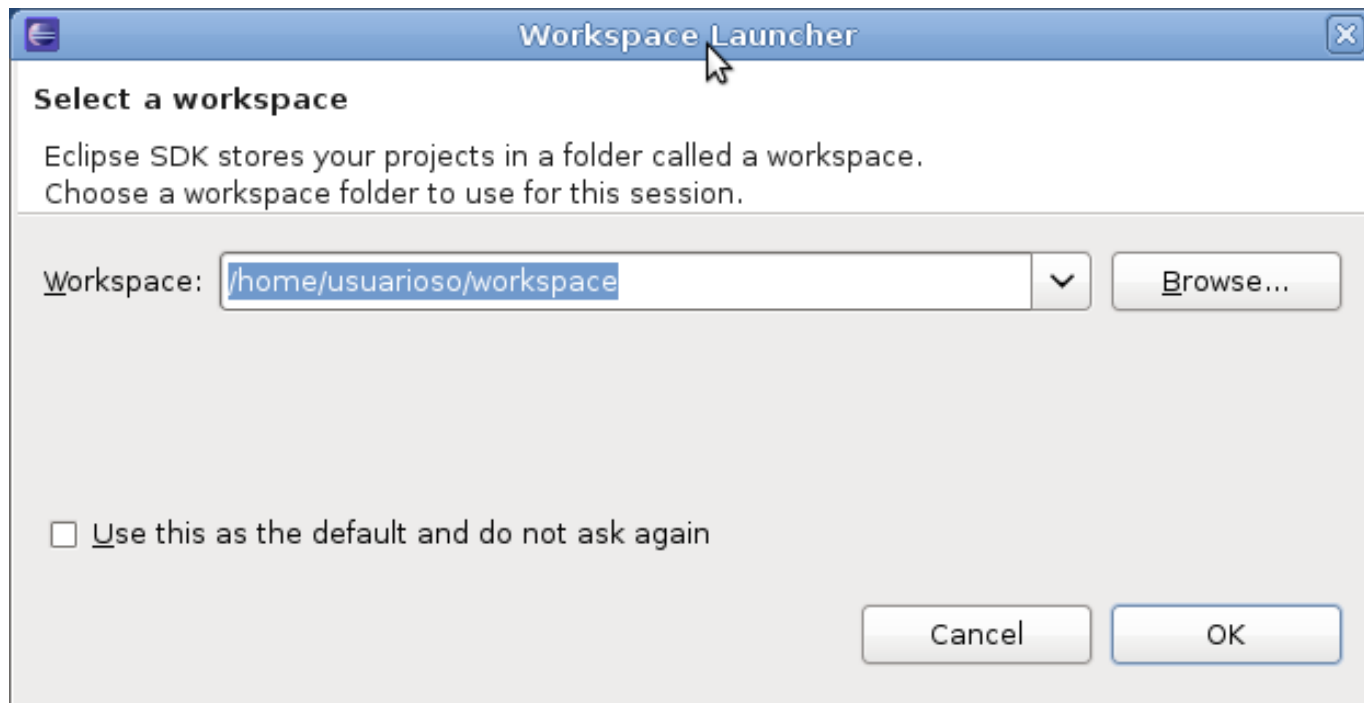
- ✓ Editor de Textos + GNU Make + Terminal
- ✓ Proyecto "C" con Makefile + IDE
- ✓ Proyecto "C" + IDE (compilación autogestionada)

## ■ Eclipse es el IDE instalado en las distribuciones de GNU/Linux del laboratorio

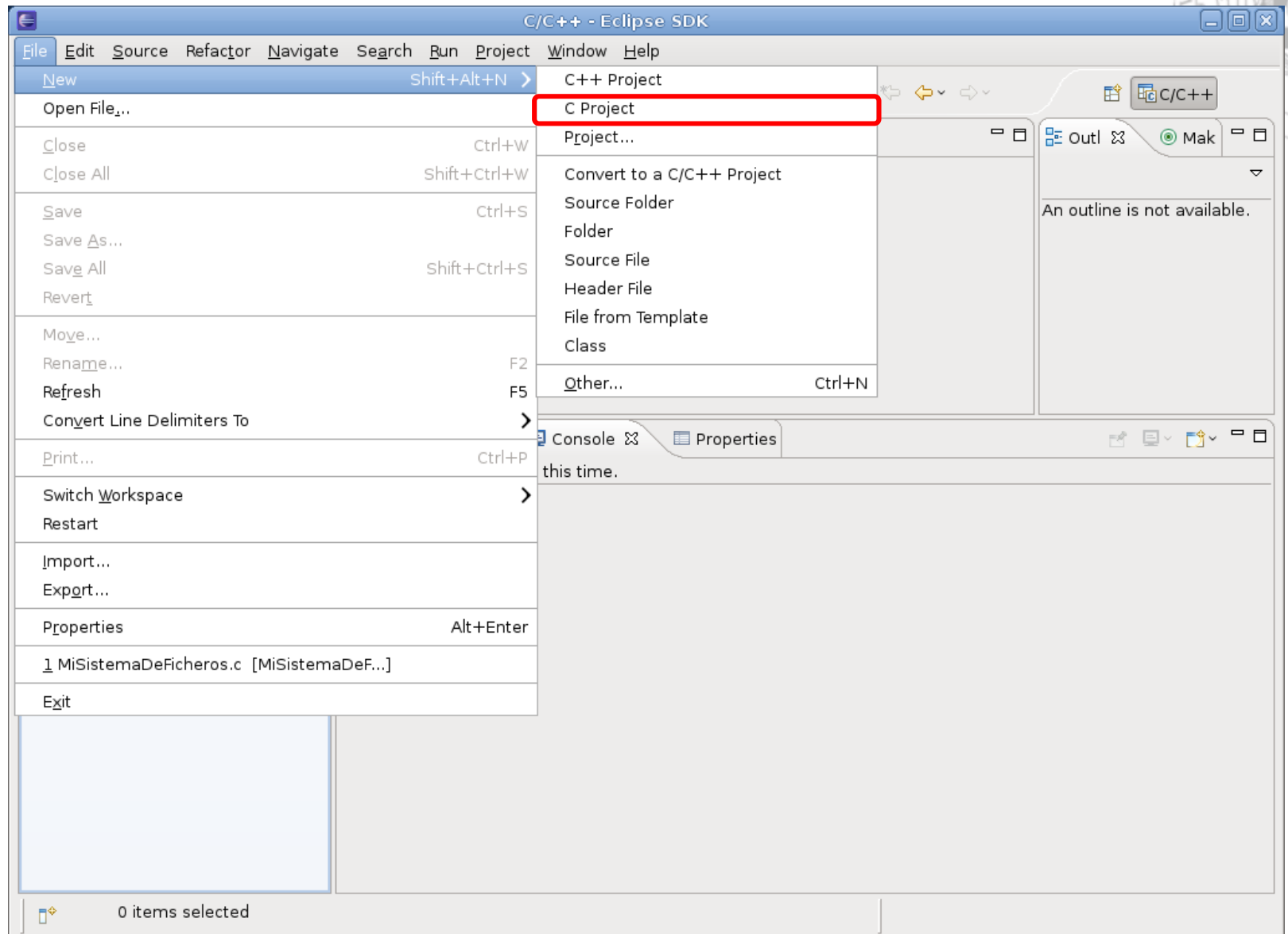
## ■ Para el desarrollo de las prácticas de esta asignatura usaremos las dos primeras alternativas (basadas en Make)

- ✓ Con cada práctica y programa de ejemplo se incluirá un Makefile para la compilación del proyecto asociado
- ✓ Ventaja: según convenga podemos cambiar entre el IDE y el terminal para ejecutar el programa o depurarlo de distintas formas

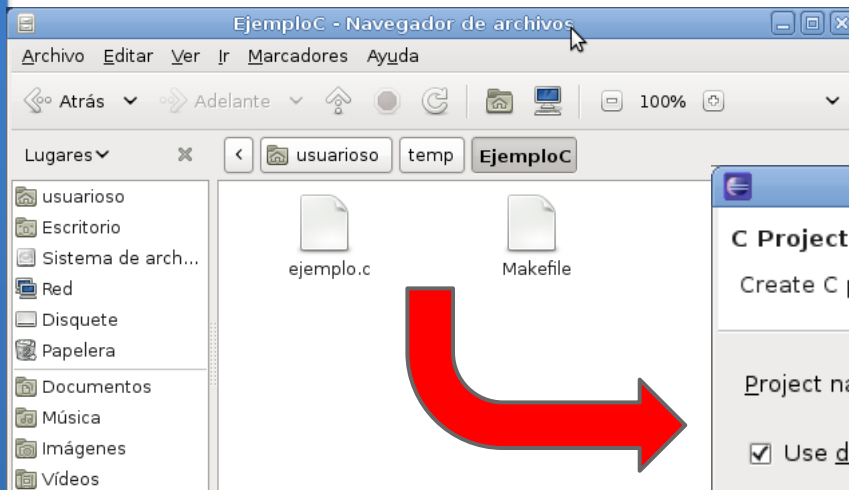
# Apertura/Creación de *workspace*



# Proyecto vacío / Compil. autogestionada



# Importar proyecto con Makefile



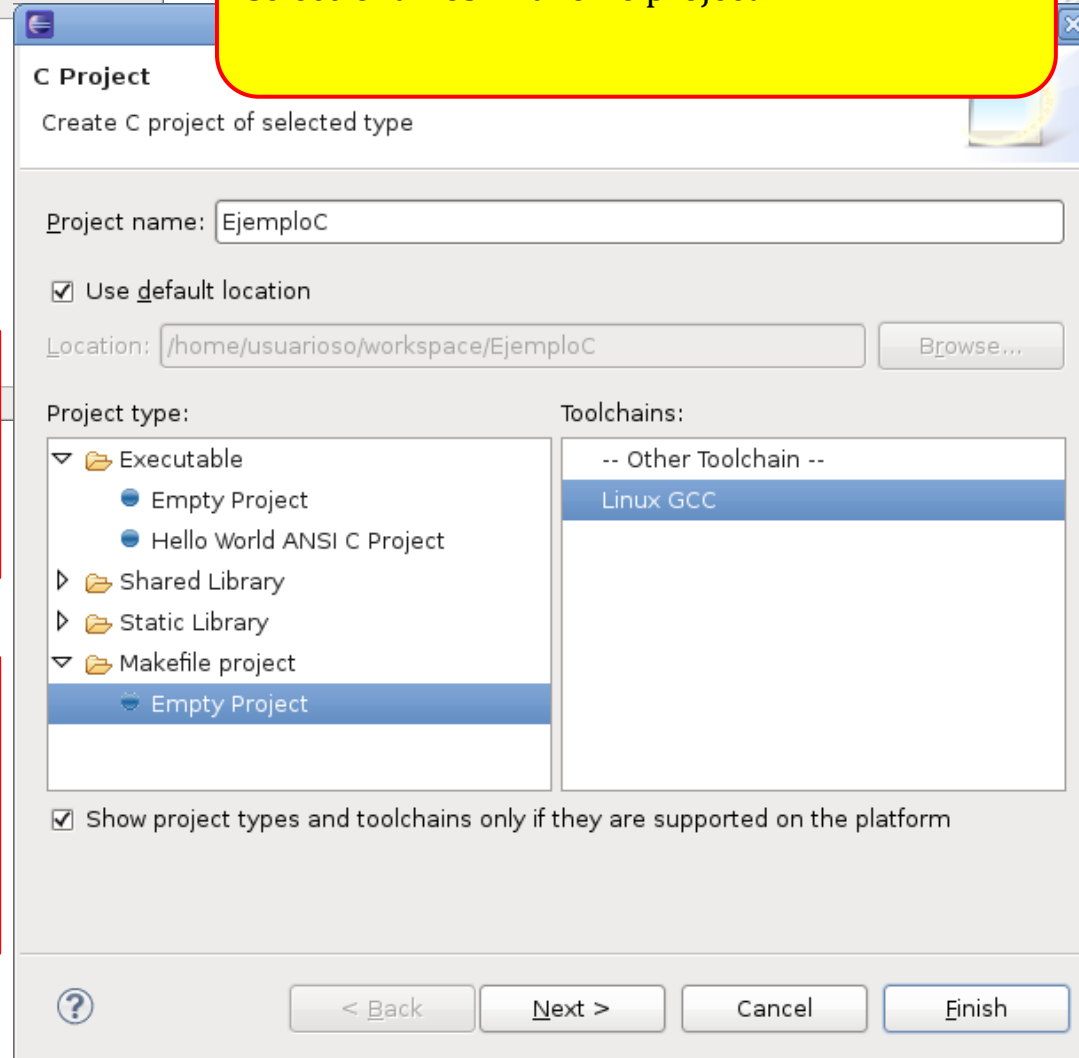
**Queremos importar un proyecto existente:**  
Creamos un nuevo proyecto C vacío, pero seleccionamos "Makefile project"

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("¡Hola %s %s!\n", argv[0],
          argv[1]);
    return 0;
}
```

```
all: ejemplo

ejemplo: ejemplo.c
    gcc -g -Wall ejemplo.c -o ejemplo

clean:
    rm ejemplo
```



# Importar proyecto con Makefile



The screenshot shows the Eclipse IDE interface. A context menu is open over a project in the Project Explorer. A yellow callout box with a red border contains the text "[...] Botón derecho sobre el nombre del proyecto" (Right-click button over the project name). A red arrow points from this text to the "Import..." option in the context menu, which is also highlighted with a red rectangle. Another red arrow points from the "Import..." option to the "Import" dialog box. The dialog box has a "Select" tab and a list of import sources. The "File System" option is selected and highlighted with a blue bar. At the bottom of the dialog, the "Next >" button is highlighted with a red rectangle.

