Herramientas Computacionales para la Astroinformática

Cristian A. Vega-Martínez (oficina: IMIP, Académicos 2)
Facundo A. Gómez

Compilación con GNU Make

Automatización de compilación y enlazado

GNU Make

https://www.gnu.org/software/make/

Make es una herramienta para gestionar **dependencias** (internas) para compilación.

- Se utiliza para automatizar el proceso de compilación que involucra múltiples archivos (dependencias), la limpieza de los temporales, y la instalación del software final.
- Puede identificar qué partes de un programa requieren ser recompiladas. Ejecuta los comandos necesarios para ello.
- Funciona en base a la definición de **reglas** en un archivo **Makefile**.

Desde la terminal se ejecuta **`make`** en el directorio a compilar.

Makefile

Al ejecutarse, **`make`** busca en el directorio un archivo de texto llamado Makefile, donde se escriben reglas del estilo:

```
objetivos ... : dependencias ...
    recetas
    ...
```

Esto debe ser un tab!

- ***objetivo*** es el nombre del archivo a crear (usualmente código objeto o ejecutable), o también puede ser el nombre de una acción a definir (e.g. *clean*);
- ***dependencias*** son los archivos necesarios para crear el objetivo; y
- ***receta*** contiene los comandos a ejecutar para ejecutar la regla.

Ejemplo de un archivo Makefile

Makefile para compilar un programa llamado mygame, compuesto por varios archivos .c

Este es un comentario

```
# Esta es mi regla principal:
mygame : main.o player.o controles.o \
         sprites.o
     gcc -o mygame main.o player.o \
          controles.o sprites.o

main.o : main.c globals.h
     gcc -c main.c

player.o : player.c globals.h
     gcc -c player.c

controles.o : controles.c
     gcc -c controles.c

sprites.o : sprites.c
     gcc -c sprites.c

clean:
     rm mygame main.o player.o \
          controles.o sprites.o
```

Makefile

Es posible definir variables dentro de los Makefiles para simplificar la vida:

Hay que dejar espacios acá

```
CC = gcc -O2

programa : programa.c
	$(CC) -o programa programa.c
```

Incluso se pueden agrupar archivo en una variable:

```
objetos = main.o player.o controles.o sprites.o

mygame : $(objetos)
	$(CC) -o mygame $(objetos)
```

Estas son reemplazadas de forma **textual** por make.
También se puede usar **+=** para agregar texto a variables definidas.

¿Cómo funciona?

- Al ejecutar `make`, la primera regla del archivo Makefile es marcada como el objetivo principal (variable: .DEFAULT_GOAL)
- Para cada regla se verifica si sus *dependencias* tienen reglas propias, en cuyo caso `make` las aplicará.
- Estas reglas se ejecutarán (recetas/comandos) **solamente** si algún archivo de sus *dependencias* es más reciente que el *objetivo* (i.e. archivo modificado), o si el *objetivo* no existe.
  Esto último se puede desactivar con la variable .PHONY. Por ejemplo, la regla clean:

```
.PHONY : clean

clean :
	-rm mygame $(objects)
```

Este - permite omitir los errores de rm

Más características

Reglas implícitas: `make` es capaz de deducir las recetas de compilación de algunos lenguajes (como C, C++, Fortran, tex). Por ejemplo:

```
programa : programa.o
programa.o : programa.c
```

Esta también se puede omitir, make incluye una regla implícita para todo .o

Estas reglas se basan en reglas *patrones*, que también se pueden sobre-escribir. Por ejemplo:

```
%.o : %.c
	$(CC) $(CFLAGS) $< -c $@
```

primera dependencia

objetivo

compilará cualquier .c encontrado y generará su respectivo .o. Para esto, se incluyen variables automáticas en make como `$<`, `$^`, `$@`, etc.

Directrices: `make` también incluye instrucciones de lectura del Makefile, tales como condicionales, inclusión de directorios, inclusión de Makefiles, etc.