Días de software libre

Desarrollo de aplicaciones en Linux

Roberto Gómez Cárdenas

rogomez@campus.cem.itesm.mx

http://webdia.cem.itesm.mx/dia/ac/rogomez

Tengo una aplicación a desarrollar

Los editores

- No es lo mismo un editor que un procesador de textos.
  - programas que trabajan con el texto puro, o sea, que no agregan codigo extra, por ejemplo para darle formato de negrita y cursiva a un parrafo.
- Editores en linux:
  - vi
  - emacs
  - gedit
  - pico
  - editor avanzado de KDE

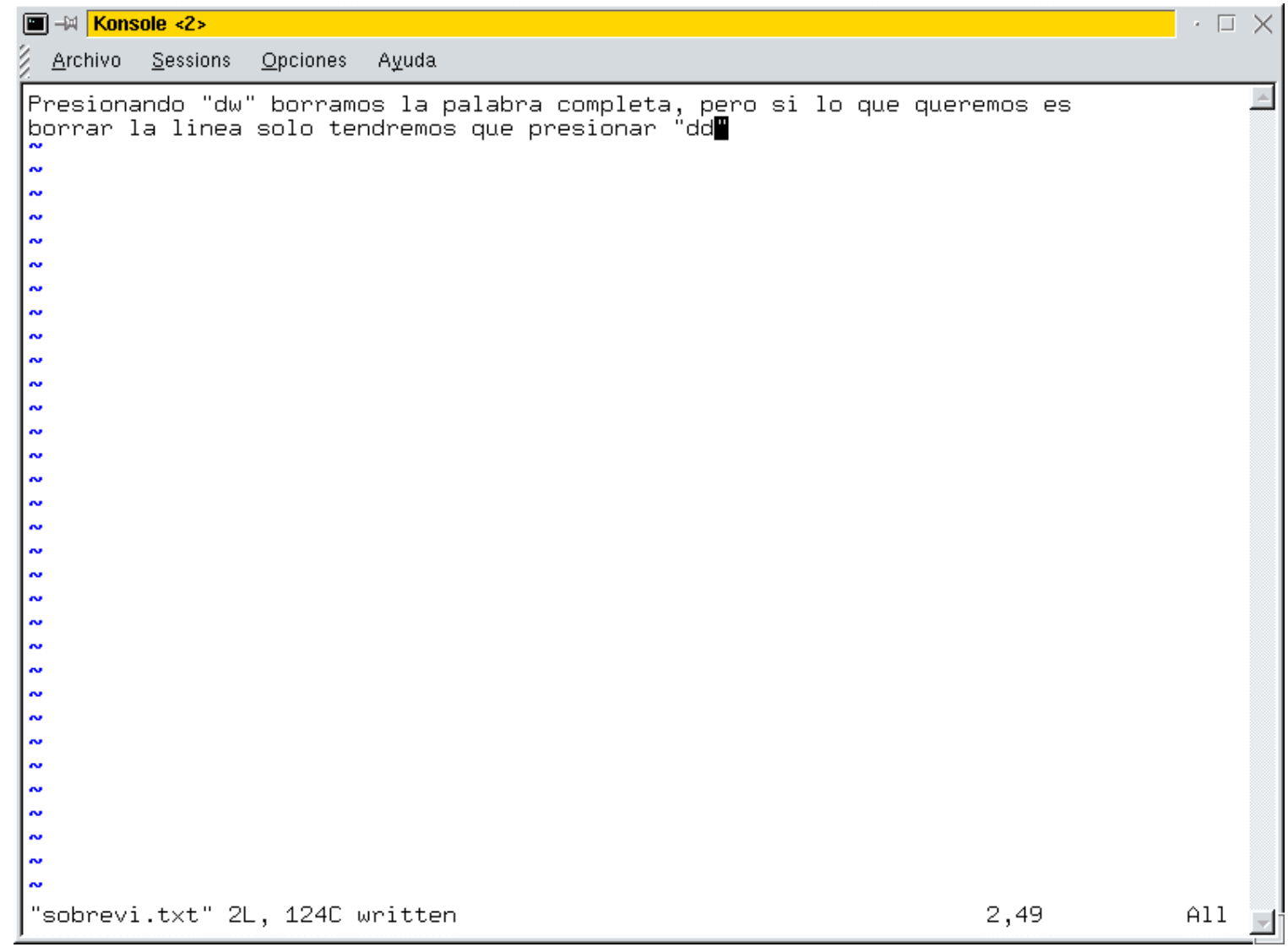
Roberto Gómez C.

El editor vi

- vi es el editor de textos garantizado en todos los equipos con GNU/Linux y UNIX
- Es rapido y por eso es indicado para la edicion remota de archivos ya que el contenido que hay que pasar por la conexión es mínimo.
- Desarrollado en la universidad de California, Berkeley
- Proviene de editor ed
  - que después evoluciona a ex
- Muy usado para archivos de configuración
  - Expresiones regulares le proporcionan mucha fuerza
- Existen versiones de ventana

Roberto Gómez C.

Una vista de vi



Roberto Gómez C.

El editor EMACS

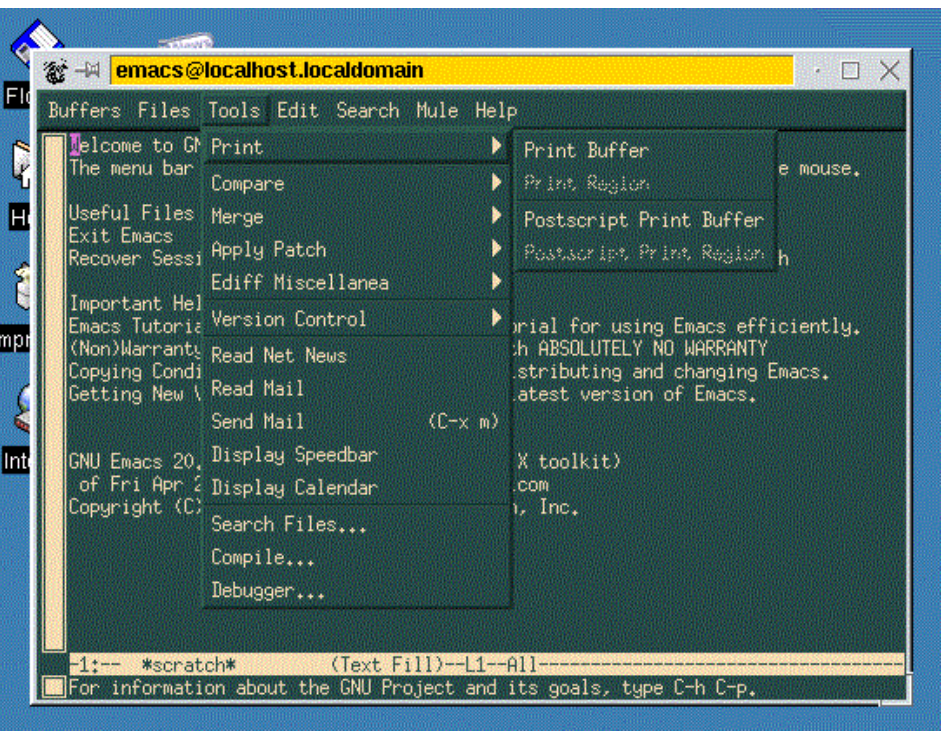
- Editor de pantalla extensible, personalizable, autodocumentado y todo eso en tiempo real.
- Permite el control de subprocesos, la autoindentacion de programas y la posibilidad de editar varios archivos a la vez entre otras.
- Los comandos de emacs pueden ser personalizados y nuevos comandos pueden ser añadidos.
- Esta escrito en Lisp y posee su propio interprete Lisp, que aceptara nuevos comandos definidos en plena edición.

Roberto Gómez C.

Características EMACS

- Presenta gran variedad de modos predefinidos,
  - facilita la utilizacion de lenguajes, tanto de programción como C, C++, Java, Fortran, ...o de formateado como HTML, Nroff, LaTEX,
  - consiguiendo en todos ellos un aspecto amigable asi como atajos de teclado para las acciones mas comunes.
- Ppuede ser usado en modo texto en un terminal o en modo grafico.
- Una version un poco mas sofisticada de emacs existe y se llama XEmacs.
  - los dos editores son muy parecidos, pero este último es mas completo.