[...] Botón derecho sobre el nombre del proyecto

Import

Select

Import resources from the local file system into an existing project.

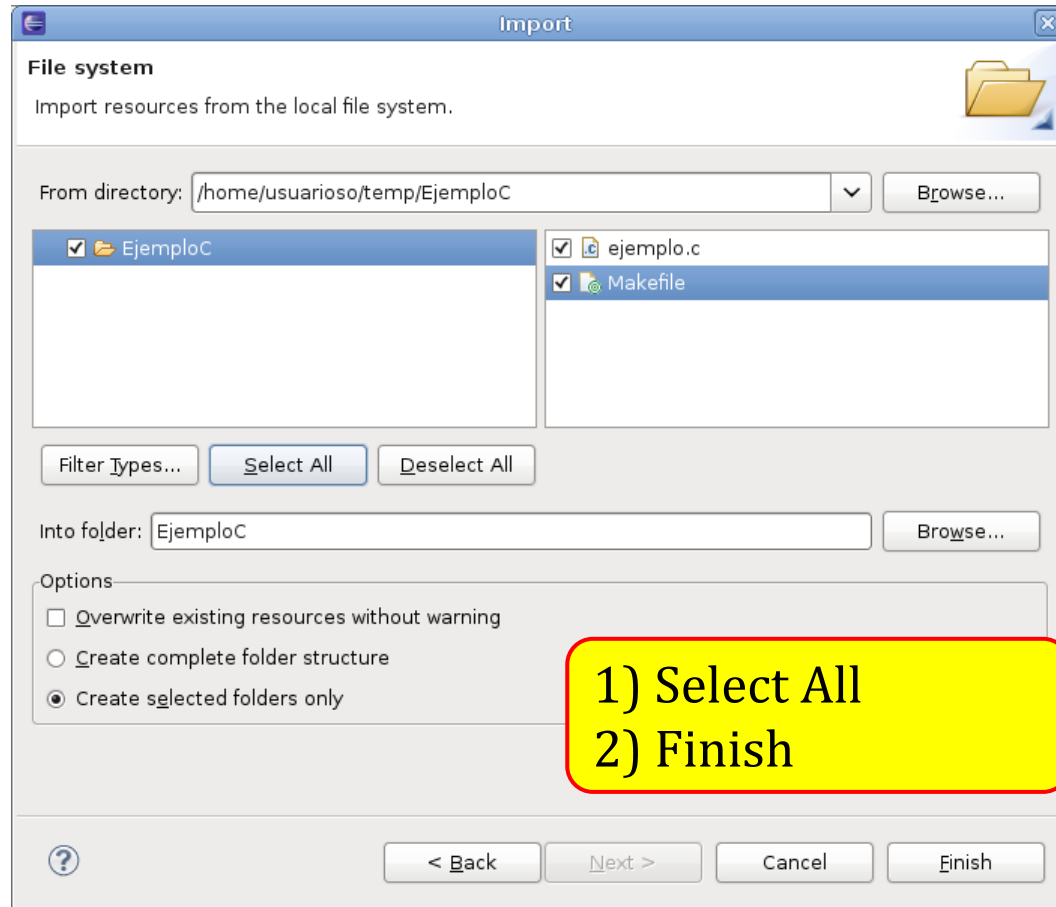
Select an import source:

type filter text

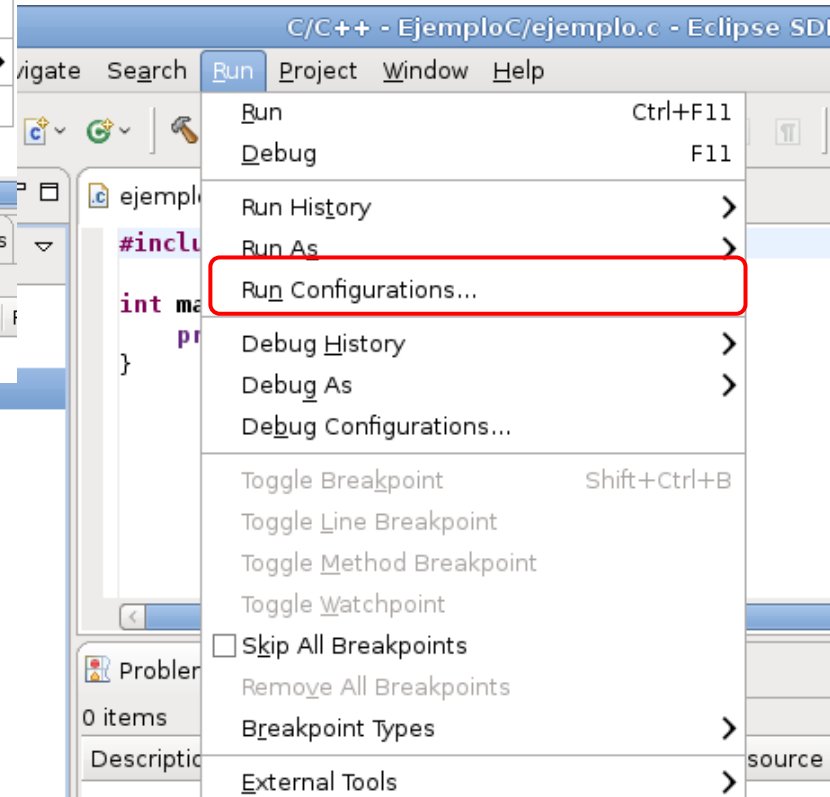
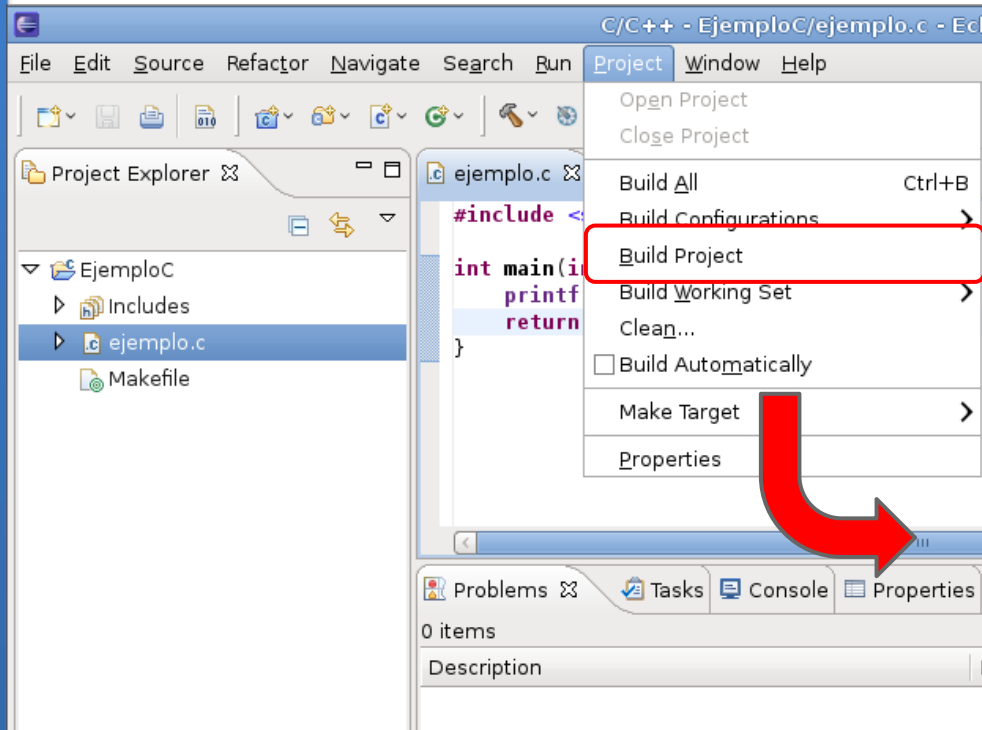
- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- CVS
- Plug-in Development
