LA INSTRUCCION MAKE

Este documento describe las características básicas de la instrucción make, con el fin de que el usuario de los sistemas de supercómputo de la UNAM pueda agilizar el mantenimiento y actualización de conjuntos de programas. Con frecuencia esta instrucción es empleada en la generación "inteligente" de archivos ejecutables, los cuales requieren de procesos de compilación en los que intervienen varios archivos interdependientes. Se asume que los lectores tienen conocimientos básicos sobre el sistema operativo UNIX.

LOS PUNTOS INCLUIDOS EN ESTE DOCUMENTO SON:

El archivo de descripción de make

La instrucción make permite mantener, actualizar y regenerar grupos de programas. Esta instrucción determina si un archivo está actualizado con respecto a otros de los cuales depende, y si es necesario actualiza el archivo. Para llevar acabo lo anterior la instrucción make se vale de un archivo de descripción (comúnmente llamado el makefile) el cual contiene:

Definiciones de macros	Líneas de la forma string=string que permiten definir variables locales en el makefile.
Líneas de dependencia	Este tipo de líneas están conformadas por dos partes separadas por el símbolo ":", a la izquierda de éste se encuentra el conjunto de ojetivos que generalmente son nombres de archivos a actualizar, a la derecha se ubican los archivos u otros objetivos de los que dependen los de la parte izquierda. Si se presenta una dependencia de un objetivo1 sobre otro objetivo2, entonces esto obliga a make a ejecutar primero la secuencia de instrucciones subordinados al objetivo2 antes que los del objetivo1.
Instrucciones ejecutables	Líneas que comienzan con un tabulador y se colocan a continuación de las líneas de dependencia. Aquí se encuentra la secuencia de instrucciones que permite actualizar los objetivos. Estas instrucciones serán ejecutados cuando los objetivos en la línea de dependencia no se encuentren actualizados con respecto a los archivos de los cuales dependen.
Líneas de comentarios	Líneas que comienzan con el símbolo # indican comentarios dentro del archivo de descripción.

Para ilustrar lo anterior, considere el ejemplo en el que tres archivos en el lenguaje "C", x.c, y.c y z.c se

complilan para generar los archivos objeto x.o, y.o y z.o, los cuales son ligados con la biblioteca matemática (-lm), para generar el archivo ejecutable prog. Asumamos además, que los archivos x.c y y.c comparten algunas declaraciones que se encuentran en el archivo defs.h. Es decir, los archivos x.c y y.c contienen en algún lugar la línea:

```
#include "defs.h"
```

El archivo de descripción del make, que describe las relaciones y operaciones para generar el ejecutable prog, podría verse como sigue:

```
$cat makefile
#ejemplo de un archivo de descripción del make
LIBS = -lm          Definición de macro
prog:  x.o y.o z.o prog es el objetivo de esta línea de dependencia
                   prog depende de tres archivos .o, loc cuales
                   se encuentran como objetivos en otras líneas de
                   dependencia
       cc -o prog x.o y.o z.o $(LIBS) Cuando prog no ha sido actualizado
                                      esta línea describe el uso de ls
                                      instrucción cc para actualizarlo
x.o:   x.c defs.h
       cc -c x.c
y.o:   y.c defs.h  El objetivo y.o depende de los
                   archivos y.c y defs.h
       cc -c y.c
z.o:   z.c
       cc -c z.c   Esta línea de instruccionrs describe el
                   uso de la instrucción cc para crear el archivo objeto z.o
```

Cuando un cambio es efectuado sobre alguno de los cuatro archivos fuente (x.c, y.c, z.c o defs.h), es posible actualizar "inteligentemente" el ejecutable prog simplemente introduciendo la instrucción make, pues éste examinará con base en el archivo de descripción, cuáles son las instrucciones mínimas necesarias que permiten actualizar a prog. Por ejemplo, si se modifica el archivo defs.h y se ejecuta la instrucción make, se tiene la siguiente respuesta:

```
$make
 cc -c x.c
 cc -c y.c
 cc -o prog x.o y.o z.o -lm
```