Algunas vistas de emacs

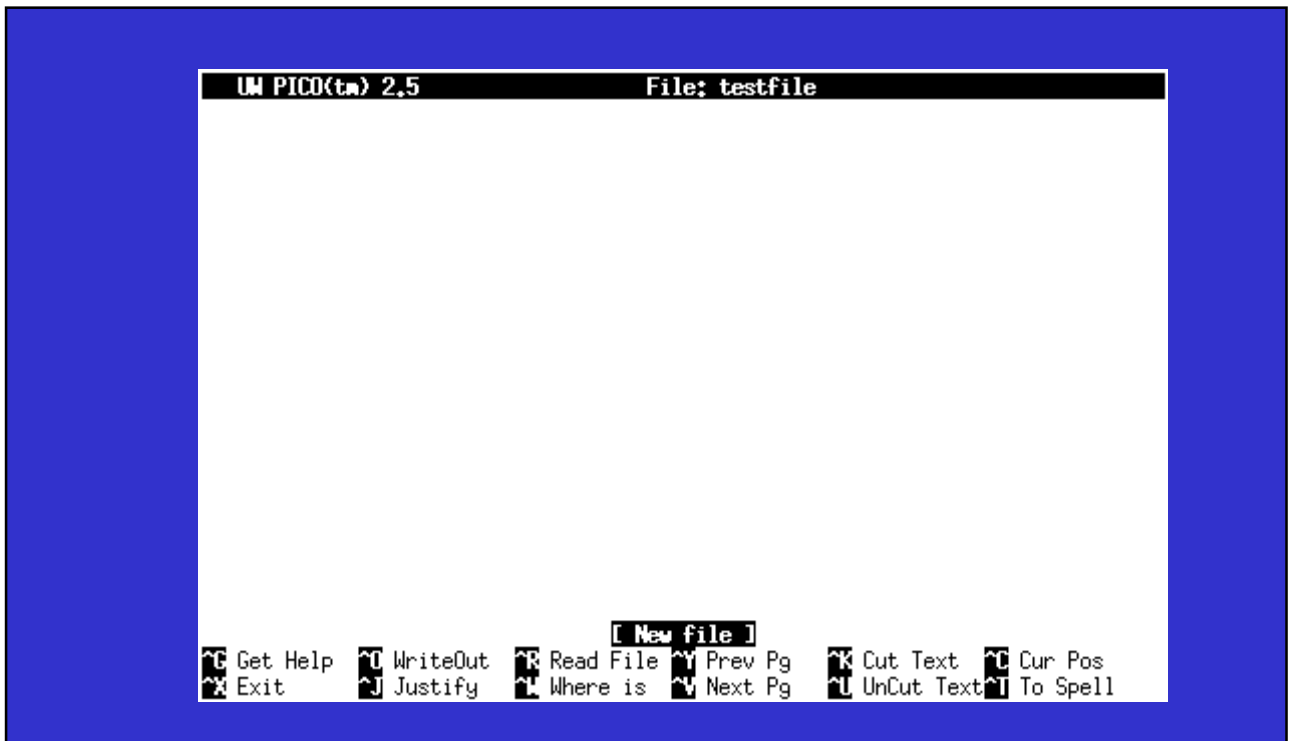



```
Buffers Files Tools Edit Search Mule Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.

Get help           C-h  (Hold down CTRL and press h)
Undo changes       C-x u       Exit Emacs              C-x C-c
Get a tutorial     C-h t       Use Info to read docs   C-h i
Activate menubar   F10  or  ESC `  or   M-`
(`C-' means use the CTRL key.  `M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)

GNU Emacs 20.6.1 (i386-mandrake-linux-gnu, X toolkit)
 of Fri Apr 28 2000 on kenobi.mandrakesoft.com
Copyright (C) 1999 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.

--11:---F1  *scratch*        (Text Fill)--L1--All----------------------------------
For information about the GNU Project and its goals, type C-h C-p.
```


Roberto Gómez C.

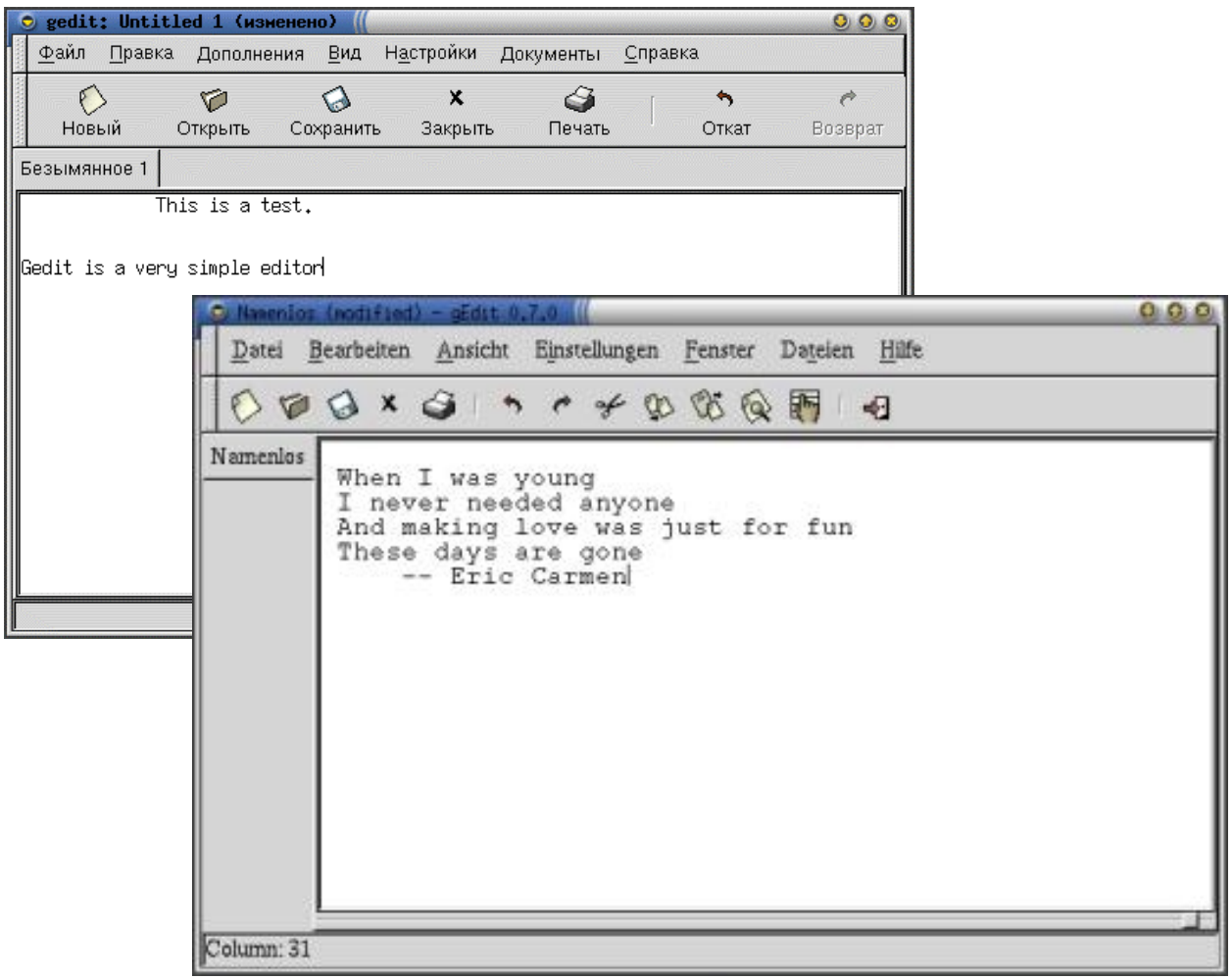
El editor pico

- Editor simple de textos, orientado a pantalla
  - display-oriented text editor.
- Comandos se encuentran parte baja de la pantalla
- Usado dentro de pine para edición de correos