- Run/Debug
- Team

< Back Next > Cancel Finish

# Importar proyecto con Makefile



# Crear perfil de ejecución



- 1) Project -> Build Project
- 2) Run -> Build Configurations ...



# Crear perfil de ejecución



Ahora nos vamos a la pestaña "Arguments"

Create, manage, and run configurations

type filter text

- C/C++ Application
- Eclipse Application
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- OSGi Framework

Configure launch settings

- Press the 'New' button
- Press the 'Import' button
- Press the 'Export' button
- Edit the configuration

Configure

Name: EjemploC Default

Project: EjemploC

Build Configuration: Default

C/C++ Application: ejemplo

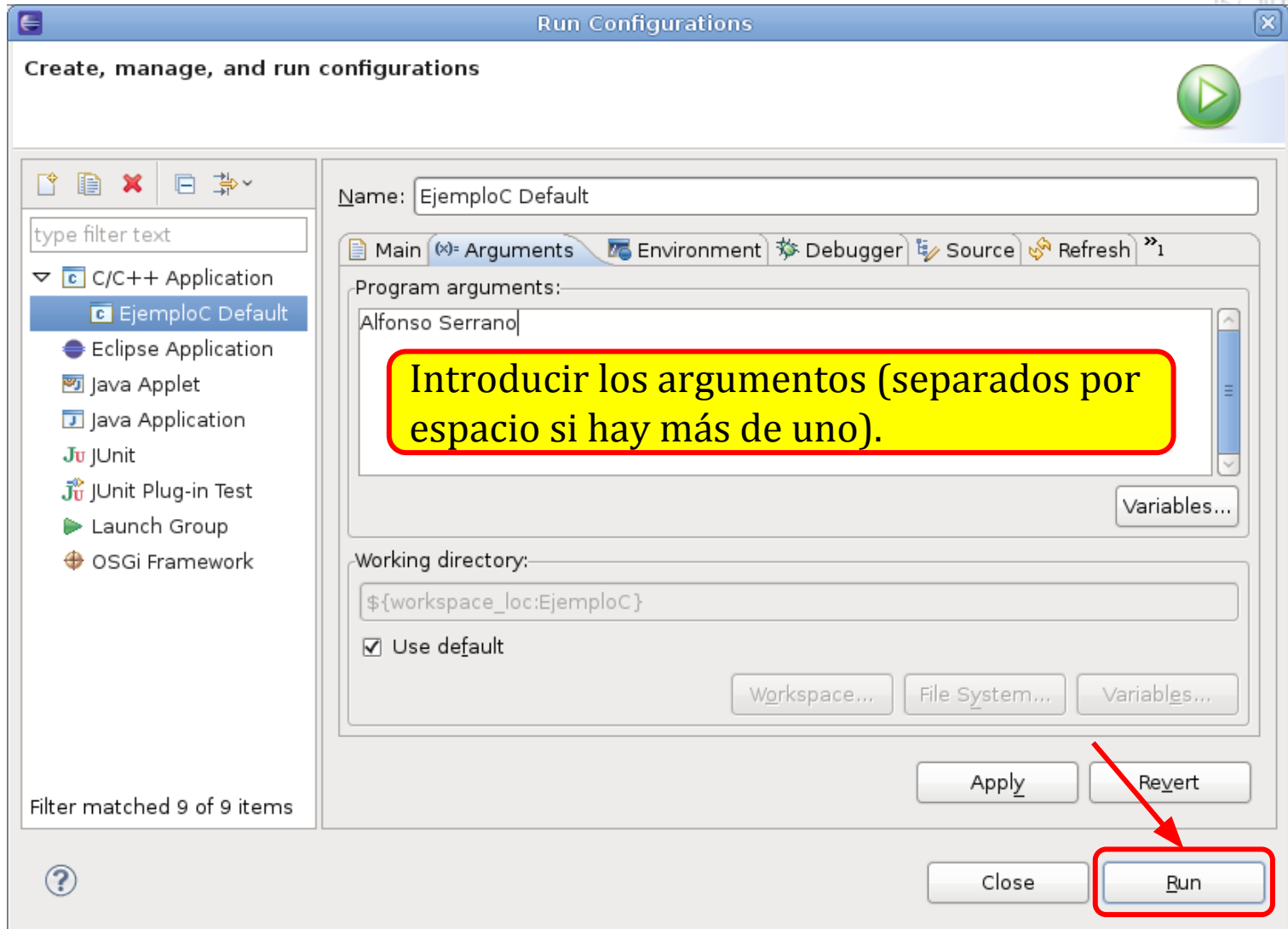
☒ Connect process input & output to a terminal.

Filter matched 9 of 9 items

