



Contenido

- Introducción
- El concepto general
- Dependencias
- Estructura del Makefile y reglas
- Variables
- Condiciones
- Reglas implícitas

Introducción: MAKE

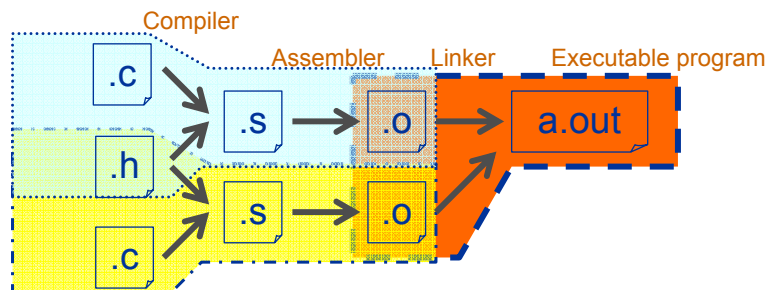
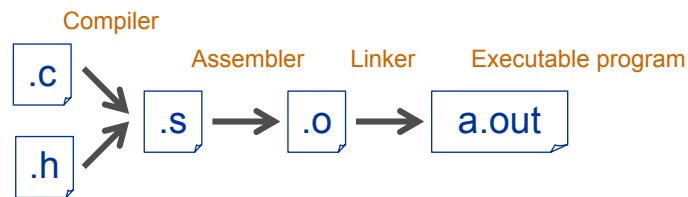
- La herramienta **make** da soporte al desarrollo de programas
- Típicamente, sólo se cambia una pequeña cantidad de datos cuando se llama al compilador
- La herramienta **make** tiene en cuenta qué porciones del programa entero se han cambiado
- **Make** sólo compila esas partes
- **Make** tiene un potente grafo de dependencias que se deriva de reglas

Makefile
Computer Architecture Laboratory 2005

© Departamento de Ingeniería Telemática, UC3M

3

El concepto general



Makefile
Computer Architecture Laboratory 2005

© Departamento de Ingeniería Telemática, UC3M

4

Separación

Compiler
Assembler Linker Executable program

Primero .c → .s → .o → a.out
.h → .s → .o → a.out

Segundo .h → .s → .o → a.out
.c → .s → .o → a.out

- Se necesitan dependencias
 - “.h” hace falta tanto para el primero como para el segundo fichero.c

Makefile Computer Architecture Laboratory 2005 © Departamento de Ingeniería Telemática, UC3M 5

Dependencias en el fichero MAKEFILE

```

#
# First example of a Makefile
#
project: data.o main.o io.o
    gcc data.o main.o io.o -o project

data.o: data.c data.h
    gcc -c data.c

main.o: data.h io.h main.c
    cc -c main.c

io.o: io.h io.c
    gcc -c io.c
    
```

Makefile Computer Architecture Laboratory 2005 © Departamento de Ingeniería Telemática, UC3M 6

Dependencia: Edición de data.c

```

project: data.o main.o io.o
gcc data.o main.o io.o -o project
data.o: data.c data.h
gcc -c data.c
main.o: data.h io.h main.c
cc -c main.c
io.o: io.h io.c
gcc -c io.c
    
```

Makefile
Computer Architecture Laboratory 2005

© Departamento de Ingeniería Telemática, UC3M

7

Ejemplo: Dependencias

```

graph TD
    main_c[main.c] --> main_o[main.o]
    define_h[define.h] --> main_o
    library_c[library.c] --> library_o[library.o]
    main_o --> program[program]
    library_o --> program
    
```

Makefile
Computer Architecture Laboratory 2005

© Departamento de Ingeniería Telemática, UC3M

8

Estructura de un Makefile

Declaración de variables	CC = /usr/bin/gcc OBJECTS = main.o library.o
Reglas explícitas	program : main.o library.o \$(CC) -c main.o library.o
Reglas implícitas	library.o : library.c main.o : main.c define.h \$(CC) -c main.c echo "main.o compiling" clean : rm -rf \$(OBJECTS)
Targets Built-in Especiales	.PHONY : clean

Makefile
Computer Architecture Laboratory 2005
© Departamento de Ingeniería Telemática, UC3M
9

Reglas

Target(s) Separador (:) Dependencias

main.o : main.c define.h

\$(cc) -c main.c

Símbolo tabulador Comandos(s) Shell

Makefile
Computer Architecture Laboratory 2005
© Departamento de Ingeniería Telemática, UC3M
10

Generación del grafo de dependencias

target : ficheros fuente(s)

command (*debe estar precedido por un tabulador*)

- Un target en el **Makefile** es un fichero que debe ser
 - creado o
 - actualizado

cuando cualquiera de sus ficheros dependencia (ficheros fuente) se modifiquen

El(los) comando(s) dado(s) en la(s) línea(s) siguiente(s) (que deben estar precedidas de un carácter tabulador) se ejecutan para crear o actualizar el fichero target

Llamada al Makefile

- Crea un **makefile** en tu directorio
 - El nombre puede ser: "**Makefile**" o "**makefile**"
 - Puedes llamarlo con: "**make**"
 - Si el nombre es diferente:
make -f your_makefile_name