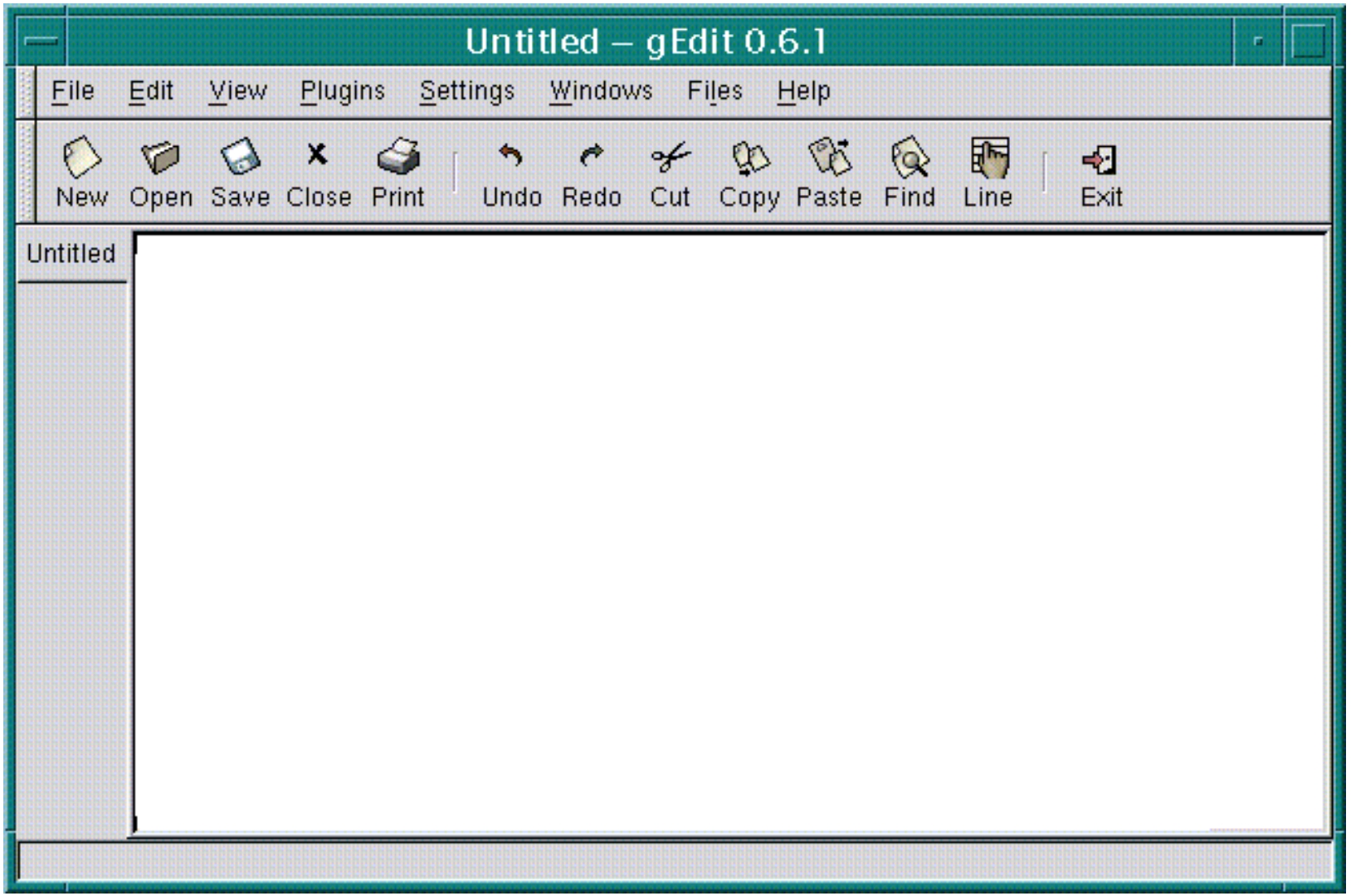
El editor gEdit

- Procesador ligero que maneja las opciones de la mayor parte de editores y algunas propias.
- Diseñado para correr bajo GNOME.



Roberto Gómez C.

Pantalla gEdit

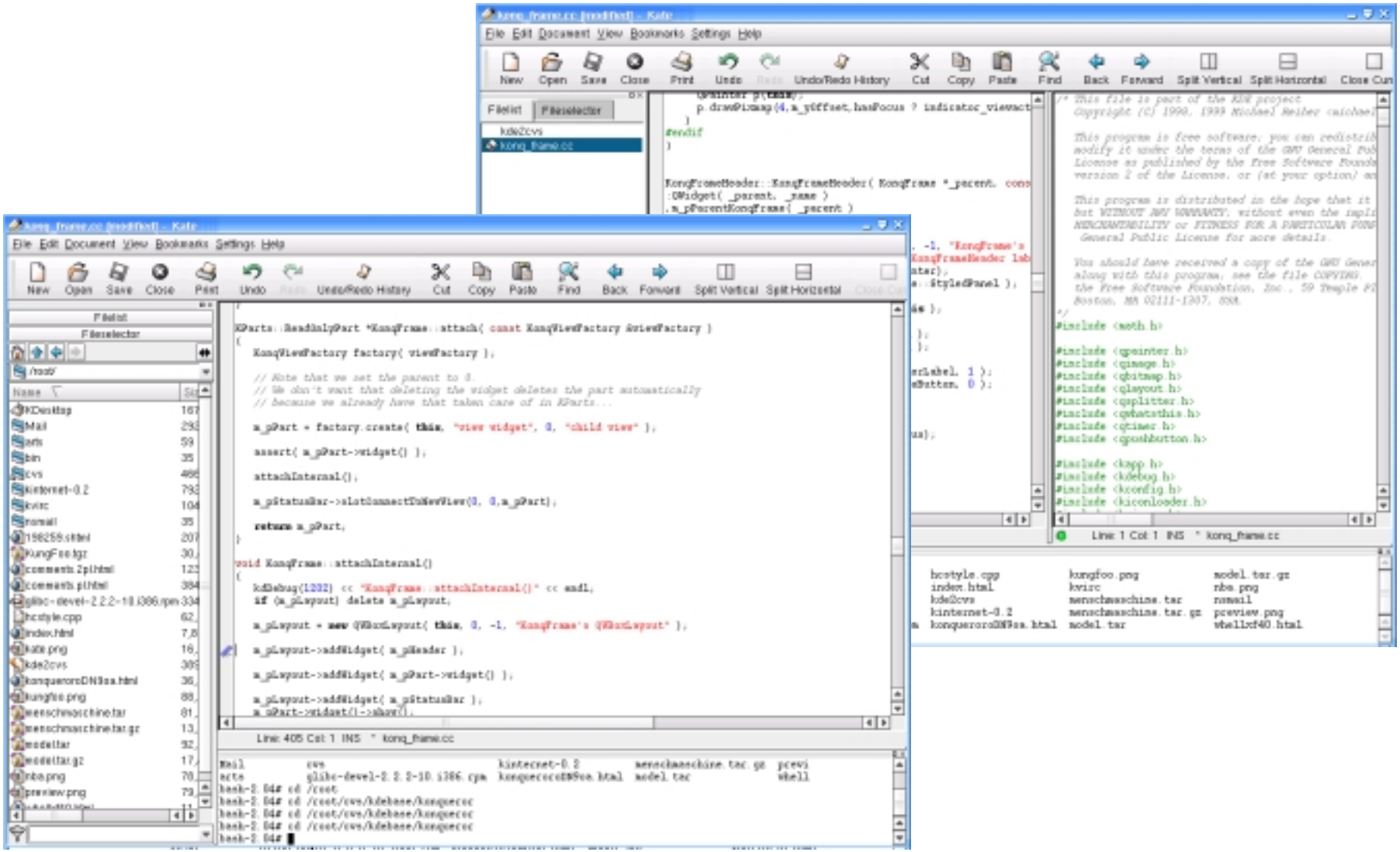


Roberto Gómez C.

El advanced editor

- Conocido como KATE: KDE Advanced Text Editor
- Editor orientado a programadores
- Soporta los lenguajes: C, C++, Objective-C, IDL, Java, Modula2, Ada, Bash, Perl, Python, HTML, LaTeX y Sather.
- Colored syntax highlighting
  - puede ser completamente personalizada
- Indentación inteligente
- Mecanismos de selección de texto avanzadas
  - persistent selection, multiple selections, block selections

Algunas pantallas de KATE



Roberto Gómez C.

