Curso Programación Linux

Jose A. Espinosa

Indice

- AREA 1: Programación Linux
- AREA 2: Optimización
- AREA 3: El núcleo

AREA 1: Programación Linux

- Introducción
- Entornos de programación
- Multitarea, multihilo
- Programación gráfica
- Intercomunicación
- Compilación cruzada

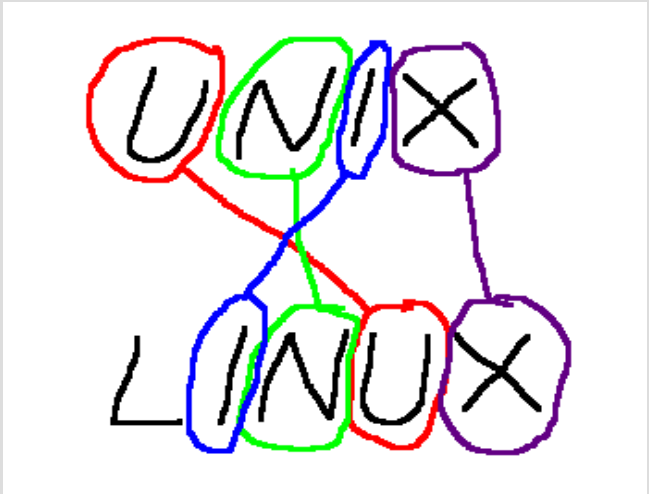
Introducción

25 de agosto de 1991, 20:57:08 GMT comp.os.minix

```
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing
since april, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to
work. This implies that I'll get something practical within a few
months, and I'd like to know what features most people would want.
Any suggestions are welcome, but I won't promise I'll implement them
:-)

Linus Benedict Torvalds (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded
fs. It is NOT portable (uses 386 task switching etc), and it probably
never will support anything other than AT-harddisks, as that's all I
have :-(.
```

Y al principio fue UNIX...