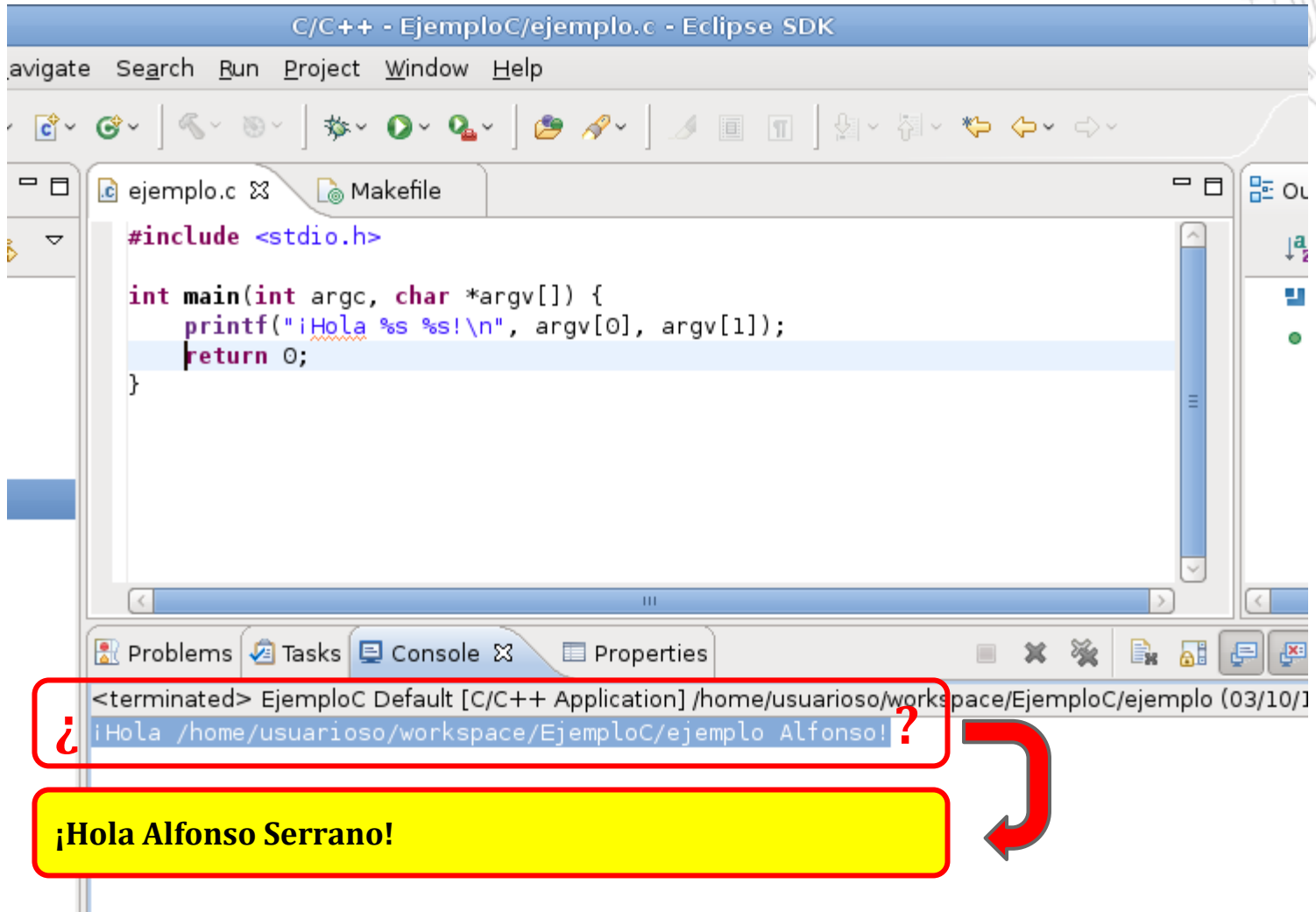
Apply Revert

Close Run

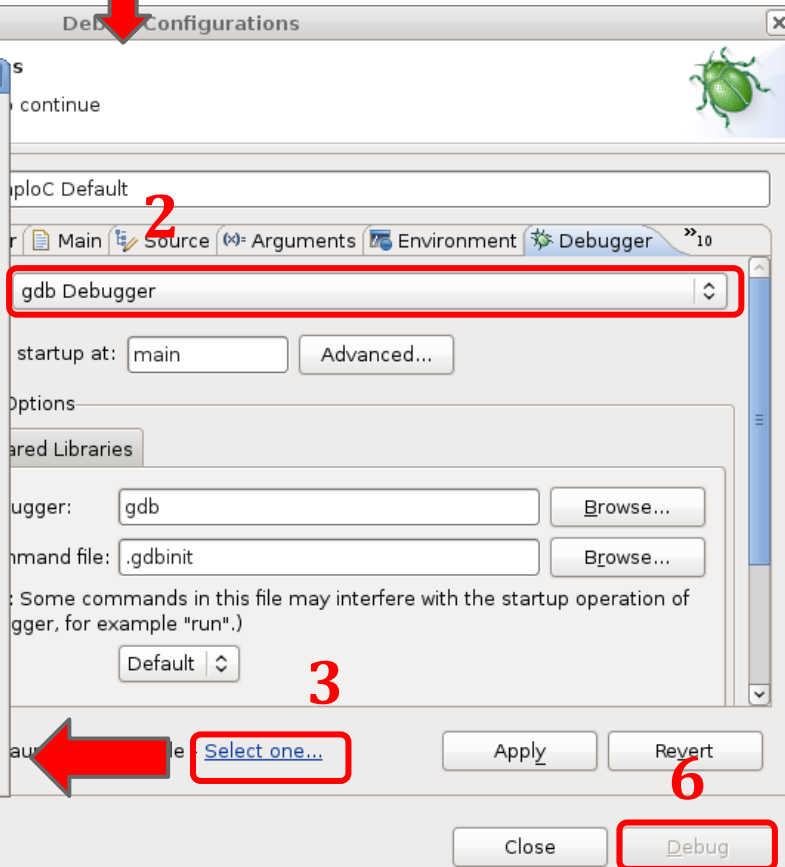
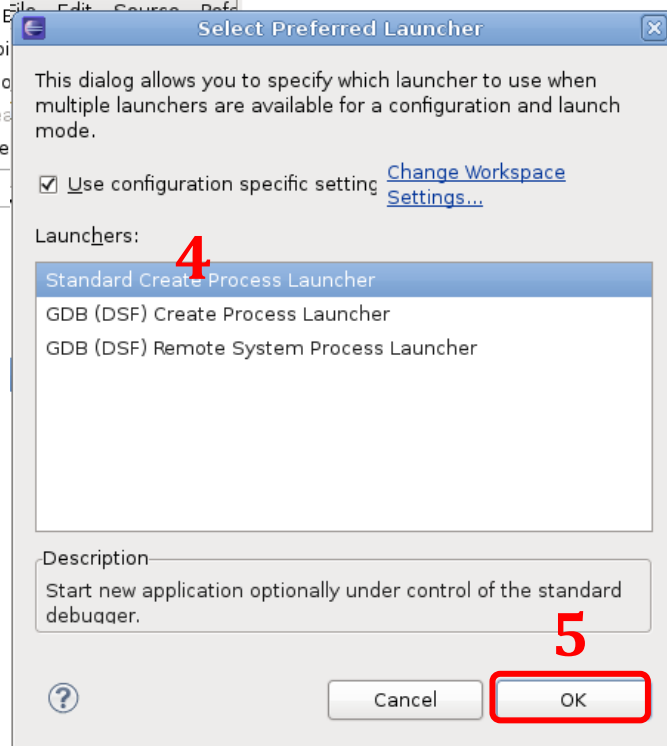
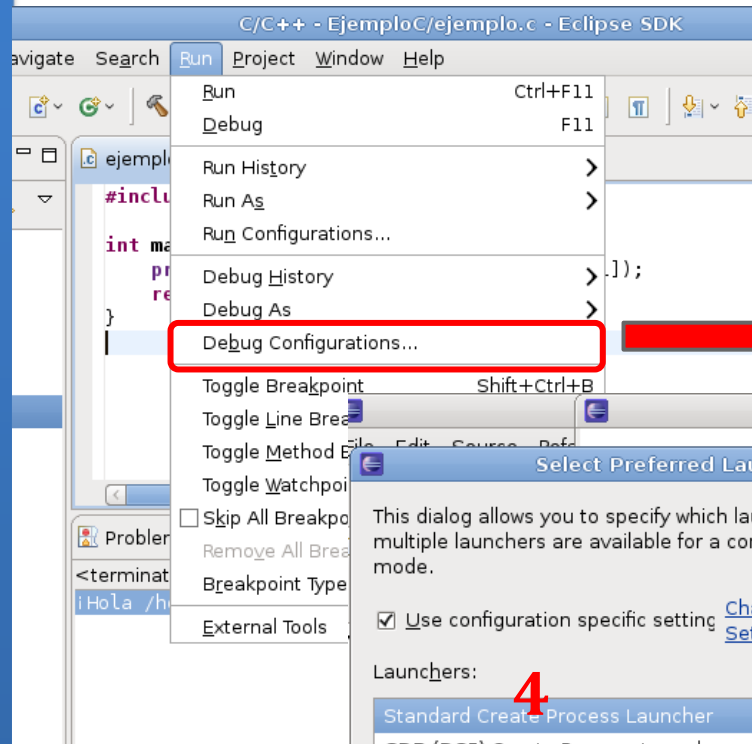
# Crear perfil de ejecución



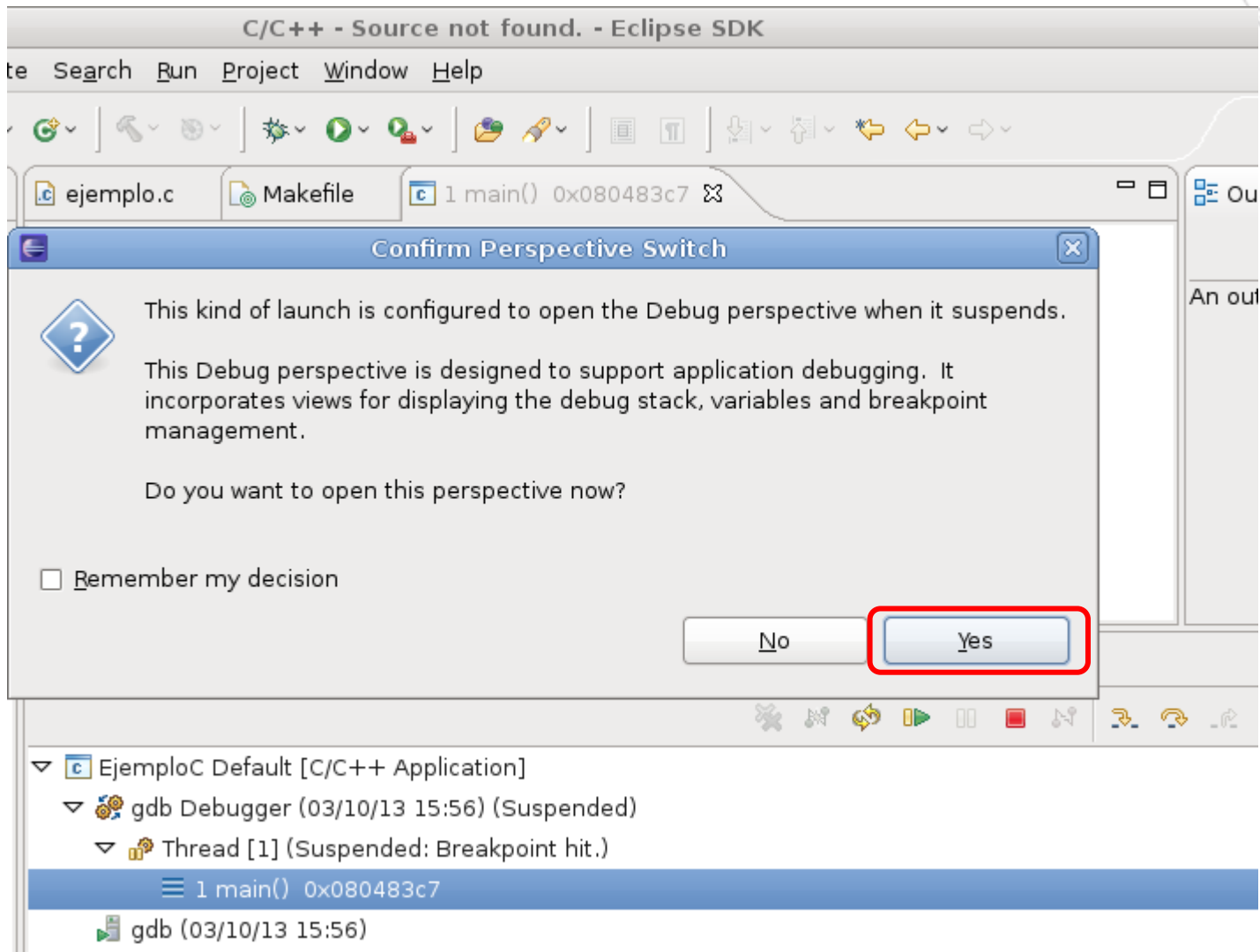
# Crear perfil de ejecución



# Crear perfil de depuración

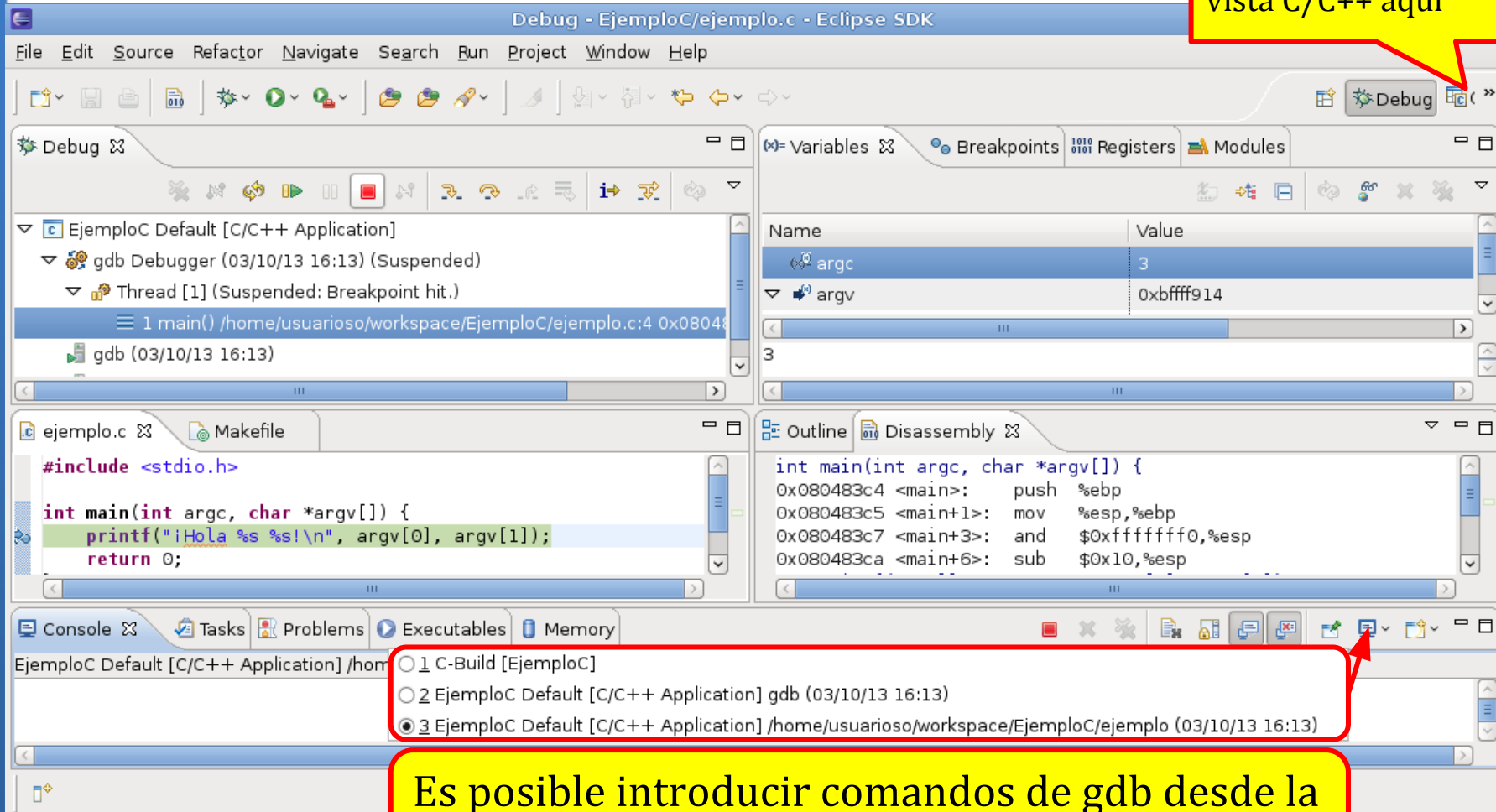