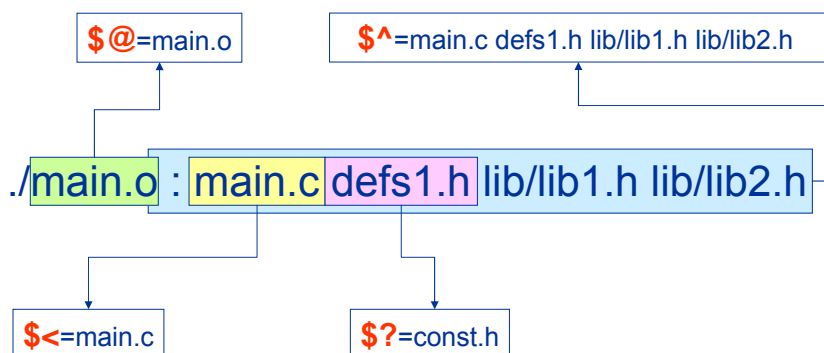
Ejemplo

- Llamada a **make** (del primer ejemplo)

```
varpa:~/Test/Makefile_Dir> make
gcc -c data.c
cc -c main.c
gcc -c io.c
gcc data.o main.o io.o -o project
varpa:~/Test/Makefile_Dir>
```

- **make** comprueba primero los ficheros fuente e intenta crear o actualizar los ficheros objeto
- Por lo tanto *data.o*, *main.o* y *io.o* se crean antes que el fichero target *project*

Variables automáticas



Sustitución de variables

`<var2> := $(<var1>:<pattern>=<replace>)`

Ex. `sources := a.c b.c c.c`
`objects := $(sources:.c=.o)`

`objects = a.o b.o c.o`

Macros para reducir Makefiles

```
OBJECTS = data.o main.o io.o
project1: $(OBJECTS)
    cc $(OBJECTS) -o project1

data.o: data.c data.h
    cc -c data.c

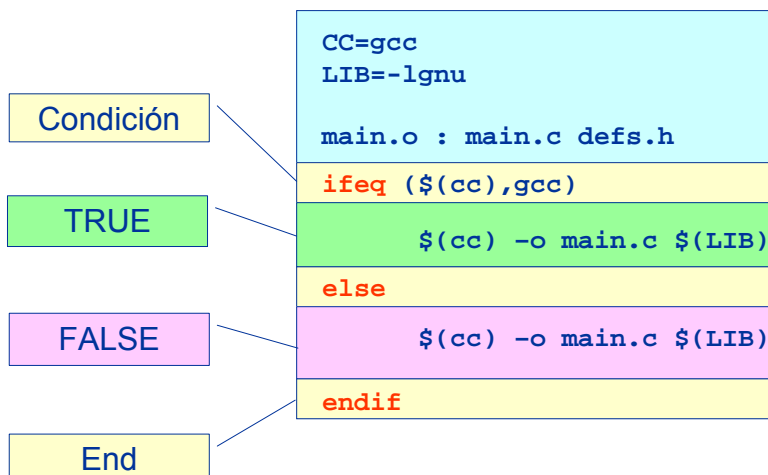
main.o: data.h io.h main.c
    cc -c main.c

io.o: io.h io.c
    cc -c io.c
```

Macros especiales

- **CC**
 - Contains the current C compiler. Defaults to cc.
- **CFLAGS**
 - Special options which are added to the built-in C rule. (See next page.)
- **\$@**
 - Full name of the current target.
- **\$?**
 - A list of files for current dependency which are out-of-date.
- **\$<**
 - The source file of the current (single) dependency.
- **\$^**
 - all prerequisites of a rule with blank symbols between listed items

Condiciones



Reglas implícitas

file.suffix1 : file.suffix2

Regla	Comando
file.o : file.c	\$(CC) -c \$(CPPFLAGS) \$(CFLAGS)
file.o : file.cc	\$(CXX) -c \$(CPPFLAGS) \$(CXXFLAGS)
file.o : file.p	\$(PC) -c \$(PFLAGS)
file : file.o	\$(CC) \$(LDFLAGS) file.o \$(LOADLIBES)
file.dvi : file.tex	\$(TEX)

Más reducciones de *Makefile*

```
OBJECTS = data.o main.o io.o
CC=/usr/bin/gcc
project: $(OBJECTS)
    $(CC) $(OBJECTS) -o project
data.o: data.h
main.o: data.h io.h
io.o: io.h
```

```
# Former Makefile
OBJECTS = data.o main.o io.o
project1: $(OBJECTS)
    cc $(OBJECTS) -o project1
data.o: data.c data.h
    cc -c data.c
main.o: data.h io.h main.c
    cc -c main.c
io.o: io.h io.c
    cc -c io.c
```

- Salida

```
varpa:~/Test/Makefile_Dir> make
/usr/bin/gcc -c -o data.o data.c
/usr/bin/gcc -c -o main.o main.c
/usr/bin/gcc -c -o io.o io.c
/usr/bin/gcc data.o main.o io.o -o project1
varpa:~/Test/Makefile_Dir>
```

Targets predefinidos

- **.PHONY : clean**
declara que "clean" no es un fichero
- **.DEFAULT**
los comandos en DEFAULT se ejecutarán para todos los targets si no tienen sus propias reglas y si no están definidos como targets
- **.SILENT**
Si se especifican prerequisites para .SILENT, entonces make no imprimirá los comandos para rehacer esos ficheros concretos antes de ejecutarlos

Conclusión

- ¡Suerte con los ejercicios!