Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Introducción a M4 para Bisoños (10425 lectures)

Per **Domingo Fiesta Segura**, ***C2H5OH*** (http://www.krenel.net)

Creado el 09/09/2002 17:25 modificado el 09/09/2002 17:25

Para las personas a las que nos da pereza escribir demasiado nos gusta descubrir formas de que el ordenador escriba por nosotros. Una de ellas es usando M4.

Índice

1. ¿Qué es M4?

M4 es un procesador de **macros**. Por macros se entiende expansiones de texto. Para entender esto basta recordar el archiconocido procesador de macros CPP utilizado en todas las *suites* de compiladores de C.

Todo el que ha programado en C sabe utilizar el CPP. Se sabe que usando `#define SUMA(a, b) (a) + (b)`, antes de compilar se procesará todo el fichero fuente buscando `SUMA(talo, cualo)` para ser substituido por `(talo) + (cualo)`. M4 tiene la misma función, la diferencia está en que utiliza un lenguaje de macros diferente y de propósito general; lo cual resulta más potente.

2. Funcionamiento

Si ejecutáis el comando `m4`[1] sin argumentos el comando espera texto por la entrada estándar. Tras cada línea introducida, ésta se procesa y se imprime el resultado por la salida estándar. También se puede usar con tuberías y, si como argumentos le pasáis nombres de ficheros, éstos se procesan secuencialmente imprimiendo el resultado por la salida estándar.

3. Utilidades de M4

La mayor utilidad que se le ha dado a M4 ha sido en las famosas **autotools** de GNU (*autoconf*, *automake*, etc). Realmente el procesador de M4 se limita a subsituir ciertas cadenas de texto por otras (a lo que llamaremos **expansión**), así podemos usarlo para tantas situaciones como nuestra imaginación nos permita. Otro ejemplo es utilizar M4 para escribr páginas web (creando una abstracción por encima del HTML).

[1] O `gm4` si estáis con *BSD y queréis usar *GNU M4*, BSD trae su propio procesador M4 para la configuración del *sendmail*

4. Introducción a la sintaxis de M4

De M4 lo único que cuesta aprender es la forma de expansión que lleva a cabo al encontrar una macro. A pesar de ello el manual oficial[11] lo explica todo muy bien. Pero como en todo se aprende y se entiende practicando.

4.1. Utilización de macros

Para utilizar una macro basta con escribir su nombre. Si tenemos que utilizar parámetros en dicha macro los rodearemos por paréntesis. Por ejemplo, si tenemos la macro CANTINFLAS la podemos utilizar escribiendo su nombre en una palabra por sí misma. Las siguientes líneas expanden la macro CANTINFLAS:

```
CANTINFLAS
Hola señor CANTINFLAS
Hola señor CANTINFLAS()puntoycoma.
Hola señor CANTINFLAS(punto, y, coma).
Hola señor CANTINFLAS(punto y coma).
```

Las dos primeras expanden la macro **sin argumentos**, la tercera expande la macro pasándole el argumento de **cadena vacía**[2], la cuarta expande la macro pasándole **tres** argumentos y la quinta expande la macro pasándole **un** argumento.

Las siguientes líneas **NO** expanden la macro CANTINFLAS:

```
eseCANTINFLASpoderoso
CANTINFLASillo
antiCANTINFLAS()
```

4.2. Definición de macros

La macro más esencial que necesitamos es aquella que nos permite definir nuevas macros. Esta macro se llama **define**; que adminte dos argumentos: el primero es el nombre de la nueva macro y el segundo es el texto por el que será reemplazada cada aparición de dicha macro. Es la misma idea que los `#define` del CPP. Veamos unos ejemplos:

```
define(`CANTINFLAS', `Bigote arrocet')
define(`BOLD', `<b>$1</b>')
define(HOLA, Hola mundo cruel)
```

Así, cada vez que se encuentre `CANTINFLAS` será substituido por `Bigote arrocet`, de la misma forma que cada vez que se encuentre `HOLA` se *expandirá* en `Hola mundo cruel`.

El `$1` se expande por el primer arugmento que se le pase a la macro. Por lo que si escribimos `BOLD(hola)` será substituido por `<b>hola</b>`. Pasa lo mismo para `$2`, `$3`, etc...

También se puede usar `$0` que se expande como el nombre de la macro. Después tenemos `$*` que expande todos los argumentos separados por comas y `$@` que hace lo mismo pero además *quotea* el resultado para evitar expansiones del mismo. La cadena `$#` se expande como el número de argumentos pasados a la macro.

> **IMPORTANTE**
>
> El uso de `$1`, `$@` y demás expansiones especiales **únicamente** funcionan dentro de la macro `define`. Si, dentro de una macro `define`, queréis que se expanda literalmente, por ejemplo, `$@` tendréis que hacerlo escribiendo algo como `$`'@`. Como siempre, probadlo para entenderlo mejor.

4.3. Comentarios

En M4 existen dos tipos de comentarios: los que se se omiten en el resultado de la expansión y los que no se interpretan pero pasan tal cual hacia la salida estándar. Probad esto con vuestro M4 y veréis la diferencia:

```
dnl Comentario que no aparecerá en el resultado.
```

```
# Comentario que sí aparecerá en el resultado
```

[2] Se consigue escribiendo un *quoteado* vacío ``'. Más adelante vermos acerca de las comillas

5. *Quoteado* o protección de cadenas de texto

Los delimitadores de cadenas, digamos, *protegidas* son ` y '. Pero es posible cambiarlos con la macro `changequote`. Voy a intentar explicar esto con cuidado pues es la clave para entender el funcionamiento de M4.