El manual de Unix

- Todo, o casi todo, el sistema esta descrito en los manuales
- Dividido en secciones
  - comandos (1)
  - llamadas de sistema (2)
  - funciones (3)
- Importante saber de que sección se esta hablando.

Roberto Gómez C.

Un ejemplo: write

WRITE(1) Linux Programmer's Manual WRITE(1)

NAME

write - send a message to another user

SYNOPSIS

write user [[tty name]]

DESCRIPTION

Write allows you to communicate with other users, by copy ing lines from your terminal to theirs.

When you run the write command, the user you are writing to gets a message of the form:

Message from yourname@yourhost on yourtty at hh:mm ...

Any further lines you enter will be copied to the speci fied user's terminal. If the other user wants to reply, they must run write as well.

Roberto Gómez C.

La misma palabra, otra sección

WRITE(2) Linux Programmer's Manual WRITE(2)

NAME

write - write to a file descriptor

SYNOPSIS

#inclide <unistd.h>

size_t write(int fd, const void *buf, size_t count);

DESCRIPTION

wwrite writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf POSIX requires that a read() which can be proved to occur after a write() has returned returns the new data. Note that not all file systems are POSIX conforming.

RETURN VALUE

On success, the number of bytes written are returned (zero indicates nothing was written). On error, -1 is returned, and errno is set appropriately. If count is zero and the file descriptor refers to a regular file,

Creando programas en Unix

- Posible pasar parámetros a un programa a nivel de la línea de comandos
- Ejemplo

```
toto@cachafas:25> suma 4 5
El resultado de la suma es 9
toto@cachafas:26>
```

- Tres parámetros de la función main:
  - argc: número de parámetros
  - argv: los parametros
  - arge: parámetros de ambiente

```
main( int argc, char *argv[ ], char *arge[ ] )
```

Los parámetros función main()

- El parámetro *argc:*
  - declaración: int argc
  - es el número total de argumentos que se le pasa al programa
  - el mismo programa cuenta como uno de los parámetros
- El parámetro *argv:*
  - declaración: char *argv[]
  - almacena los argumentos
  - los argumentos son de tipo string, por lo que si hay valores numericos hay que conventirlos antes de usarlos
  - es un arreglo que empieza en argv[0] y que almacena el nombre del programa

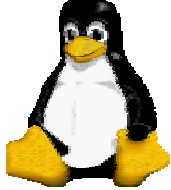
El último parámetro

- El parámetro *arge:*
  - mismas características que argv
  - almacena variables de ambiente sobre el que el programa se ejecutará
  - llamadas sistema relacionadas con arge:
    - int clearenv (void)
    - int setenv (const char * nom, const char * valeur, int ecrase)
    - int putenv (const char *string)

Ejemplo de uso

```
int main(int argc, char *argv[], char *arge[])
{

  int i;

   printf("Se introducieron %d parametros \n“,argc);
   printf("Los parametros son: \n");
   for (i=0; i<argc; i++)
     printf(" Parametro %i: %s \n",i,argv[i]);
   printf("Las variables de ambiente son:");
   while (*arge != 0)
     printf(" %s \n", *arge++);
}
```

Salida del ejemplo



```
user@bordeaux:2> Se introducieron 5 parametros
Los parametros son:
 Parametro 0: prog1
 Parametro 1: uno
 Parametro 2: dos
 Parametro 3: tres
 Parametro 4: cuatro
Las variables de ambiente son: PWD=/home/rogomez
 HOSTNAME=bordeaux
 LESSOPEN=|/usr/bin/lesspipe.sh %s
 USER=rogomez
 LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01
 INPUTRC=/etc/inputrc
 SHELL=/bin/bash
```

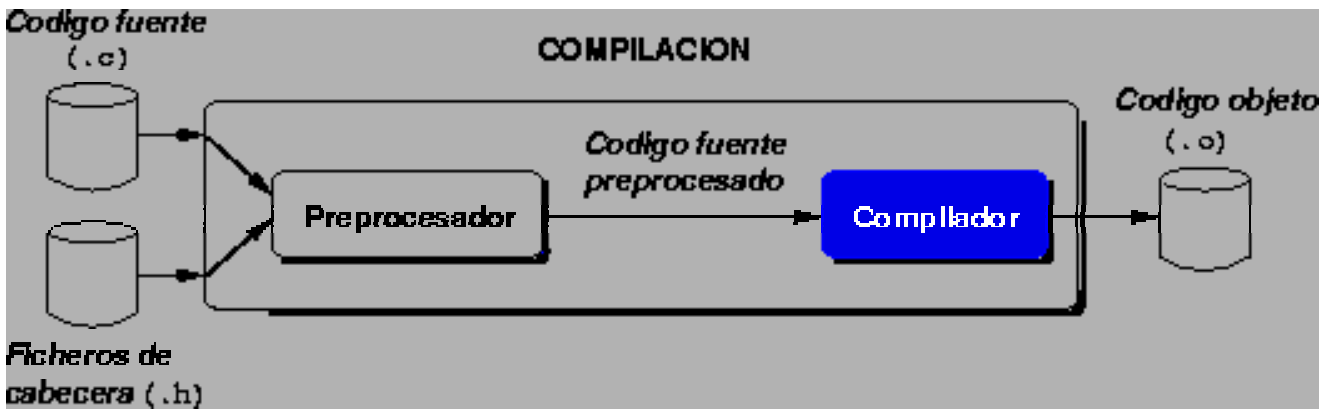
Convención interpretación argumentos

- Opciones cortas
  - consiste de un guión y de un solo caracter
  - más rápidos de teclear
- Opciones largas
  - consisten de dos guiones seguidos de un nombre que consiste de minusculas, mayusculas y guiones
  - más fáciles de recordar y de leer

Formato corto	Formato largo	Propósito
-h	**- - help**	**despliega ayudas**
-o archivo	**- - output archivo**	**especifica archivo salida**
-v	**- - verbose**	**imprime mensajes**

La compilación de programas

- La compilación tiene como objetivo
  - analizar la sintaxis y la semántica del código fuente preprocesado y traducirlo a código objeto.
- Formalmente, el preprocesamiento es la primera fase de la compilación, aunque comúnmente se conoce como compilación al análisis y generación de código objeto



El compilador gcc

- Las siglas GCC significan GNU Compiler Collection (Colección de compiladores GNU)
  - antes: GNU C Compiler (Compilador C GNU).
- Es una colección de compiladores y admite diversos lenguajes: C,C++, Objetive C, Chill, Fortran, y Java.
- El compilador se distribuye bajo la licencia GPL (General Public License)
- Existen versiones para prácticamente todos los sistemas operativos.
  - viene incluido en la mayoría (si no en todas) las distribuciones de GNU/Linux.
  - la versión DOS de este compilador es el DJGPP.

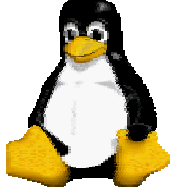
¿Qué hace gcc?

- Llama al preprocesador de C, al compilador de C, al ensamblador y al ligador.
- El preprocesador de C expande constantes simbólicas y definiciones de macro y también incluye archivos de encabezado
  - en general todo lo que es precedido por un #
- La fase de compilación crea el código de lenguaje ensamblador correspondiente a las instrucciones en el archivo fuente.
- Luego el ensamblador crea un código objeto.

¿Qué hace gcc?