- Sistema operativo
- Portable
- Multiusuario
- Multitarea
- 99% escrito en c



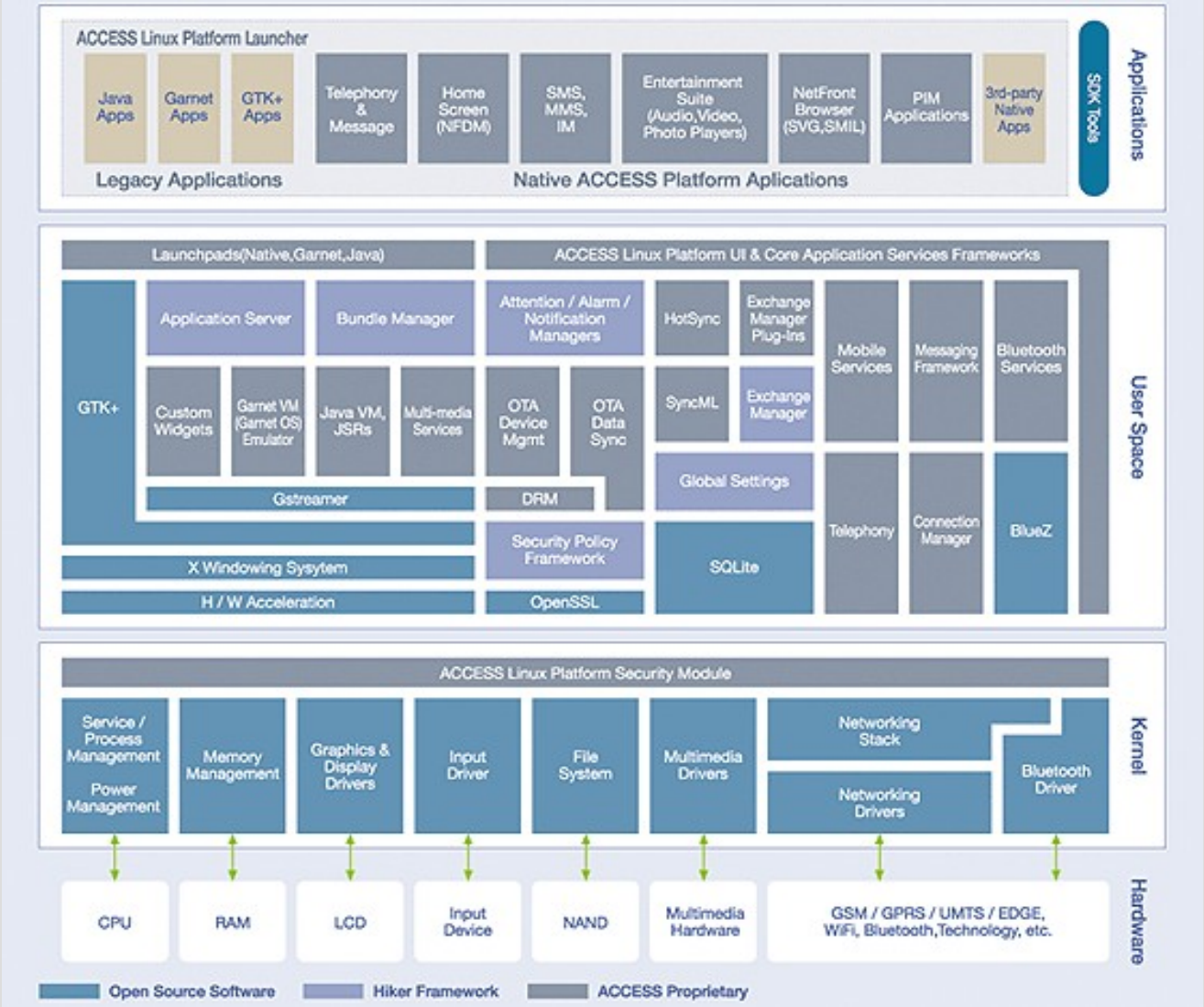
De UNIX a Linux

- 1966: MULTICS
- 1973: Porting a C del núcleo de UNIX
- 1982: System III (primer unix comercial)
- 1983: System V
- 1987: Minix
- 1990: IEEE 1003 : POSIX.1
- 1991: Linux

