Introducción a GNU Make

Algoritmos y Estructuras de Datos II

Makefile

- Es un archivo de texto, generalmente llamado `Makefile`
- Se invoca desde la consola con el comando `make`.
- Útil para la organización del proceso de compilación.
- Sirve también para automatizar otras tareas de mantenimiento.

Un makefile básico

```
dictionary : helpers.o index.o data.o pair.o list.o dict.o main.c
        gcc -o dictionary  \
        helpers.o index.o data.o pair.o list.o dict.o main.c

helpers.o : helpers.c helpers.h dict.h list.h data.h index.h
        gcc -c helpers.c -o helpers.o

index.o   : index.c index.h
        gcc -c index.c -o index.o

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

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

list.o    : list.c list.h data.h index.h pair.h
        gcc -c list.c -o list.o

dict.o    : dict.c dict.h data.h index.h list.h
        gcc -c dict.c -o dict.o
```

Usando variables

```
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
OBJECTS = helpers.o index.o data.o pair.o list.o dict.o

dictionary : $(OBJECTS)
        $(CC) $(CFLAGS) -o dictionary $(OBJECTS) main.c

helpers.o : helpers.c helpers.h dict.h list.h data.h index.h
        $(CC) $(CFLAGS) -c helpers.c -o helpers.o

index.o   : index.c index.h
        $(CC) $(CFLAGS) -c index.c -o index.o

data.o    : data.c data.h
        $(CC) $(CFLAGS) -c data.c -o data.o

pair.o    : pair.c pair.h data.h index.h
        $(CC) $(CFLAGS) -c pair.c -o pair.o

list.o    : list.c list.h data.h index.h pair.h
        $(CC) $(CFLAGS) -c list.c -o list.o

dict.o    : dict.c dict.h data.h index.h list.h
        $(CC) $(CFLAGS) -c dict.c -o dict.o

.PHONY : clean

clean  :
        rm -f dictionary $(OBJECTS)
```

Usando funciones

```
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)

dictionary : $(OBJECTS)
        $(CC) $(CFLAGS) -o dictionary $(OBJECTS)

helpers.o : helpers.c helpers.h dict.h list.h data.h index.h
        $(CC) $(CFLAGS) -c helpers.c -o helpers.o

index.o   : index.c index.h
        $(CC) $(CFLAGS) -c index.c -o index.o

data.o    : data.c data.h
        $(CC) $(CFLAGS) -c data.c -o data.o

pair.o    : pair.c pair.h data.h index.h
        $(CC) $(CFLAGS) -c pair.c -o pair.o

list.o    : list.c list.h data.h index.h pair.h
        $(CC) $(CFLAGS) -c list.c

dict.o    : dict.c dict.h data.h index.h list.h
        $(CC) $(CFLAGS) -c dict.c -o dict.o

.PHONY : clean

clean  :
        rm -f dictionary $(OBJECTS)
```

Usando sustituciones

```
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)

dictionary : $(OBJECTS)
        $(CC) $(CFLAGS) -o dictionary $(OBJECTS)

helpers.o : helpers.c helpers.h dict.h list.h data.h index.h
        $(CC) $(CFLAGS) -c $< -o $@

index.o   : index.c index.h
        $(CC) $(CFLAGS) -c $< -o $@

data.o    : data.c data.h
        $(CC) $(CFLAGS) -c $< -o $@

pair.o    : pair.c pair.h data.h index.h
        $(CC) $(CFLAGS) -c $< -o $@

list.o    : list.c list.h data.h index.h pair.h
        $(CC) $(CFLAGS) -c $< -o $@

dict.o    : dict.c dict.h data.h index.h list.h
        $(CC) $(CFLAGS) -c $< -o $@

.PHONY : clean

clean  :
        rm -f dictionary $(OBJECTS)
```

Reglas implícitas

```
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)

dictionary : $(OBJECTS)
        $(CC) $(CFLAGS) -o dictionary $(OBJECTS)

helpers.o : helpers.c helpers.h dict.h list.h data.h index.h

index.o   : index.c index.h

data.o    : data.c data.h

pair.o    : pair.c pair.h data.h index.h

list.o    : list.c list.h data.h index.h pair.h

dict.o    : dict.c dict.h data.h index.h list.h

.PHONY : clean

clean  :
        rm -f dictionary $(OBJECTS)
```

Reglas genéricas

```
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:.c=.o)

dictionary : $(OBJECTS)
        $(CC) $(CFLAGS) -o dictionary $(OBJECTS)

$(OBJECTS) : %.o : %.c
$(filter-out main.o, $(OBJECTS)) : %.o : %.h

helpers.o : dict.h list.h data.h index.h

pair.o    : data.h index.h

list.o    : data.h index.h pair.h

dict.o    : data.h index.h list.h

.PHONY : clean

clean  :
        rm -f dictionary $(OBJECTS)
```

Dependencias automáticas

```
TARGET = dictionary
CC = gcc
CFLAGS  = -Wall -Werror -Wextra -pedantic -std=c99 -g
SOURCES = $(wildcard *.c)
HEADERS = $(wildcard *.h)
OBJECTS = $(SOURCES:.c=.o)

.all : $(TARGET)

$(TARGET) : $(OBJECTS)
        $(CC) $(CFLAGS) -o $(TARGET) $(OBJECTS)

-include .depend

.depend : $(SOURCES) $(HEADERS)
        $(CC) -MM $(SOURCES) > .depend

clean  :
        rm -f $(TARGET) $(OBJECTS)
```

Resumen

- La estructura de una regla es:

  ```
  objetivo: prerequisitos
    <tab> comando
  ```

- El comando de una regla se ejecuta cuando algún prerequisito ha cambiado o bien el archivo objetivo no existe.
- Una regla no necesariamente está asociada a un archivo. Por ejemplo, la regla `clean` se invoca tipeando `make clean` en la consola. La directiva `.PHONY` indica que la regla no está asociada a un archivo.
- Ejercicio: Definir una regla `check` para ejecutar `valgrind` en el diccionario sin tener que tipear el comando.

Resumen

- Se pueden definir variables, y expandirlas usando $(VAR).
- Hay variables automáticas como "`$<`" que se expande al primer elemento de los prerequisitos y "`$@`" que se expande al nombre del objetivo.
- Hay funciones como `wildcard` y `filter-out` que son útiles para manipular la lista de objetivos o prerequisitos.
- Hay directivas como `include` que es útil para incluir en nuestro makefile un segmento de otro makefile.
- El compilador `gcc` con el flag `-MM` genera reglas para el makefile automágicamente.
- Hay ciertas reglas conocidas que no hace falta escribirlas en el makefile.
- `make` puede hacer muchisisisísmas cosas más: mirar el manual.