# Make Tutorial

Seminario Técnicas Digitales III
Marzo 2009

# Propósito

Make es una herramienta que permite ejecutar una secuencia de procesos.

Utiliza un script, llamada comúnmente ***makefile***

Es capaz de determinar automáticamente cuales pasos de una secuencia deben repetirse debido al cambio en algunos de los archivos involucrados en la construcción de un objeto, o en una operación.

# Usos mas comunes

- Recompilar programas que residen en diversos archivos, y...
- Testing de programas.

# Temario

- Ejemplo for dummies
- Variables
- Macros internas (solo algunas)
- Empaquetando objetos dentro del makefile
- Empaquetando el programa y fuentes (disribución)

# ¿Como se usa make?

De manera sencilla lo que tenemos que hacer es definir un archivo llamado Makefile en el directorio raíz de nuestro proyecto (en realidad lo podemos poner en otro lado) y dentro de ese archivo escribimos las reglas necesarias para construir nuestro proyecto.

Luego, alcanza con ejecutar

***make <regla>***

en el directorio en el que definimos el Makefile.

# Ejemplo 1

```
dependiente: dependiente.o dependencia.o
    gcc -g dependiente.o dependencia.o -o dependiente
dependiente.o: dependiente.c
    gcc -g -O0 -c dependiente.c -o dependiente.o
dependencia.o: dependencia.c
    gcc -g -O0 -c dependencia.c -o dependencia.o
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
```

# Variables

```
CC=gcc
dependiente: dependiente.o dependencia.o
    $(CC) -g dependiente.o dependencia.o -o
dependiente
dependiente.o: dependiente.c
    $(CC) -g -O0 -c dependiente.c -o dependiente.o
dependencia.o: dependencia.c
    $(CC) -g -O0 -c dependencia.c -o dependencia.o
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
```

# Macros especiales

- CC  Contiene el nombre del compilador C. Default cc.
- CFLAGS Flags y opciones que se agregan a la línea de compilación. (ver proxima.)
- $@  Nombre completo del target.
- $? lista de las dependencias que se encuentran desactualizadas.
- $< El archivo fuente de la dependencia actual.

# Mas variables y operadores

```
CC=gcc
CFLAGS=-g -O0
LDFLAGS=-g
OBJS=dependiente.o dependencia.o
dependiente: $(OBJS)
    $(CC) $(LDFLAGS)  $(OBJS) -o $@
dependiente.o: dependiente.c
    $(CC) $(CFLAGS) -c $< -o $@
dependencia.o: dependencia.c
    $(CC) $(CFLAGS) -c $< -o $@
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
```

**Esta macro $@ se refiere al target**

# Si miramos los objetos...

```
CC=gcc
CFLAGS=-g -O0
LDFLAGS=-g
OBJS=dependiente.o dependencia.o
dependiente: $(OBJS)
    $(CC) $(LDFLAGS)  $(OBJS) -o $@
%.o: %.c Makefile
    $(CC) $(CFLAGS) -c $< -o $@
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
```

# Empaquetando y distribuyendo

```
Makefile
CC=gcc
CFLAGS=-g -O0
LDFLAGS=-g
OBJS=dependiente.o dependencia.o
SOURCES=dependiente.c dependencia.c
HEADERS=*.h
dependiente: $(OBJS)
    $(CC) $(LDFLAGS)  $(OBJS) -o $@
%.o: %.c Makefile
    $(CC) $(CFLAGS) -c $< -o $@
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
entrega: $(SOURCES) $(HEADERS) Makefile
    tar zcvf entrega.tar.gz $(SOURCES) $(HEADERS) Makefile
```

# Vagueando... malll

```
Makefile
CC=gcc
CFLAGS=-g -O0
LDFLAGS=-g
OBJS=dependiente.o dependencia.o
SOURCES=$(OBJS:.o=.c)
HEADERS=*.h
dependiente: $(OBJS)
    $(CC) $(LDFLAGS)  $(OBJS) -o $@
%.o: %.c Makefile
    $(CC) $(CFLAGS) -c $< -o $@
clean:
    rm -f ./*.o
    rm -f dependiente
new: clean dependiente
entrega: $(SOURCES) $(HEADERS) Makefile
    tar zcvf entrega.tar.gz $(SOURCES) $(HEADERS) Makefile
```