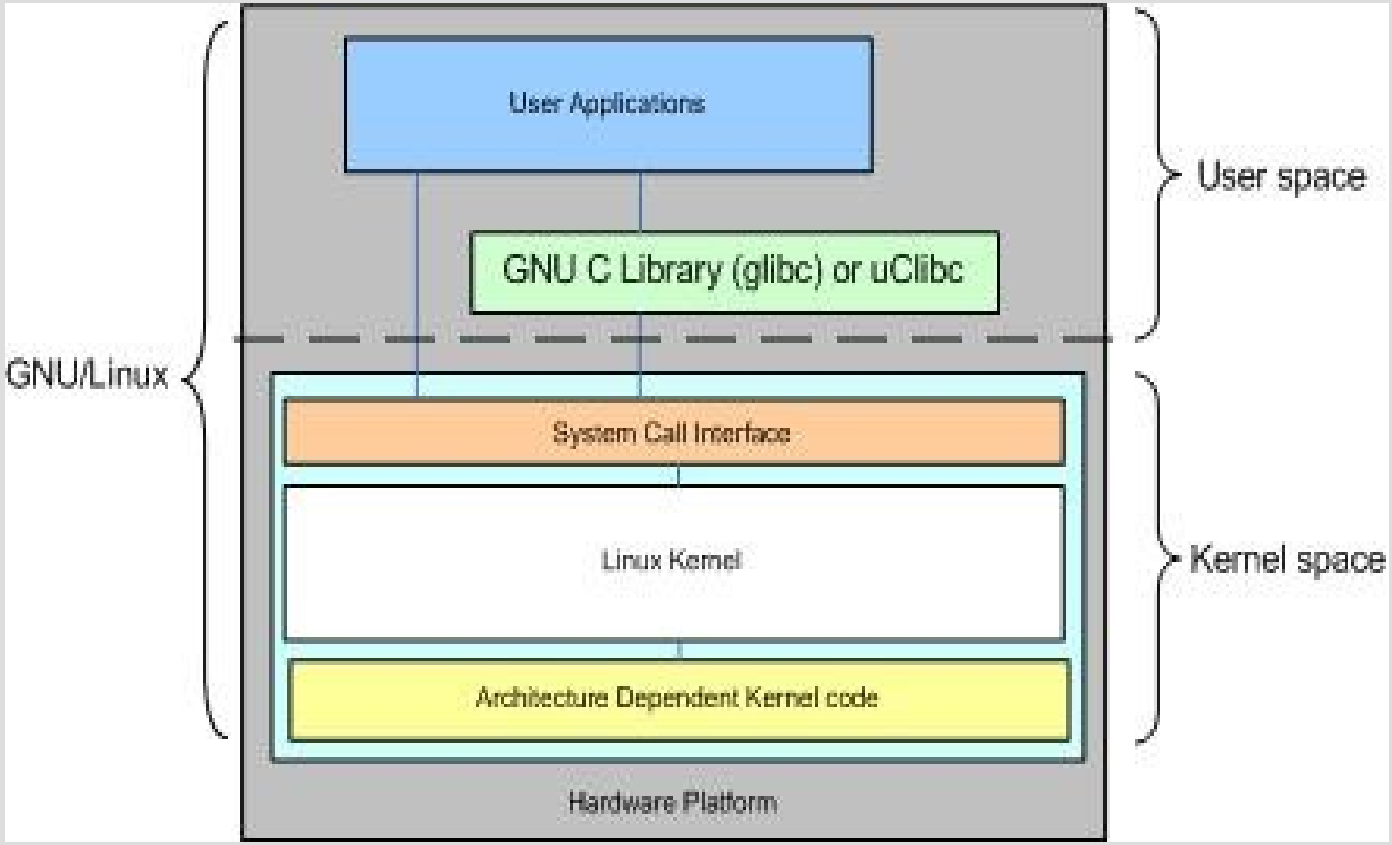
Prehistoria Linux

Release Series	Date of Initial Release	Number of Releases	Time to Final Release in Series	Duration of Series
0.01	9/17/91	2	2 mos.	2 mos.
0.1	12/3/91	85	27 mos.	27 mos.
1.0	3/13/94	9	1 mo.	12 mos.
1.1	4/6/94	96	11 mos.	11 mos.
1.2	3/7/95	13	6 mos.	14 mos.
1.3	6/12/95	115	12 mos.	12 mos.
2.0	6/9/96	34	24 mos.	32 mos.
2.1	9/30/96	141	29 mos.	29 mos.
2.2	1/26/99	14	9 mos.	--
2.3	5/11/99	60	12 mos.	--

Y Linux es...


ACCESS Linux Platform Launcher
Java Apps
Garnet Apps
GTK+ Apps
Telephony & Message
Home Screen (NFDM)
SMS, MMS, IM
Entertainment Suite (Audio,Video, Photo Players)
NetFront Browser (SVG,SMIL)
PIM Applications
3rd-party Native Apps
SDK Tools
Legacy Applications
Native ACCESS Platform Aplications
Applications
Launchpads(Native,Garnet,Java)
ACCESS Linux Platform UI & Core Application Services Frameworks
GTK+
Application Server
Bundle Manager
Attention / Alarm / Notification Managers
HotSync
Exchange Manager Plug-ins
Mobile Services
Messaging Framework
Bluetooth Services
Custom Widgets
Garnet VM (Garnet OS) Emulator
Java VM, JSRs
Multi-media Services
OTA Device Mgmt
OTA Data Sync
SyncML
Exchange Manager
Gstreamer
DRM
Global Settings
Telephony
Connection Manager
BlueZ
X Windowing Sysytem
Security Policy Framework
SQLite
H / W Acceleration
OpenSSL
User Space
ACCESS Linux Platform Security Module
Service / Process Management
Power Management
Memory Management
Graphics & Display Drivers
Input Driver
File System
Multimedia Drivers
Networking Stack
Networking Drivers
Bluetooth Driver
Kernel
CPU
RAM
LCD
Input Device
NAND
Multimedia Hardware
GSM / GPRS / UMTS / EDGE, WiFi, Bluetooth,Technology, etc.
Hardware
Open Source Software
Hiker Framework
ACCESS Proprietary