- Un archivo objeto es creado por cada archivo fuente.
- Archivo objeto no es directamente ejecutable
  - necesario “ligarlo” (enlazarlo) a través de un ligador
- El ligador
  - identifica la rutina principal como el punto de entrada inicial (lugar donde empieza la ejecución)
  - asocia las referencias simbólicas a las direcciones de memoria y las une con las librerias para producir un ejecutable.
- Al final, gcc “liga” (enlaza) el código objeto y crea un archivo ejecutable

Sintaxis del compilador



gcc [-opción [argumento(s)_opción] archivo

- Cada opción va precedida por el signo **-**
  - algunas opciones no están acompañadas de argumentos (p.e. -c ó -g) de ahí que **argumento(s)_opción** sea opcional.
- **archivo**
  - indica el archivo a procesar
  - interpreta por defecto que un archivo contiene código en un determinado formato dependiendo de la extensión del archivo
    - **.c** código fuente lenguaje C
    - **.h** archivo de encabezado
    - **.cc** y **.cpp** código fuente lenguaje C++

Un primer ejemplo

```
int main(void)
{
        printf("Hola mundo \n");
        return 0;
}
```

```
user@bordeaux:2>ls
     toto.c
user@bordeaux:3>cc toto.c
user@bordeaux:4>ls
     a.out*    toto.c
user@bordeaux:5>./a.out
     Hola mundo
user@bordeaux:6>
```

Cambiando nombre del ejecutable

gcc –o ejecutable archivo

- Opción **–o**
  - especifica el nombre del archivo de salida (ejecutable), resultado de la tarea solicitada al compilador.
  - si no se especifica la opción -o, el compilador generará un archivo ejecutable (su tendencia es la de realizar el trabajo completo: compilar y enlazar) y le asignará un nombre por defecto: a.exe (MS-DOS) ó a.out (Linux/Unix).
  - si se especifica la opción -c generará el fichero objeto archivo.o

Ejemplo cambio nombre ejecutable

```
user@bordeaux:2>ls
     toto.c
user@bordeaux:3>cc -o toto toto.c
user@bordeaux:4>ls
     toto*     toto.c
user@bordeaux:5>./toto
     Hola mundo
user@bordeaux:6>
```

Generando solo el archivo objeto

gcc –c archivo

- Opción **-c**
  - solo se efectua el preprocesamiento y la compilación de los archivos fuentes.
  - no se lleva a cabo la etapa de ligado (enlazado).
  - al final produce **archivo.o**

```
user@bordeaux:2>ls
     toto.c
user@bordeaux:3>cc -c toto.c
user@bordeaux:4>ls
     toto.o    toto.c
user@bordeaux:5>
```

Archivos biblioteca y el ligador

- En fase final de la compilación:
  - ligador busca bibliotecas específicas para funciones que usa el programa en cuestión
  - combina módulos objeto para esas funciones con los módulos objeto del programa.
- Por default, el compilador busca la biblioteca estándar de C libc.a
  - contiene funciones que se encargan de las operaciones de entrada y salida
  - provee otras muchas capacidades de propósito general.

Ligando bibliotecas específicas

gcc archivo –larchivo

- Opción **-l**
  - especifica al ligador el directorio donde se encuentran los archivos de biblioteca.
  - se puede usar varias veces para especificar distintos directorios de biblioteca.
  - los archivos de biblioteca que deben usarse se especifican con la opción -larchivo
  - esta opción hace que el ligador busque en los directorios de bibliotecas (entre los que están los especificados con -l) un archivo de biblioteca llamado libarchivo.a y lo usa para ligarlo

Código del ejemplo

```
#include <math.h>

int main(void)
{
  float x;
  int y;

  y = 20;
  x = sqrt(y);
  printf("La raiz cuadrada de %d es %f \n",y,x);
  return 0;
}
```

Compilando el código

```
user@bordeaux:2>ls
    raiz.c
user@bordeaux:3>cc raiz.c
/var/tmp/ccxRaaAR1.o: In function `main':
/var/tmp/ccxRaaAR1.o(.text+0x24): undefined reference
                                                to `sqrt‘
user@bordeaux:4>
```

¿Y cómo se que biblioteca usar?

```
user@bordeaux:3>man sqrt
  Reformatting page.  Wait... done

  Mathematical Library                                sqrt(3M)

  NAME
       sqrt - square root function

  SYNOPSIS
       cc [ flag ... ] file ... -lm [ library ... ]
       #include <math.h>

       double sqrt(double x);

  DESCRIPTION
            :
            :
user@bordeaux:4>
```

Recompilando el código

```
user@bordeaux:4>ls
   raiz.c
user@bordeaux:5>cc raiz.c -lm
user@bordeaux:6>./a.out
La raiz cuadrada de 20 es 4.472136
user@bordeaux:7>
```

Especificando archivos encabezados

gcc archivo –I path

- Por defecto, gcc busca los archivos de encabezado en dos lugares en específico
  - el directorio de trabajo
  - directorio donde las librerías estandar se localizan (/usr/include/)
- Opción **–I**
  - permite especifica el directorio donde se encuentran los archivos a incluir por la directiva #include.
  - se puede utilizar esta opción varias veces para especificar distintos directorios.

Roberto Gómez C.

Ejemplo

```
user@bordeaux:4>ls
    prueba.c          archivos/
user@bordeaux:5>ls archivos
    toto.h      cachafas.h
user@bordeaux:6>more prueba.c
#include <stdio.h>
#include “toto.h”

int main ()
      :
      :
user@bordeaux:7> gcc –I archivos/ prueba.c
user@bordeaux:8>
```

Incluyendo directorios bibliotecas

- Los directorios de bibliotecas se especifican de forma análoga.
- Por ejemplo, la siguiente orden:

  gcc –o prueba -L/usr/local/lib prueba.o -lutils

- llama al ligador para que lige el objeto prueba.o con la biblioteca libutils.a y obtenga el ejecutable prueba.
- Busca el fichero de biblioteca libutils.a en el directorio /usr/local/lib (suponiendo la ejecución bajo Linux/Unix).

Pasando macros

- -D nombre[=cadena]
  - define una constante simbólica llamada nombre con el valor cadena.
  - si no se especifica el valor, nombre simplemente queda definida.
  - cadena no puede contener blancos ni tabuladores.
  - equivale a una linea #define al principio del archivo fuente, salvo que si se usa -D, el ámbito de la macrodefinición incluye todos los archivos especificados en la llamada al compilador.

Códigos de prueba

```
#include <stdio.h>
void f (void);
int main (void)
{
   printf ("MACRO en main(): %d\n", MACRO);
   f ();
   return (0);
}
```

prueba.c

```
#include <stdio.h>

void f (void)

{

   printf ("MACRO en f(): %d\n", MACRO);

}
```

funcion.c

Ejemplos definición macros

- Primer ejemplo

  gcc -DMACRO=10 prueba.c funcion.c -o ejecutable

  – la definición de MACRO se conoce en los dos archivos fuente involucrados en la llamada a gcc

  MACRO en main(): 10 MACRO en f(): 10

- Segundo ejemplo
  – no es necesario especificar un valor asociado a las macros en la llamada a gcc.
  – en este caso, por defecto se les asigna el valor 1

  gcc -DMACRO prueba.c funcion.c -o ejecutable

  MACRO en main(): 1 MACRO en f(): 1

Otras opciones importantes

- -Wall
  - muestra todos los mensajes de advertencia del compilador.
- -g
  - incluye en el ejecutable la información necesaria para poder trazarlo empleando un depurador.
- -v
  - muestra con detalle en stderr las órdenes ejecutadas por gcc.

Código ejemplo opción Wall

```
main(int argc, char *argv[])
{
   int a,b;
   a = atoi(argv[1]);
   b = atoi(argv[2]);
   printf("La suma de %d y %d es %d \n",a,b,a+b);
}
```

Compilando el código

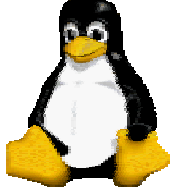
```
user@bordeaux:4>gcc toto.c –o toto
user@bordeaux:5> gcc –Wall toto.c –o toto
prog5.c:2: warning: return type defaults to `int'
prog5.c: In function `main':
prog5.c:4: warning: implicit declaration of function `atoi'
prog5.c:6: warning: implicit declaration of function `printf'
prog5.c:7: warning: control reaches end of non-void function
user@bordeaux:6>
```

Corrigiendo y compilando

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   int a,b;
   a = atoi(argv[1]);
   b = atoi(argv[2]);
   printf("La suma de %d y %d es %d \n",a,b,a+b);
   return (0);
}
```

```
user@bordeaux:4>gcc  -Wall toto.c –o toto
user@bordeaux:5>
```

Ejemplo verbose



