Tutorial.

GNU make

El utilitario `make` de GNU es una herramienta para actualizar, en forma optimizada y automática, los diversos archivos de programas que integran un proyecto de software. Las reglas de actualización se escriben en un archivo de texto llamado usualmente `makefile` o `Makefile`. La actualización se invoca dando el comando

```
make
```

que 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.
`make` puede usarse con cualquier lenguaje de programación cuyas tareas puedan lanzarse mediante comandos. En realidad puede usarse para manejar cualquier conjunto de archivos en los cuales haya archivos que deban actualizarse toda vez que cambien ciertos otros.
`make` acepta parámetros para realizar sólo ciertas tareas, como ser eliminar archivos intermedios o compilar sólo un módulo. Las reglas para cada tarea deben estar escritas en el `makefile`.

Nota: en los ejemplos de línea de comando mostramos el indicador de comando $, que no debe digitarse.

Programa ejemplo.

Usaremos como ejemplo un programa escrito en C que implementar una calculadora en notación polaca tomado del libro clásico de Kernighan yRitchie, "El lenguaje de programación C".

Notación polaca.

La notación polaca es una forma de indicar operandos y operaciones apta para ser implementada mediante un "stack" o pila. En las operaciones con dos operandos, como 5 + 3, se ingresan primero los operandos 5, 3 y luego la operación; el ingreso de la operación ya implica la ejecución y presentación del resultado. Como puede verse en los ejemplos, no necesita signo de igual ni paréntesis. Con un poco de práctica, esta notación es tan efectiva como la tradicional, requiere menos símbolos y es más fácil de implementar.
Ejemplo 1:

para calcular la expresión 10 - 8 se digita

```
10 8 - 4 5 + *
```

Ejemplo 2:

para calcular la expresión (10 - 8) * (4 + 5) se digita

```
10 8 - 4 5 + *
```

Archivos de programa.

Este programa es simple; puede implementarse en un archivo único, pero lo usaremos de ejemplo de un programa grande repartiendo el código en varios archivos de código c (extensión .c) y archivos de encabezados para las declaraciones (extensión .h).
El código está integrado por los archivos:

calc.h
getch.c
getop.c
main.c
stack.c

La compilación simple.

La compilación de este programa puede hacerse con el comando

```
$ gcc -o polaca main.c getch.c getop.c stack.c
```

que crea el archivo ejecutable polaca, que puede invocarse digitando

```
$ ./polaca
```

para probar su funcionamiento.
Toda 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`.

Un `makefile` elemental.

Cree 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, no se ejecutan. El resto de las líneas son reglas. Cada regla empieza con un nombre seguido de ":". Las líneas iniciadas con tabulador son continuación de la misma regla.

Compile el programa dando el comando

```
$ make
```

Observe los mensajes indicativos de las tareas realizadas. Digite luego

```
$ ./polaca
```

para verificar su funcionamiento. Digitando

```
$ make clean
```

se borra el ejecutable y los archivos objeto.

Reglas de make.

Las reglas tienen el siguiente formato:

```
destino : requisito ...
    comando
   ...
```

`destino` es el nombre de un archivo a crear, un ejecutble o un archivo objeto (.o). Tambien puede ser el nombre de una tarea realizar: 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` es el 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` es una acción a realizar. La creación de un destino puede requerir varios comandos. NOTAR que cada línea de comando debe comenzar con TABULADOR, no sirven los espacios. Notar también que el comando es un comando normal, que puede darse en la línea de comandos del shell, y debe ejecutar correctamente, pues `make` no sabe nada de él.
`\` es el caracter de continuación de línea; colocado al final de una línea es como si todo estuviera escrito en la primera línea; se usa sólo para facilitar la lectura.

Uso de variables.

La siguiente versión de `makefile` muestra el 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. Existen nombres de variables comprendidas por `make`, para usos específicos, como `CC` en el ejemplo; otras son costumbre, como `OBJECTS`; también pueden crearse otras a gusto del programador.

Reglas predefinidas.

`make` dispone de algunas reglas predefinidas. En particular, conoce el comando para compilar objetos desde C. Esto hace innecesario escribir los comandos en las reglas que crean los objetos, en tanto respondan a la manera usual de hacerlo.

```
# 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:

.PHONY : clean
    clean :
             -rm polaca $(objects)
```

Escribir la regla para `clean` de esta forma evita confusiones con un posible archivo llamado "clean", y también evita la detención de `make` ante errores del comando `rm`, por ejemplo si no se encuentran los archivos a borrar, situación normal si no fueron creados.

Es posible escribir reglas predefinidas específicas para cada makefile:

```
# CC = gcc
.SUFFIXES: .o .cpp
.cpp.o:
    $(CC) -c $<
```

La línea `.SUFFIXES` declara los sufijos a reconocer; la regla .cpp .o indica como obtener un archivo .o a partir de un archivo .cpp; la macro $< es interna de `make` y designa los prerrequisitos de la regla en el momento de ser usada (archivos fuente, archivos objeto). Luego de fijar esta regla genérica alcanza con escribir la lista de dependencias para obtener la actualización de un archivo con sufijo .o.

Opciones.

El comando make acepta entre otras estas opciones:

-n

muestra comandos a ejecutar, sin ejecutarlos

-f *archivo*

indica el nombre del archivo donde se encuentran las reglas, si no se llama "makefile".

Resumen.

La compilación de un ejecutable a partir de programas en C y sus archivos de encabezado puede simplificarse creando un archivo `makefile` siguiendo los modelos mostrados.

Para compilar usando un archivoun llamado makefile basta dar el comando

```
$ make
```

Si sólo quieren verse los comandos a ejecutar, digitar

```
$ make -n
```

Los comandos a ejecutar dependen del estado de actualización de los archivos; si no han habido cambios desde la última creación del archivo objeto la regla no se ejecuta.

```
$ make -f reglas.make
```

ejecuta make tomando las reglas del archivo `reglas.make`.