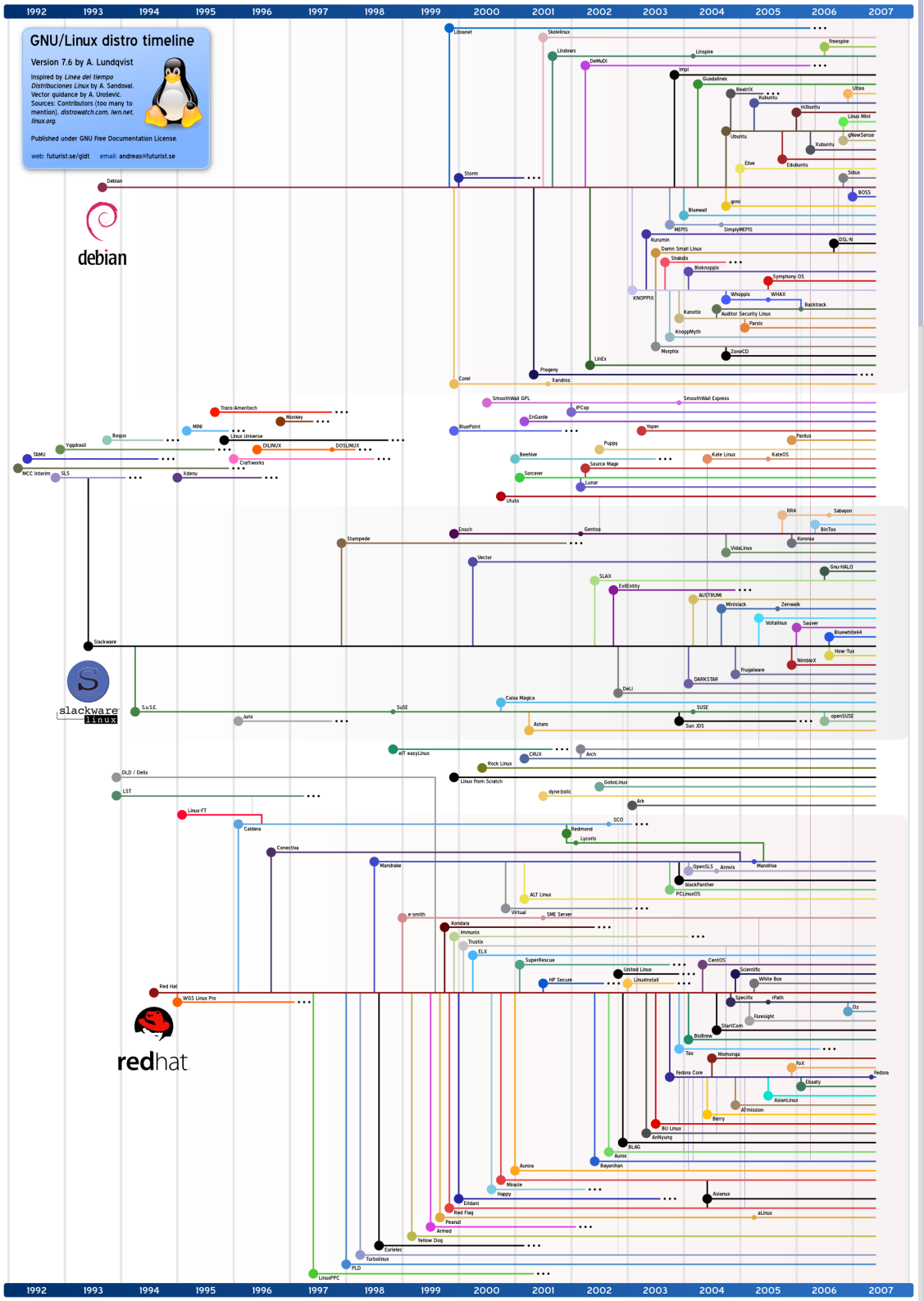
Arquitectura Básica



¿Qué es una distribución?

