



SEMINARIO-TALLER DE SOFTWARE (STI-S)

Herramienta Make

Índice

- 1. Introducción.**
- 2. Modularidad.**
- 3. Archivos de Cabecera.**
- 4. Programa ejemplo**
- 5. Reglas**
- 6. Uso de variables**
- 7. Reglas predefinidas**
- 8. Opciones**

Introducción

- Es una ***herramienta*** que se utiliza para tratar, de forma ***optimizada y automática***, los diversos archivos de programas que integran un proyecto de software.
- Las reglas para realizar el tratamiento o actualización se escriben en un ***archivo*** de texto llamado usualmente **makefile**.
- Se invoca mediante el ***comando make***.
 - Ejecuta las reglas del archivo makefile recompilando sólo las partes que han sido modificadas desde la última compilación, y enlaza los módulos en código objeto construyendo el ejecutable.
- Puede usarse con cualquier lenguaje de programación cuyas tareas puedan lanzarse mediante comandos.
- Acepta parámetros para realizar sólo ciertas tareas (las reglas para cada tarea deben estar escritas en el makefile).
 - Eliminar archivos intermedios, compilar sólo un módulo, etc...

Modularidad

- Normalmente, a la hora de programar no creamos un único archivo `.c`, sino varios de ellos conteniendo diferentes funciones del programa por módulos. Esto nos proporciona una mejor **organización** del código, una mejor **modularidad** y, sobre todo, más **facilidad** (y velocidad) a la hora de compilar.
- Además tenemos los **archivos de cabecera .h**, que utilizamos para definir parámetros o funciones que vamos a utilizar en el código.

```
/* calculadora calc.h */  
#define NUMERO 0  
void push(double);    /* extrae de la pila */  
double pop(void);    /* coloca en la pila */  
int gettop(char[]);  
int getch(void);
```

- Las funciones o sus cabeceras deben definirse siempre antes de usarse.

Archivos de cabecera

- Permiten modularizar el código y favorecer la ocultación de información.
- Puede contener:
 - Declaraciones de funciones: *void nombre_funcion(void);*
 - Otras directivas de inclusión: *#include "a.h"*
 - Comentarios
 - Definiciones de tipos de dato (*typedef*)
 - Constantes
- Nunca debe tener:
 - Definiciones de variables: *int dia;*
 - Definiciones de funciones: *void nombre_funcion(void){...};*

Programa ejemplo

- Si tenemos un programa grande con el código repartido en varios archivos de código fuente (extensión `.c`) y archivos de cabecera para las declaraciones (extensión `.h`):
 - [calc.h](#)
 - [getch.c](#)
 - [getop.c](#)
 - [main.c](#)
 - [stack.c](#)
- La compilación de este programa puede hacerse con el comando:

```
$ gcc -o polaca main.c getch.c getop.c stack.c  
$ ./polaca
```
- Cada vez que se modifique alguno de los archivos que integran el programa, este comando recompila todo, haya cambiado o no. En un programa grande, esto es un inconveniente.

Compilación con Make

- Crear un archivo **makefile** con el siguiente contenido:

```
# makefile para calculadora polaca, versión 1
# usar tabulador (no espacios) en la línea de comando
polaca : main.o stack.o getop.o getch.o
    gcc -o polaca main.o stack.o getop.o getch.o
main.o: main.c calc.h
    gcc -c main.c
stack.o: stack.c calc.h
    gcc -c stack.c
getop.o: getop.c calc.h
    gcc -c getop.c
getch.o: getch.c
    gcc -c getch.c
clean:
    rm polaca \
        main.o stack.o getop.o getch.o
```

- Los renglones iniciados con **#** son **comentarios** y el resto de las líneas son **reglas**. Cada regla empieza con un nombre seguido de ":",“.

Reglas del Make

- Las reglas tienen el siguiente formato:

destino : requisito ...

comando

...

main.o : main.c calc.h

gcc -c main.c

- **Destino:** nombre de un archivo a crear, un ejecutable o un archivo objeto (.o). También puede ser el nombre de una tarea realizar (por ejemplo es usual usar "**clean**" como indicativo de la regla que ejecuta los comandos necesarios para eliminar archivos objeto y recomenzar una compilación desde cero).
- **Requisito:** nombre de un archivo del cual depende el destino a crear. Un destino suele depender de varios archivos requisito. Cuando el destino es el nombre de una tarea no hay requisitos.
- **Comando:** acción a realizar. La creación de un destino puede requerir varios comandos.

Variables

- Una variable es un nombre simbólico que será substituido por su valor en el momento de la aplicación de las reglas posteriores. Para definir una variable se utiliza la siguiente sintaxis:

<IDENTIFICADOR>=valor

OBJECTS = stack.o getop.o getch.o (Ejemplo)

- Make define algunas variables que se pueden utilizar y que tienen valores por defecto:

AR: Programa de empaquetado. Por defecto es ar.

CC: Compilador de C. Por defecto es cc.

CXX: Compilador de C++. Por defecto es g++.

CPP: Preprocesador de C.

CFLAGS: Opciones de compilación C.

CXXFLAGS: Opciones de compilación C++.

CPPFLAGS: Opciones para el preprocesador.

LDLIBS: Opciones para el montador (enlazador).

LDLIBS: Librerías a utilizar en el montaje.

Uso de variables

```
# makefile para calculadora polaca, versión 2
# uso de variables
# usar tabulador (no espacios) en la línea de comando
CC = gcc
OBJECTS = stack.o getop.o getch.o
polaca : main.o $(OBJECTS)
    $(CC) -o polaca main.o $(OBJECTS)
main.o: main.c calc.h
    $(CC) -c main.c
stack.o: stack.c calc.h
    $(CC) -c stack.c
getop.o: getop.c calc.h
    $(CC) -c getop.c
getch.o: getch.c
    $(CC) -c getch.c
top.o getch.o
clean:
    rm polaca \
        main.o stack.o getop.o getch.o
```

- En esta versión resulta fácil cambiar el compilador, o agregar un nuevo nombre de archivo objeto.

Reglas predefinidas

- Se disponen de algunas reglas predefinidas. Por ejemplo el **comando para compilar objetos en C**. Esto hace innecesario escribir los comandos en las reglas que crean los objetos:

```
# makefile para calculadora polaca, versión 3
# uso de variables; reglas predefinidas
CC = gcc
OBJECTS = stack.o getop.o getch.o
polaca : main.o $(OBJECTS)
    $(CC) -o polaca main.o $(OBJECTS)
main.o: calc.h
stack.o: calc.h
getop.o: calc.h
getch.o:
clean:
    rm polaca \
        main.o stack.o getop.o getch.o
```

- Por ejemplo make entenderá que si necesita un fichero **stack.o** y tiene un fichero **stack.c debe compilarlo**. De esta forma podemos simplificar el fichero makefile.

Opciones

- Acepta varios parámetros por línea de comando relativos a su comportamiento y la visualización de información durante la ejecución:

-n	Imprime las órdenes que <i>make</i> efectúa pero sin ejecutarlas.
-f <fichero>	Indica a <i>make</i> que debe utilizar <Fichero> como si fuera un <i>makefile</i> aunque tenga otro nombre.
-p	Imprime las definiciones de macros y las reglas implícitas
-C dir	Cambia al directorio y dir procesa el <i>makefile</i> que allí se encuentre.