Cuando hablamos de una *expansión* nos referimos a dos cosas:

- Expandir una macro o
- Si se trata de un texto rodeado por ` ' simplemente se eliminan las comillas.

Si se ha dado el segundo caso no hacemos nada más. Pero el proceso de expansión de una macro es algo más sofisticado y, además, recursivo. Cuando M4 **reconoce** una macro procede a expandirla de la siguiente forma:

1. Expansión de los argumentos.
2. Se procede a la substitución del texto.
3. Se vuelve a procesar el resultado de esta expansión.

Mucho ojo porque esto pasa para cada macro que se encuentra. Si resulta que mientras se expanden los argumentos de una macro se reconoce otra macro se realiza este mismo proceso para esta última. He aquí la importancia de saber **proteger** el texto *quoteándolo*. Así pues, para la inclusión de cadenas **literales** en el argumento de una macro basta con rodearla de comillas dobles, como por ejemplo:

```
define(`CANTINFLAS', `$1')
Hola señor CANTINFLAS(``Y esto aparece tal cual, metamos lo que metamos'')
```

El procedimiento sería el siguiente. Se reconoce la macro `define` por lo que se expanden sus argumentos. Como los argumentos estaban *quoteados*, se quitan las comillas y no se produce ninguna expansión extra. Esta macro se expande en la cadena vacía ``'.

Posteriormente se encuentra la macro `CANTINFLAS` (previamente definida) y se procede a su expansión. Se expande su argumento (que estaba protegido) por lo que se le quitan las comillas externas. La expansión de la macro produce dicho argumento tal cual fue expandido, es decir, `` `Y esto aparece tal cual, metamos lo que metamos' ``. Tras esto se vuelve a procesar este texto resultante por lo que se vuelven a eliminar las comillas quedando el texto literal.

Nótese que si se encuentra una macro mientras se procesaba otra (paso 3), se vuelve a llevar a cabo todo el proceso de expansión sobre aquella. Para aprender mejor este proceso os recomiendo que hagáis las siguientes pruebas:

```
define(`echo1', `$*')
define(`echo2', `$@')
define(`echo3', `$1')
echo3(`echo2(una, `prueba')'), tonta)
echo3(echo1(`echo2(una)', `prueba'), `tonta')
echo2(`echo1(echo3(una))', echo2(`prueba', tonta))
```

...y así sucesivamente.

6. Ramificaciones

Con M4 es posible *retrasar* la salida de texto gracias a las ramificaciones[3]. Cuando cambiamos de ramificación el texto generado se guarda temporalmente para ser expulsado por mosotros manualmente o automáticamente al final. Las macros involucradas en éste proceso son `divert` y `undivert`. Ambas aceptan un único parámetro numérico que es el número de ramificación.

La ramificación por defecto es la número 0. Cuando cambiamos de rama con `divert` e introducimos texto éste se guarda en la rama correspondiente. Cuando se acaba la entrada estándar es expulsado por orden ascendente de ramificación. A no ser que lo expulsemos nosotros manualmente con `undivert`. Veámoslo con algunos ejemplos.

```
Aquí pongo texto normal
divert(2)
Éste texto se ramifica por la rama 2.
divert(3)
Hola hola ésto es la ramificación divnum !!!
divert(1)
Éste texto se ramifica por la rama 1 pero aparece antes que el de la rama 2.
divert(0)
Volvemos a la salida estándar.
undivert(3)
divert(-1)
Éste texto NO APARECE en la salida
```

En realidad `divert` se expande en la cadena vacía pero cambia la ramificación actual. Sin embargo `undivert` se expande por el texto almacenado en la rama indicada. Hay otra macro relacionada llamada `divnum` que no necesita argumentos y que se expande por el número de ramificación actual.

[3] En inglés: *diversions*

7. Notas finales

Si necesitamos que una macro se expanda entre texto que no está separado por espacios podemos usar la cadena vacía `'. Por ejemplo:

```
ese`'CANTINFLAS`'poderoso
anti`'CANTINFLAS()
```

También mencionar que M4 tiene macros para implementar condicionales, como `ifdef` e `ifelse`. Pero realmente os recomiendo echarle un vistazo al manual[11] que es muy completo salvo por la explicación de la expansión.

Finalmente, quiero hacer notar de que M4 **no** es un lenguaje de programación sino un procesador de macros. Simplemente se limita a reconocer y expandir macros. En el manual hay un ejemplo de como implementar un bucle (de forma recursiva) pero como ejemplo ilustrativo. Si teneis ganas de ver M4 en su máximo esplendor echadle un vistazo a los ficheros `.m4` de las distribuciones de *autoconf* y *automake*.

Si, en cambio, queréis empezar por algo más suave podéis ver las definiciones[12] de macro para Bulma M4[13].

Generado: lunes, 23-agosto-2004 2:42

Lista de enlaces de este artículo:

1. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=1#sec_1
2. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=2#sec_2
3. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=2#sec_3
4. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=3#sec_4
5. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=3#sec_4_1
6. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=3#sec_4_2
7. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=3#sec_4_3
8. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=4#sec_5
9. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=4#sec_6
10. http://bulma.net/body.phtml?nIdNoticia=1492&nIdPage=5#sec_7
11. http://www.gnu.org/manual/m4-1.4/m4.html
12. http://mnm.uib.es/~etanol/bulma/transform.m4
13. http://bulmalug.net/body.phtml?nIdNoticia=1618

E-mail del autor: etanol _ARROBA_ krenel.net
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1492