- Selección de paquetes que acompañan a una cierta versión del núcleo linux
- Un sistema de paquetes (rpm, deb, etc.)
- Unos repositorios de paquetes
- Unas políticas de mantenimiento y actualización.
- Unas plataformas hw soportadas
- OBJETIVO: facilidad de instalación y mantenimiento

Distribuciones



Compilación en Unix

- Editores
- Compiladores
- Utilidades
- Entornos

Editores

- Empecemos por lo básico: vi
- Está presente en todos los unix (o variantes)
- Es independiente del terminal
- Existente en unix desde 1983 (Unix System V)
- Se controla exclusivamente con el teclado
- Pero... Hay que aprender los comandos

vi

- Modo Texto 

- Modo Comando

i Insertar antes del cursor.
I Insertar al principio de la línea.
a Añadir después del cursor.
A Añadir al final de la línea.
o Abrir línea debajo de la actual.
O Abrir línea encima de la actual.
R Sobreescribir (cambiar) texto.
r Sobreescribir el carácter sobre el que está el cursor.
c Reemplaza caracteres.
cw Reemplaza palabras.
C, c$ Reemplaza hasta el fin de línea.
c0 Reemplaza desde el comienzo de línea.

flechas Mover en distintas direcciones.
h ó BS Una posición hacia la izquierda.
l ó SP Una posición hacia la derecha.
k ó - Una línea hacia arriba.
j ó + Una línea hacia abajo.
$ Fin de línea.
0 (Cero) Principio de línea.
1G Comienzo del archivo.
G Fin del archivo.
18G Línea número 18.
Ctrl-G Mostrar número de línea actual.
w Comienzo de la palabra siguiente.
e Fin de la palabra siguiente.
E Fin de la palabra siguiente antes de espacio.
b Principio de la palabra anterior.
^ Primera palabra de la línea.
% Hasta el paréntesis que aparea.
H Parte superior de la pantalla.
L Parte inferior de la pantalla.
M Al medio de la pantalla.
23| Cursor a la columna 23.
Ctrl-f Una pantalla adelante.
Ctrl-b Una pantalla atrás.
Ctrl-l Refrescar la pantalla.
Ctrl-d Media pantalla adelante.
Ctrl-u Media pantalla atrás.

vi

- Modo EX

```
:q        Salir si no hubo cambios.
:q!       Salir sin guardar cambios.
:w        Guardar cambios.
:w arch1  Guardar cambios en archivo arch1.
:wq       Guardar cambios y salir.
:r arch2  Insertar un archivo.
:e arch2  Editar un nuevo archivo.
:e! arch2 Idem sin salvar anterior.
:r! comdo Insertar salida de comando.
:shell    Salir al shell (vuelve con exit).
:.=       Muestra el número de línea en que
se halla en cursor.
```

- Mover

```
:1   Mueve a línea 1.
:15  Mueve a línea 15.
:$   Mueve a última línea.
```

- Busqueda

```
/str Buscar hacia adelante cadena 'str'.
?str Buscar hacia atrás cadena de 'str'.
n    Buscar siguiente (si se usó /) o anterior (si
se usó ?).
N    Buscar anterior (si se usó /) o siguiente (si
se usó ?).
fc   Buscar el siguiente carácter 'c' en la línea.
Fc   Buscar el anterior carácter 'c' en la línea.
tc   Ir al carácter anterior al siguiente 'c'.
Tc   Ir al carácter posterior al precedente 'c'.
;    Repetir el último comando f, F, t, o T.
,    último comando f, F, t, o T en sentido
inverso.
```

gcc

- GNU Compiler collection
- Front ends para distintos lenguajes:
  - Ada
  - C
  - C++
  - Fortran
  - Java
  - Objective-c / c++

A compilar...

- Crear un programa en c hola.c que saque por pantalla la famosa: “hola mundo” (usando el vi, p.ej.)
- Ejecutar el comando de compilación
  - `gcc hola.c`
- Ejecutar el resultado
  - `./a.out`
- Si queremos que nos lo saque con otro nombre:
  - `gcc hola.c -o hola`

Ejemplo 2

- Escribir un programa en c que calcule y nos imprima la raiz cuadrada de 25. Llamarlo hello2.c
- Compilarlo...
- ...
- ¿Que ha fallado?
- Probar `gcc -lm hello2.c`

Las librerías

- En linux las librerías dinámicas se encuentran en /usr/lib, /lib
- Se pueden especificar otras librerías por su camino completo
- Se puede definir dónde encontrar las librerías con LDD_LIBRARY_PATH
- Comando ldd para conocer las librerías de las que depende