```
user@bordeaux:7> gcc -v prueba.c
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux.....
gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
 /usr/lib/gcc-lib/i386-redhat-linux/2.96/cpp0 -lang-c -v ....
GNU CPP version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
ignoring nonexistent directory "/usr/i386-redhat-linux/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc-lib/i386-redhat-linux/2.96/include
 /usr/include
End of search list.
 /usr/lib/gcc-lib/i386-redhat-linux/2.96/cc1 /tmp/ccM7eHWy.i
GNU C version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
user@bordeaux:8>
```

El depurador gdb

- Depurador de alto nivel, capaz de analizar la ejecución de un programa en términos de las declaraciones del lenguaje C.
- También proporcionan una visión de bajo nivel para analizar la ejecución de un programa en términos de las instrucciones de la máquina.

Ejemplo de salida por error

```
toto@cachafas:412>more toto.c
main()
{
   int i;
   char *frase;
   i=3;
   i=i+1;
   frase[i]='a';
   printf("La frase es: %s \n",frase);
}
toto@cachafas:413>gcc toto.c -o toto
toto@cachafas:414>toto
Segmentation fault (core dumped)
toto@cachafas:415>
```

Depurando el programa

```
toto@cachafas:415> gcc toto.c -g -o toto
toto@cachafas:416> gdb toto
(gdb) list
1
2     main()
3     {
4       int i;
5       char *frase;
6       i=3;
7       i=i+1;
8       frase[i]='a';
9       printf("La frase es: %s \n",frase);
10    }
```

Definiendo punto ruptura y corriendo

(gdb) break 6

Breakpoint at 0x1965c; file toto.c, line 6.

(gdb) run

Starting program: /root/toto

Breakpoint 1, main() at toto.c:8
6 i=3;

Preguntando datos

```
(gdb) whatis i
type = int
(gdb) print i
$1 = -4263756
(gdb) n
7 i=i+1
```

Continuando la ejecución

(gdb) c

Continuing

Program recived signal SIGSEGV

Segmentation fault

0x10680 in main() at toto.c:8

8 frase[I]='a';

(gdb)

¿Y si debo pasar argumentos a través línea comandos?

- Dentro de gdb se puede correr:

  (gdb) run arg1 arg2 arg3 ...

  corre el programa que se está debuggeando con esos argumentos
- También es posible:

  (gdb) set args arg1 arg2 arg3 ...
  (gdb) run

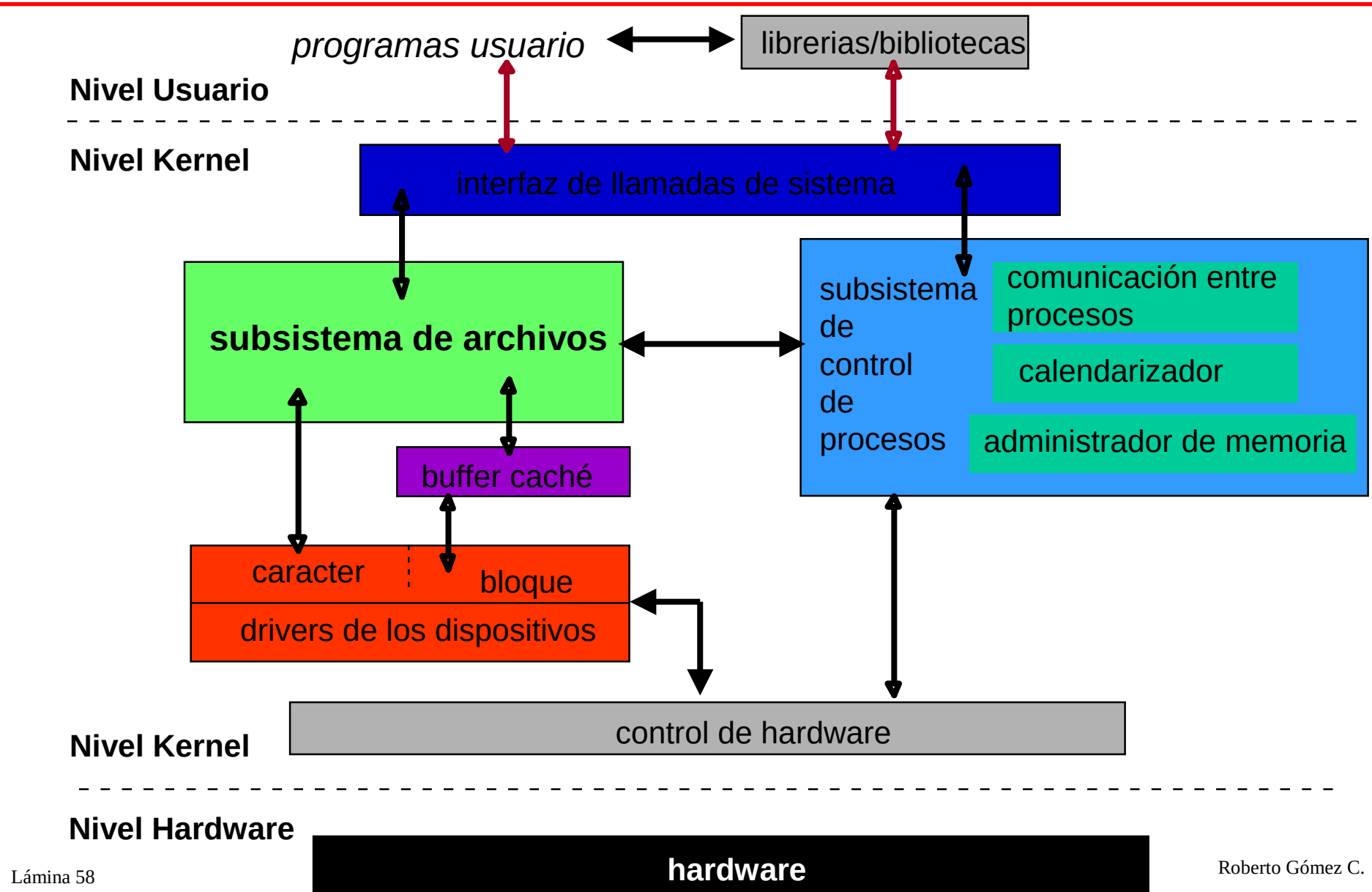
Otros depuradores

- Depuerador dbx
- Depurador xdbx
- Depurador ddd

Las llamadas de sistema

- Son los puntos de entrada a través de los cuales un proceso activo puede obtener servicios del kernel
- Es el medio a través del cual los programas de los usuarios solicitan un servicio.
- A cada llamada le corresponde un procedimiento de biblioteca que el usuario puede llamar desde sus programas
- El procedimiento pone sus parámetros en un lugar específico, y después ejecuta una llamada TRAP

Diagrama del núcleo de Unix



Llamadas de sistema en Unix

- Linux proporciona entre 60 y 200
- Toda la información acerca de ellas se encuentra en la sección 2 del manual de Unix
- Necesitan un archivo de encabezado
  - #include <math.h>
  - #include “toto.h”
  - #include “cachafas.h”
- Algunas requieren “ligarlas” con su biblioteca
  - gcc calculo.c -l math -o calculo

¿Qué regresa una llamada de sistema?

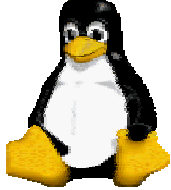
- En su mayoría regresan dos tipos de valores:
  - menor a cero: hubo un error en la ejecución
  - cero o mayor a cero: todo salió bien
- Si hubo error es posible ver la naturaleza del error a través de la página del manual de la llamada y de la variable *errno* del archivo de encabezado *errno.h*
  - tambien se puede usar la función perror()
- Es importante verificar que valor regresa una llamada.

Código ejemplo

```
main()
{
  char frase[10];
  int fd;

  fd = open("toto", O_RDONLY);
  read(fd,&frase,sizeof(frase));
  printf("Leido: %s \n",frase);
}
```

```
toto@cachafas:41>gcc toto.c -o toto
toto@cachafas:42>toto
Leido: äh @„ûÿ¿(ûÿ¿¡„  œ  ¤–  Xûÿ¿ 5 @
toto@cachafas:43>
```

Corrigiendo, compilando y ejecutando



```
main()
{
  char frase[10];
  int fd, leido;

  fd = open("toto", O_RDONLY);
  if (fd < 0) {
   printf("Error en fd \n"); exit(1);
  }
  leido = read(fd,&frase,sizeof(frase));
  if (leido < 0 ){
     printf("Error en read \n");  exit(1);
  }
  printf("Leido: %s \n",frase);
}
```