Por otro lado, si prog se encuentra actualizado con respecto a todos los archivos de las cuales depende y es ejecutado la instrucción make:

```
$make
'prog' is up to date
```

Si al momento de la ejecución de make, no se especifica en la línea de instrucciones un objetivo, entonces el primer objetivo de descripción (en nuestro caso prog) es asumido como punto de entrada para la ejecución de make. Por ejemplo, supongamos que los cuatro archivos fuente han sido modificados y que en lugar de querer actualizar prog, sólo se desea actualizar el archivo objeto x.o:

```
$make x.o
cc -c x.c
```

Ir al inicio

Sintaxis de la instrucción make

```
make [-f makefile] [-b] [-d] [-e] [-i] [-n] [-p] [-q] [-r] [-s] [-t]
     [objetivos] [variables para el makefile]
```

-fmakefile	especifica el nombre del archivo de make. Los default son: makefile, Makefile, s.makefile y s.Makefile.
-b	Compatibilidad con archivos de descripción viejos para el make.
-d	Debug para make
-i	Ignora errores, continua la ejecución
-r	make no utiliza las reglas internas
-s	No imprime salida en pantalla al momento de la ejecución
-t	Actualiza los objetivos aún sin reconstruirlos
-p	Imprime el conjunto de macro definiciones y descripción de objetivos
-n	Despliega instrucciones pero no los ejecuta
-e	Las variables del shell serán preferidas por el make sobre las variables (macros) definidas dentro del archivo de descripción

Ir al inicio

Macros internos al make

Antes de cualquier instrucción dentro del archivo de descripción sea ejecutado, ciertos macros internos al make son evaluados, Estos macros son útiles en la definición de reglas que permiten especificar objetivos. Cuatro son los macros internos de make:

Macro	Significado

$@	Este macro especifica el nombre del objetivo actual. Por ejemplo:
	
	`#En este caso $@ representará a prog`
	`prog : x.o y.o`
	`cc –o $@ x.o y.o`

$?	Representa al conjunto de prerrequisitos que no han sido actualizados son respecto al objetivo. Por ejemplo:

```
# En este caso $? representa a x.c o y.c o ambos
x.o y.o : x.c y.c
          cc -c $?
```

$<	Representa el nombre del prerrequisito que causó la acción. Este macro es empleado sólo en reglas con extensiones (ver reglas internas).
$*	Representa el nombre del objetivo actual con el sufijo removido. Este macro es empleado sólo en reglas con extensiones (ver reglas internas).

Ir al inicio

Variables y macros en make

Existen una serie de variables que la instrucción make puede emplear al momento de su ejecución, la siguiente tabla indica la prioridad con el que la instrucción make asume estas variables:

Prioridad	Lugar de definición
1	Variables definidas en la línea de instrucciones.
2	Variables definidas dentro del archivo de descripción (macros).
3	Variables definidas en el shell.
4	Variables definidas en al tabla interna de reglas.

Como se indico en el punto anterior, la opción -e de make, intercambia las prioridades 2 y 3 con que esta instrucción asume las variables. Por ejemplo, considérese el siguiente archivo de descripción:

```
$cat arch.conf.m
#Las secuencias $@ y $? dentro de este archivo de
#descripción, son macros internos al make.
FUENTES = main.c sub1.c sub2.c
OBJETOS = main.o sub1.o sub2.c
LIBS = lm
PROG = ejecutable
all : $(PROG)
$(PROG) : $(OBJETOS)
        cc -o $@ $(OBJETOS) -$(LIBS)
$(OBJETOS) : $(FUENTES)
        cc -c $?
install : all
        cpset $(PROG) $(HOME)/paquetes/$(PROG) 2755 gv gv
clean :
        -rm -f core
clobber : clean
        -rm -f $(OBJETOS) $(PROG)
```