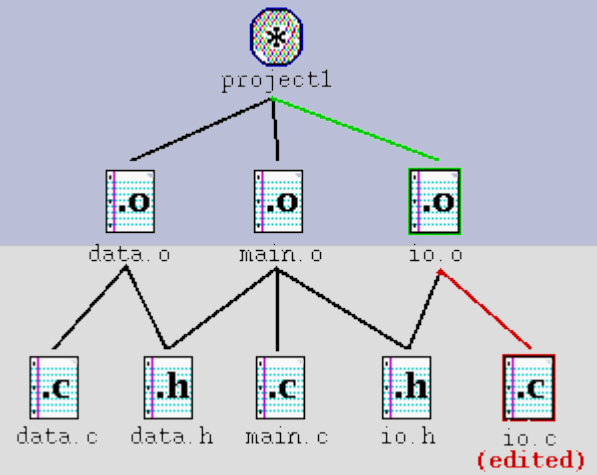
El ciclo de compilación

- Preprocesado
  - gcc -E archivo.c
- Compilado
  - gcc -c archivo.c
- Ensamblado
  - gcc -S archivo.c
- Linkado
  - gcc

Proyectos compuestos

- Dependencias
- Make
- IMake
- Autoconf

Dependencias



- Un proyecto puede tener múltiples archivos, con dependencias de archivos de cabecera, librerías o archivos de configuración
- Es altamente ineficiente recompilar todos los archivos cada vez que modificamos alguno
- Para evitarlo existen las herramientas como make

make

- El comando make es un sistema para construir codigo compilado, aunque se puede utilizar para más cosas
- make necesita un archivo de configuracion

```
Makefile
```

- Se puede cambiar el archivo con la opción -f
- Un archivo Makefile se compone de reglas

```
objetivo: prerequisitos
comandos
```

Ejemplo Makefile

```
# Lincado
sample: main.o example.o
	cc -o sample main.o example.o
	echo sample: make complete

# Compilar las fuentes
main.o: main.c main.h
	cc -c main.c
example.o: example.c defs.h
	cc -c example.c
```

Distintos objetivos

```
# Objetivo principal
.PHONY: clean install

# Eliminar ejecutable y objetos
clean:
	rm sample main.o example.o
	echo clean: make complete

# Instalar el producto final
install:
	cp sample /usr/local
	echo install: make complete
```

Uso de variables

```
# archivos objeto
objects = main.o example.o

# Lincar objetos
sample: $(objects)
	cc -o sample $(objects)
	echo sample: make complete

# Compilar fuentes
main.o: main.c main.h
	cc -c main.c
example.o: example.c defs.h
	cc -c example.c

# Eliminar ejecutables y objetos
clean:
	rm sample $(objects)
	echo clean: make complete
```

Toques finales

```
# Compilador:
CC=gcc

objects = main.o example.o

# Regla por defecto
all:   sample
       echo all: complatada a compilacion

sample: $(objects)
   $(CC) -o $@ $+

# Regla general para compilar los .c a .o:
%.o:%.c
   $(CC) -c $+

#Nos aseguramos de recompilar si cambian las cabeceras:
main.o: main.h defs.h
example.o: example.h defs.h
```

Vuestro turno

- Crear un programa que use 2 archivos fuente y 2 archivos de cabecera
- Crear un archivo makefile para compilarlos
- Ejecutar el makefile varias veces (make) modificando los archivos de uno en uno para ver su efecto.

Más problemas

- Con make se pueden resolver dependencias dentro de una plataforma
- ¿Cómo se resuelven dependencias para compilar en varias plataformas?
- Característica de UNIX : portable
- Primer gran proyecto en experimentar el problema: X-Windows

X-Windows

- A diferencia de ms-windows, el sistema gráfico de UNIX no es parte del núcleo
- Utiliza un sistema cliente-servidor y el paradigma de DISPLAY
- Las pantallas se pueden compartir para que los clientes ejecuten aplicaciones en ellas.
- Los clientes y servidores son independientes de la plataforma y compatibles.