```
toto@cachafas:41>gcc toto.c -o toto
toto@cachafas:42>toto
Error en fd
toto@cachafas:43>
```

Posibles errores llamadas sistema

- El sistema puede no contar con más recursos.
- Linux puede bloquear alguna llamada de sistema intente realizar una operación para la cual no cuenta con los permisos necesarios.
- Los argumentos de la llamada de sistema pueden ser inválidos.
  - ya sea por error de programación o porque el usuario introdujo datos incorrectos
- Por razones externas al sistema
  - llamada intenta acceder dispositivo desconectado
- Llamada puede ser interrumpida por un evento externo.

¿Y cómo corregir el error?

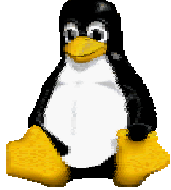
- Imprimir el número de error.
  - variable errno
- Verificar a que tipo de mneumonico pertenece
  - buscar en el archivo errno.h
- A través del manual de la llamada que lo produjo verificar que pudo provocar dicho error.
  - ejecutar comando man

Ejemplo: código lectura archivo

```
#include <strings.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>

main()
{
  int fd;
  char frase[80];

  printf("Lectura de un archivo \n");
```

```
 if ( (fd = open("toto", O_RDONLY)) < 0) {
   printf("Error no. %d en open \n",errno);
   exit(1);
 }

if ( read(fd,fd,sizeof(frase)) < 0 ) {
   perror("Error en el read");
   exit(2);
 }
 printf("Se leyo: %s \n",frase);
 close(fd);
 printf("Fin del ejemplo \n");
}
```

Compilando, ejecutando y depurando



```
user@bordeux:7>gcc cachafas.c
user@bordeaux:8>a.out
Lectura de un archivo
Error no. 2 en open
user@bordeaux:9>grep 2  /usr/include/sys/errno.h
#define  ENOENT      2     /* No such file or directory  */
#define  E2BIG       7    /* Arg list too long            */
#define  ENOMEM      12   /* Not enough core             */
#define  ENOTDIR     20   /* Not a directory          */
:
:
user@bordeaux:10>
```

Buscando en el manual de Unix

```
user@bordeaux:10>man 2 open
Reformatting page.  Wait... done

System Calls     open(2)

NAME
     open - open a file

SYNOPSIS
     #include <sys/types.h>

ERRORS
 The open() function fails if:
```

EACCES	Search permission is denied on component of the path prefix, or the file exists and the permissions specified by oflag are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or...
EEXIST	O_CREAT and O_EXCL are set, and the named file exists.
EINTR	A signal was caught during open().
EFAULT	path points to an illegal address.
ENOENT	O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the path argument points to an empty string.

:

:

user@bordeaux:11>

```
user@bordeaux:11>echo "hola mundo" > toto
user@bordeaux:12>a.out
Lectura de un archivo
Error en el read: Bad address
user@bordeaux:13>grep Bad /usr/include/sys/errno.h
#define     EBADF     9       /* Bad file number     */
#define     EFAULT    14      /* Bad address  */
user@bordeaux:14>man 2 read
System Calls                                        read(2)
NAME
    read, readv, pread - read from file
SYNOPSIS
    #include <unistd.h>
                        :
                        :
```

```
ssize_t read(int fildes, void *buf, size_t nbyte);
                        :
                        :
ERRORS
    The read(), readv(), and pread() functions will fail if:

    EAGAIN  Mandatory file/record locking was set,
            O_NDELAY or O_NONBLOCK was set,
            and there was  a blocking record lock.
     EBADF   fildes is not a valid  file  descriptor  open
             for reading.
     EBADMSG       Message waiting to be read on a stream is not
                   a data message.
    EFAULT     buf points to an illegal address.
                :
                :
user@bordeaux:15>
```

```
if ( (fd = open("toto", O_RDONLY)) < 0) {
   printf("Error no. %d en open \n",errno);
   exit(1);
 }
if (read(fd,fd,sizeof(frase)) < 0 ) {
   perror("Error en el read");
   exit(2);
 }
 printf("Se leyo: %s ",frase);
 close(fd);
 printf("Fin del ejemplo \n");
}
```

read(fd,&frase,sizeof(frase)

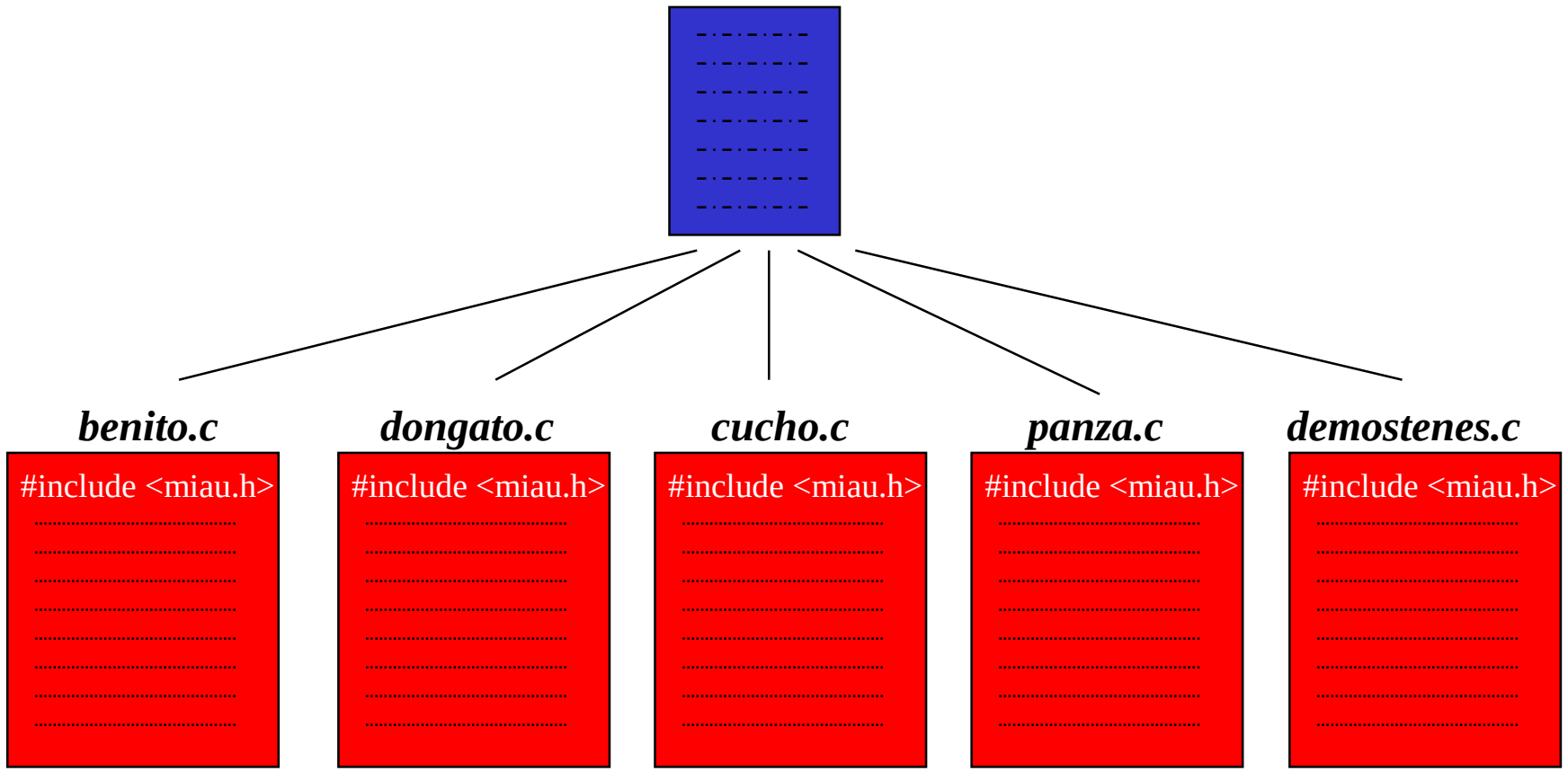
Ejecutando el resultado