# Crear perfil de depuración



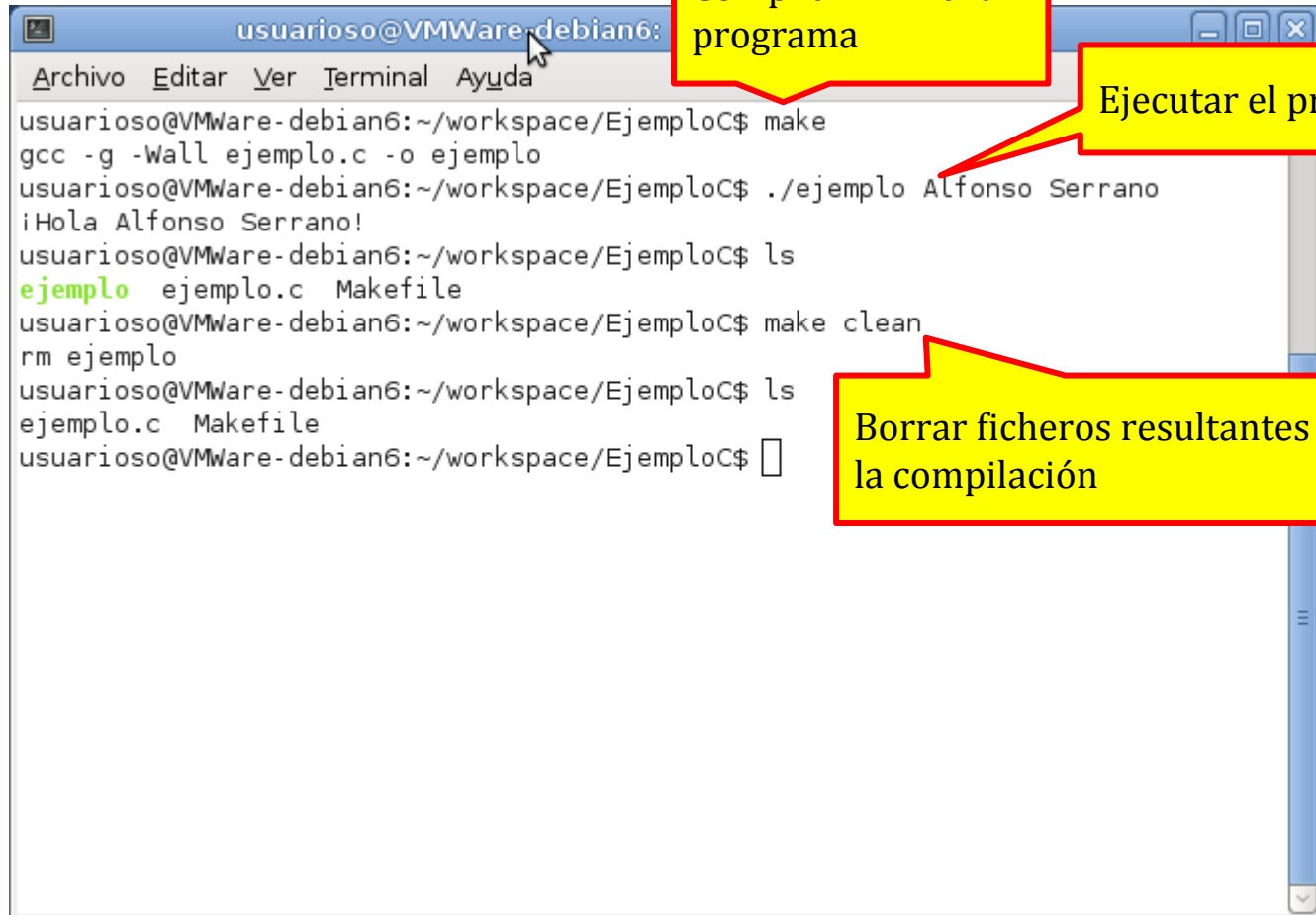
# Crear perfil de depuración

Se puede volver a la vista C/C++ aquí



Es posible introducir comandos de gdb desde la consola seleccionando el modo "gdb"

# Usando make



```
usuario@VMWare-debian6:~/workspace/EjemploC$ make
gcc -g -Wall ejemplo.c -o ejemplo
usuario@VMWare-debian6:~/workspace/EjemploC$ ./ejemplo Alfonso Serrano
¡Hola Alfonso Serrano!
usuario@VMWare-debian6:~/workspace/EjemploC$ ls
ejemplo  ejemplo.c  Makefile
usuario@VMWare-debian6:~/workspace/EjemploC$ make clean
rm ejemplo
usuario@VMWare-debian6:~/workspace/EjemploC$ ls
ejemplo.c  Makefile
usuario@VMWare-debian6:~/workspace/EjemploC$
```

Compilar + Enlazar  
programa

Ejecutar el programa

Borrar ficheros resultantes de  
la compilación