X e Imake

- X window requería su compilación en múltiples plataformas
- La complejidad del código y su extensión hacían muy dificil mantener makefiles para cada combinación de plataforma-hardware
- Se desarrolla imake para permitir compilar el código de X-Window de la misma manera en todas las plataformas

Funcionamiento Imake

- Ejemplo... ver fuentes

GNU Build System

- Desarrollado casi al tiempo que imake
- Pretende descentralizar el desarrollo de configuraciones para distintas plataformas
- Conocido por autotools
- Comprende:
  - Autoconf
  - Automake
  - Libtool

Autoconf

- Procesa los archivos configure.in o configure.ac
- Genera un script llamado configure
- Está diseñado para ntentar salvar las diferencias que existen entre distintos sabores de Unix
- Autoconf incluye la herramienta Autoheader que se usa para manejar los archivos de cabecera de C.

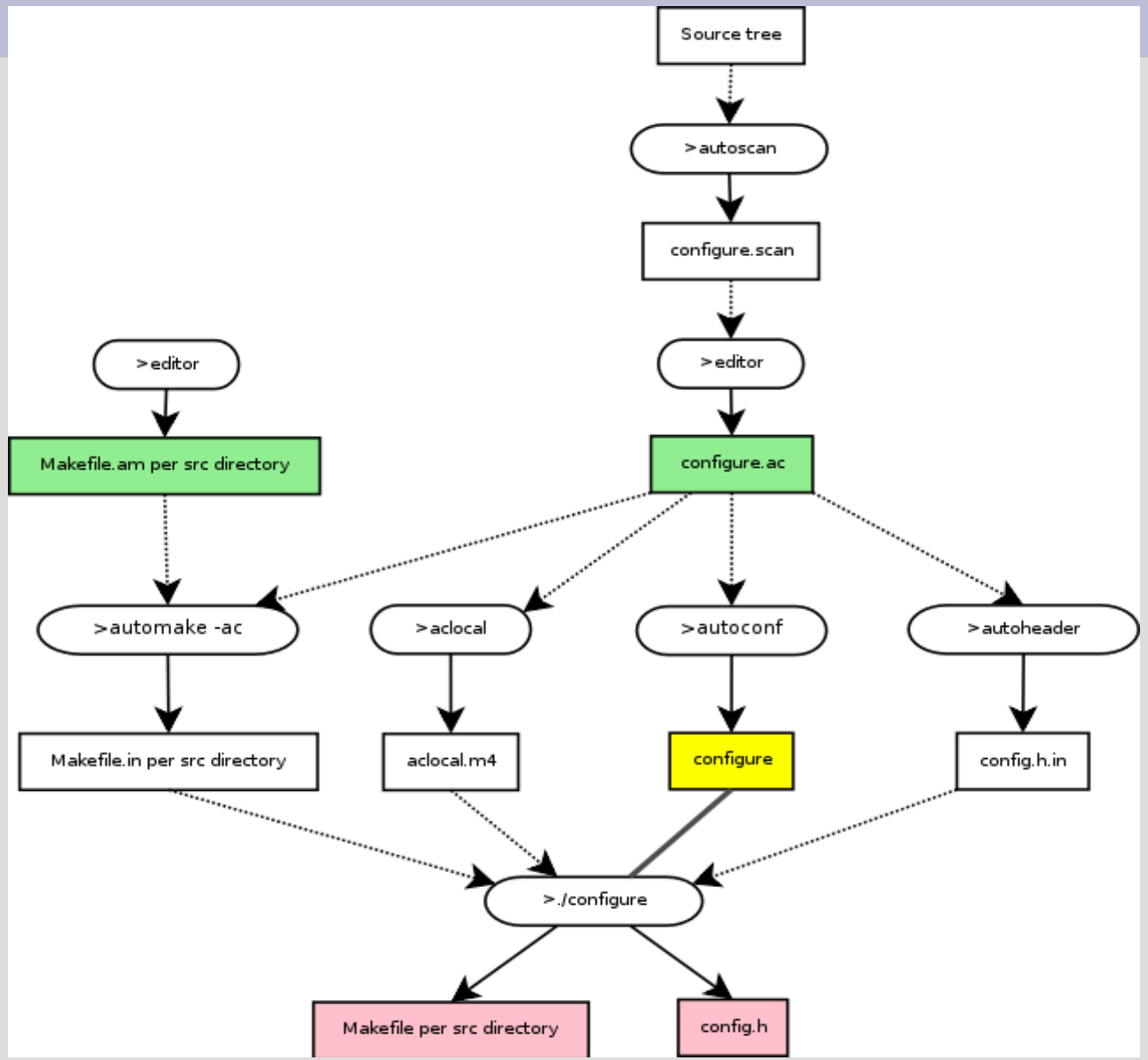
Automake

- ayuda a crear archivos Makefile portables
- Toma como entrada un archivo Makefile.am y lo transforma en un Makefile.in.
- Este, a su vez, es utilizado por Autoconf para generar el archivo Makefile final.

Libtool

- Libtool ayuda a crear librerías estáticas y dinámicas para varios Sistemas Operativos Unix.
- Libtool abstrae el proceso de creación de las librerías ocultando las diferencias entre los distintos sistemas (entre GNU/Linux y Solaris por ejemplo).

Proceso


Source tree
>autoscan
configure.scan
>editor
>editor
Makefile.am per src directory
configure.ac
>automake -ac
>aclocal
>autoconf
>autoheader
Makefile.in per src directory
aclocal.m4
configure
config.h.in
>./configure
Makefile per src directory
config.h

Ejemplo – Autoconf I

- Creamos un directorio para nuestras fuentes con esta estructura:
- hello
  - src
    - hello.c
    - hello.h

Ejemplo – Autoconf II

- Creamos en hello el archivo
  - `Makefile.am`
- Contenido:
  - `SUBDIRS=src`