```
user@bordeaux:15>a.out
Lectura de un archivo
Se leyo: hola mundo
Fin del ejemplo
user@bordeaux:16>
```

Algunas llamadas sistema

- open( )
- read( )
- write( )
- create( )
  - semget( )
  - semop( )
  - semctl( )
  - shmget( )
  - shmat( )
  - shmdt( )
  - shmctl( )
- fork( )
- getpid( )
- getppid( )
- exit( )
- wait( )
- execlp( )
- execvp( )
- signal( )
- kill ( )
- sigaction ( )
- pipe ( )
- mkfifo ( )

La utilería make



Roberto Gómez C.

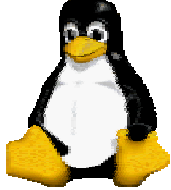
Características utilería make

- Lee una especificación de la depenencia de los componentes de un programa, así como la forma de procesarlos y actualizar una versión del programa
- Verifica el tiempo en el cual varios componentes fueron modificados por última vez
- Calcula la cantidad mínima de recompilación que se necesita hacer para crear una nueva versión consistente, y después corre los procesos
- Util para programas grandes que pueden ser divididos en varios archivos fuentes:

Formato archivo dependencia

- Líneas en blanco son ignoradas
- El carácter # es usado como comentario,
  - se ignora desde su aparición hasta el salto de línea
- Una línea con un signo de igual (=) define un macro
- Un conjunto de dependencias se declara usando dos puntos, (:),
  - en una línea los nombres a la izquierda dependen de los nombres a la derecha nombres izquierda = objetos
- Una línea de dependencia puede estar seguida de una o mas líneas indentadas de comandos

El comando make

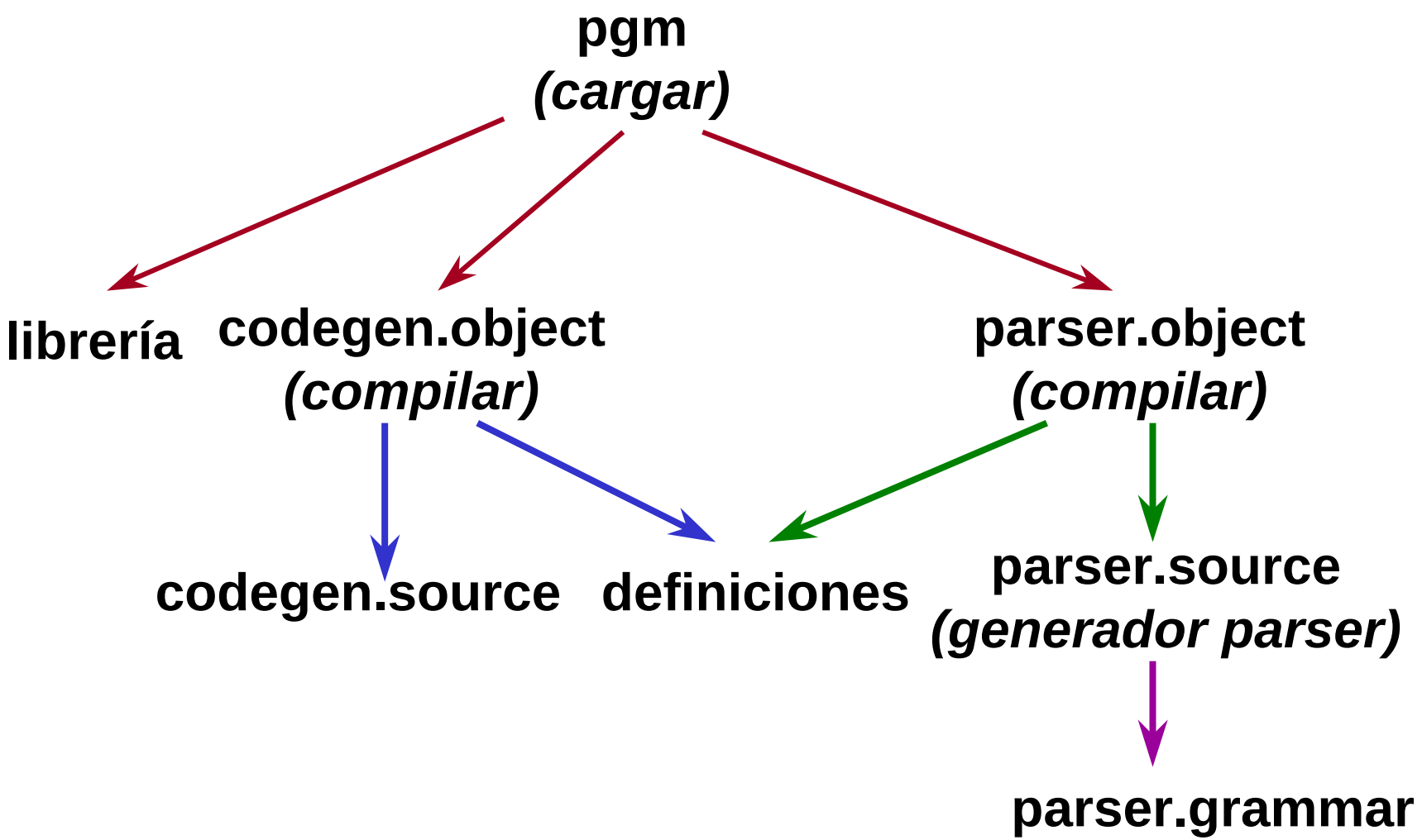


- El comando make busca por un archivo de descripción llamado *Makefile* en directorio de trabajo
- Por default *make*, (sin argumentos), inspecciona todas las líneas de dependencia y verifica las fechas de modificación de los archivos:
  - si han cambiado ejecuta los comandos asociados a dicha línea
- Si se pasa como argumento el nombre de un item que sea igual a un objeto de la línea de dependencia, los comandos asociados a dicha línea seran ejecutado

Algunas opciones utiles

- Si se desea saber que hara make sin ejecutar ningun comando se puede teclear:
  - make –n
- Si se quiere forzar que todas las fechas, (tiempo modificación archivos), sean consistentes:
  - make –t
- Si se quiere especificar un archivo en particular:
  - make -f

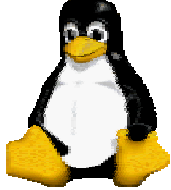
Ejemplo dependencias



Ejemplo dependencias

```
pgm:  codegen.o   parser.o    libreria
        cc codegen.o parser.o libreria -o pgm        #carga

codegen.o:  codegen.c  definiciones
        cc -c codegen.c                              # compila

parser.o:      parser.c  definiciones
        cc -c parser.c                               # compila

parser.c:      parser.y                  # genera parser en archivos
        yacc parser.y                    # file y.tab.c
        mv y.tab.c parser.c              # cambia nombre de la salida
```

Ejemplo ejecución make



```
user@bordeaux:2>ls -l
total 22
-rw-rw----   1  alex        179  Jun  21  18:20  calc.c
-rw-rw----   1 alex         354  Jun  21  16:02  calc.o
-rwxrwx---  1 alex         6337  Jun  21  16:04  compute
-rw-rw----   1  alex        780  Jun  21  18:20  compute.c
-rw-rw----   1  alex         49  Jun   21  16:04  compute.h
-rw-rw----   1  alex        880  Jun  21  16:04  compute.o
-rw-rw----   1  alex        311  Jun 21  15:56  makefile
user@bordeaux:3> make
cc     -c  -o  compute.c
cc     -c  calc.c
cc     -o  compute  compute.o  calc.o
user@bordeaux:4>
```

Recomendaciones finales

- Utilizar el editor que más les acomode
- Cuando se use una llamada de sistema hay que verificar que todo haya salido bien.
- En caso de error
  - encontrar el punto exacto (i.e. línea) donde ocurrio
  - posible usar un depurador (gdb) para lo anterior
  - utilizar el man de la función que produjo el error y verificar la subsección de errores
- Sistemas grandes dividirlos en módulos
  - utilizar la utilería make para compilar cada uno de los módulos

Días de software libre

Desarrollo de aplicaciones en Linux

Roberto Gómez Cárdenas

rogomez@campus.cem.itesm.mx

http://webdia.cem.itesm.mx/dia/ac/rogomez