Si antes de ejecutar la instrucción make, con este archivo de configuración, se teclea:

```
$LIBS = lsci
$export LIBS
```

```
$PROG = prog1
$export PROG
```

Entonces, ahora LIBS y PROG son variables del shell y al ejecutar el make:

```
$make -f arch.conf.m PROG=paqt1
cc -c main.c
cc -c sub1.c
cc -c sub2.c
cc -o paqt1 main.o sub1.o sub2.o -lm
```

Para que la instrucción make prefiera las variables del shell, sobre las variables definidas en el archivo de configuración, se emplea la opción e. Así, al ser removidos los archivos generados por la instrucción anterior y ser introducida la instrucción make con esta opción, se obtendra la siguiente salida:

```
$make -f arch.conf.m clobber PROG=paqt1
rm -f core
rm -f main.o sub.o sub2.o paqt1
$make -f arch.conf.m -e install
cc -c main.c
cc -c sub1.c
cc -c sub2.c
cc -o prog1 main.o sub1.o sub2.o -lsci
cpset prog1 /usr/gv/paquetes/prog1 27755 gv gv
```

Ir al inicio

Reglas internas

La instrucción make puede inferir directamente la actualización de un archivo con una determinada extensión a partir de un archivo con el mismo nombre pero con otra extensión. Para tal fin, la instrucción make se vale de una tabla interna que contiene las reglas de inferencia que le permite decidir qué archivos actualizar y como hacerlo. Es importante alcarar que esta serie de reglas internas, son empleadas por make siempre y cuando no exista el archivo de descripción o si existe, no ha sido descrito dentro de él la manera de obtener un objetivo en particular. Esta tabla se encuentra en /usr/src/cmd/make/rules.c y para ver las reglas que contiene es suficiente con teclear:

```
$make -fp - 2>/dev/null </dev/null
```

Para entender la importancia de esta tabla de reglas, supongamos que se tiene el archivo que contiene un fuente en Fortran **prog.f** del cual se quisiera obtener su archivo objeto o el ejecutable (**prog.o** y **prog**). Para obtener el archivo objeto **prog.o** basta con teclear:

```
$make prog.o
cf77 -c prog.f
```

Pero si lo que desea es su ejecutable, entonces se teclea:

```
$make prog
cf77 prog.f -o prog
```

Obsénvese que en ninguno de los dos casos anteiores se he especificado un archivo de descripción para make. Sin embargo, dentro de la tabla interna de reglas, make encuentra descrito, entre otras cosas la manera de obtener, a partir de archivos fuentes de Fortran, C, Pascal, etc., sus archivos objeto o ejecutables. En nuetro ejemplo en particular, la instrucción make ha empleado las siguientes líneas de la tabla interna de reglas:

```
FFLAGS =
CF = cf77
LDFLAGS =
.f.o:
        $(CF) $(FFLAGS) -c $*.f
.f  :
        $(CF) $(FFLAGS) $(LDFLAGS) $< -o $@
```

En estas líneas se describe la regla para crear un archivo con extensión .o a partir de un archivo con extensión .f, esto se especifica como .f.o: De manera semejante, la regla para crear el archivo con extensión nula (archivo ejecutable) a partir de uno con extensión .f: Obsérvese, que los macros $< y $*.f representan ambos el nombre del archivo fuente.

La opción -r le indica a la instrucción make que ignore la tabla interna de reglas. De esta manera, si se emplea esta opción en el ejemplo anterior, se tiene el siguiente mensaje de error:

```
$make -r prog.o
Don't know how to make prog.o. Stop.
```

Ir al inicio

Referencias

UNICOS Support Tools Guide	SG-2016 6.0
UNICOS Utility Programs	TR-UUP
Guía Introductoria a la computadora CRAY Y-MP	
Entorno de programación UNIX	Brian W. Kernighan & Rob Pike

Ir al inicio