- Esto le indica los subdirectorios a los que ha de bajar

Ejemplo – Autoconf III

- En el directorio src hay que incluir un Makefile.am que indique:
  - Que programas construir.
  - El directorio de instalación para “make install”.
  - Que archivos fuente se necesitan para cada programa

```
helloprgdir=../
helloprg_PROGRAMS=hello
hello_SOURCES=hello.c
```

Ejemplo – Autoconf IV

- Desde el directorio hello ejecutar:
  - `autoscan`
- Generará un archivo
  - `configure.scan`
- Devolverá un error (no preocuparse)
- Renombrar `configure.scan` a `configure.ac`
- Editar el archivo `configure.ac`

Ejemplo – Autoconf V

- Cambiamos estos valores:
- `AC_PREREQ(2.50)`
- `AC_INIT(hello, 1.0, jose@espinosa.nom.es)`
- `AM_INIT_AUTOMAKE`
- `AM_CONFIG_HEADER(config.h)`
- `AC_CONFIG_FILES([Makefile src/Makefile])`

- Aquí será donde incluyamos dependencias de librerías y demás (ver fuente)

Ejemplo – Autoconf VI

- Ejecutar `aclocal`
  - Genera el archivo : `aclocal.m4`
- Ejecutar `automake -a -c`
  - Convierte los `Makefile.ac` en `Makefile.in`
  - Añade todos los demás archivos necesarios
- Revisar los archivos generados
  - Incluye `INSTALL`, `COPYING`, ..

Ejemplo – Autoconf VII

- Ejecutar
  - `autoheader`
- Genera un `config.h`
- Finalmente, ejecutar:
  - `autoconf`
- Se creará el script `configure`

Al final...

- `./configure`
- `make`
- `make install`

Ejercicios

- Leer el manual de autoconf
- Incluir una librería de dependencia (libm por ejemplo)
- PISTA: AC_CHECK_LIB o AC_SEARCH_LIB

Entornos de programación

- Kdevelop (C/C++ QT/KDE)
- Anjuta (C/C++ GTK/GNOME)
- Eclipse CDT (Java, C/C++, ...)
- Code Dragon (wxWidgets)
- Geany (gtk, simple pero efectivo)

Referencias

- http://www.firstmonday.dk/issues/issue5_11/moon/index.html
- http://es.wikipedia.org/wiki/Unix
- http://club.telepolis.com/jagar1/Unix/Vi.htm
- http://gcc.gnu.org/
- http://www.developingprogrammers.com/index.php/2006/01/05/au