Introducción a Sistemas Operativos: Usando el shell

Clips xxx
Francisco J Ballesteros

1. Funciones y otras estructuras de control

Aún tenemos pendiente escribir nuestro script para procesar los ficheros de configuración de nuestra aplicación y ver si todos son correctos.

Lo primero que deberíamos hacer es pensar en qué opciones y argumentos queremos admitir en nuestro script. Por ejemplo, un posible uso podría ser

```
chkappconf [-sd] [fich...]
```

Hemos utilizado la sintáxis que vemos en la synopsis de las páginas de manual:

- Los flags `-d` y `-s` pueden utilizarse opcionalmente y permiten comprobar sólo los directorios utilizados en los ficheros de configuración o sólo los shells mencionados. Si no indicamos ninguno de estos flags queremos que se comprueben ambas cosas.
- Si se indican nombres de fichero (uno o más) como argumento entonces haremos que se comprueben sólo dichos ficheros, en caso contrario comprobaremos todos los ficheros en los directorios `$HOME` de todos los usuarios.

Sabemos que `$*` corresponde al vector de argumentos en un script, excluyendo el nombre del script (`argv[0]`, que sería `$0`). Una posibilidad es procesar dicha lista de argumentos mientras veamos que comienzan por "`-`", y luego veremos cómo procesar las opciones que indican.

Podemos empezar con algo como esto, que guardaremos en nuestra primera versión de `chkappconf`:

```
#!/bin/sh

while test $# -gt 0
do
    echo $1
    shift
done
```

Aquí utilizamos la estructura de control `while` del shell, que ejecuta el comando indicado como condición y, mientras dicho comando termine con éxito, ejecuta las sentencias entr `do` y `done`. Como condición,

```
test $# -gt 0
```

compara el número de argumentos con `0` usando *greater than* como comparación. Por último, dentro del cuerpo del `while` escribimos el primer argumento y posteriormente lo tiramos a la basura. La primitiva `shift` tira el primer argumento y conserva el resto.

Vamos a probarlo:

```
unix$ chkappconf -a b c
-a
b
c
unix$
```

Ahora estaría bien detener el while si el primer argumento no compienza por un "`-`". Por el momento podríamos usar `egrep` para eso. Hagamos un par de pruebas:

```
unix$ echo -a | egrep '^-'
-a
unix$ echo a | egrep '^-'
unix$
```

¡Estupendo! Pero sólo nos interesa ver si `egrep` dice que la entrada tiene ese aspecto o no. En lugar de enviar su salida a `/dev/null`, usaremos la opción `-q` (*quiet*) de `egrep` que hace que no imprima nada y tan sólo llame a *exit(2)* con el estatus adecuado.

```
unix$ echo -a | egrep -q '^-'
unix$ echo $?
0
unix$ echo a | egrep -q '^-'
unix$ echo $?
1
```

Ya casi estamos. Ahora necesitamos poder escribir como condición que tanto `test` compruebe que hay algún argumento como el comando anterior vea que comienza por un "`-`". Pero esto es fácil. El shell dispone de "`&&`" y "`||`" como operadores. Significan lo mismo que al evaluar condiciones en C, pero naturalmente esto es shell y utilizamos líneas de comandos. Nuestro script queda así por el momento:

```
#!/bin/sh

while test $# -gt 0 && echo $1 | egrep -q '^-'
do
    echo option arg: $1
    shift
done
echo argv is $*
```

Y si lo ejecutamos, vemos que funciona:

```
unix$ chkappconf -a -ab c d
option arg: -a
option arg: -ab
argv is c d
unix$
```

El siguiente paso es procesar cada uno de los argumentos correspondientes a opciones. Aunque podríamos recurrir a `egrep` de nuevo, vamos a utilizar la estructura de control `case` del shell, que sabe como comprobar si un string encaja con una o más expresiones. Veamos antes un ejemplo:

```
unix$ x=ab
unix$ case $x in
> a*)
> echo a;;
> b*)
> echo b;;
> esac
a
unix$
```

Esta estructura compara el string entre `case` e `in` con cada una de las expresiones de cada rama del `case`. Estas expresiones son similares a las utilizadas para globbing y terminan en un "`)`". Tras cada expresión se incluyen las líneas de comandos que hay que ejecutar en dicho caso, terminando con un "`;;`". Por último, se cierra el `case` con "`esac`".

Podemos escribir expresiones como

```
case ... in
[ab]c|d)
    ...
    ;;
esac
```

Esto es, es posible escribir conjuntos de caracteres "`[...]`" y también indicar expresiones alternativas separadas por un "`|`". Dado que `case` intentar encajar las expresiones en el orden en que se dan, y que "`*`" encaja con cualquier cosa, no existe `else` o `default` para `case`. Basta usar

```
case ... in
...
*)
    ... este es el default ...
    ;;
esac
```

Volvamos a nuestro script. Vamos a definir un par de variables para los flags y a comprobar si aparecen como opciones.

```
#!/bin/sh

dflag=y
sflag=y
while test $# -gt 0 && echo $1 | egrep -q '^-'
do
    case $1 in
    *d*)
        dflag=y
        sflag=n
        ;;
    esac
    case $1 in
    *s*)
        sflag=y
        dflag=n
        ;;
    esac
    shift
done
echo dflag $dflag
echo sflag $sflag
echo argv is $*
```

Habitualmente habríamos usando "`n`" como valor inicial para los flags y los habríamos puesto a "`y`" si aparecen. Pero en este caso hay que procesar tanto directorios como shells si ninguno de los flags aparece. ¡Probémoslo!

```
unix$ chkappconf -d c d
dflag y
sflag n
argv is c d
unix$
```

Ya disponemos de `$dflag` para ver si hay que procesar sólo directorios y de `$sflag` para ver si hay que procesar sólo shells. Además, hemos dejado `$*` con el resto de argumentos.

Pero... ¿Y si hay una opción inválida? Bueno, siempre podemos recurrir a `echo` y `egrep` para ver si las opciones tienen buen aspecto:

```
...
while test $# -gt 0 && echo $1 | egrep -q '^-'
do
    if echo $1 | egrep -q '[^-sd]' ; then
        echo usage: $0 '[-sd] [file...]' 1>&2
        exit 1
    fi
    case $1 in
    ...
done
...
```

Ahora podemos ponernos a hacer el trabajo según las opciones y los argumentos. Lo primero que haremos es dejar en una variable la lista de ficheros que hay que procesar.

```
files=$*
if test $# -eq 0 ; then
    files="/home/*/.myapp"
fi
```

Suponiendo que los ficheros de configuración se llaman `.myapp` y que queremos procesar dichos ficheros para todos los usuarios si no se indica ningún fichero.

Ahora podríamos procesar un fichero tras otro...

```
for f in $files ; do
    echo checking $f...
    if test $dflag = y ; then
        checkdirs $f
    fi
    if test $sflag = y ; then
        checkshells $f
    fi
done
```

Aquí vamos a utilizar `checkdirs` y `checkshells` para comprobar cada fichero. Aunque podríamos hacer otros scripts, parece que lo más adecuado es definir funciones. Y sí, ¡el shell permite definir funciones!.

Para definir una función escribimos algo como

```
myfun() {
    ...
}
```

En este caso, `myfun` es el nombre de la función. Tras el nombre han de ir los paréntesis (¡Sin nada entre ellos!) y después los comandos que constituyen el cuerpo entre llaves. Recuerda que estás usando el shell, esto no es C.

Dentro de la función, los argumentos se procesan como en un script. Así pues, en nuestro caso "`$1`" es el nombre del fichero que hay que comprobar y las funciones podrían ser algo como

```
checkdirs()
{
    echo checking dirs in $1
}
checkshells()
{
    echo checking shells in $1
}
```

Hemos terminado. Mostramos ahora el script entero para evitar confusiones.

```
#!/bin/sh

checkdirs()
{
    file=$1
    dirs=‘egrep ’^[a-z]+dir ’ config | \
        awk ’{print $1}’ | sort | uniq‘
    for d in $dirs ; do
        if test ! -d $d ; then
            echo no dir $d in $file
        fi
    done
}


checkshells()
{
    file=$1
    shells=‘egrep ’^shell ’ config | \
        awk ’{print $1}’ | sort | uniq‘
    for s in $shells ; do
        if test ! -x $x ; then
            echo no shell $s in $file
        fi
    done
}


dflag=y
sflag=y
while test $# -gt 0 && echo $1 | egrep -q ’^-’
do
    if echo $1 | egrep -q ’[^-sd]’ ; then
        echo usage: $0 ’[-sd] [file...]’ 1>&2
        exit 1
    fi
    case $1 in
    *d*)
        dflag=y
        sflag=n
        ;;
    esac
    case $1 in
    *s*)
        sflag=y
        dflag=n
        ;;
    esac
    shift
done
```

```
files=$*
if test $# -eq 0 ; then
    files="a b c"
fi
for f in $files ; do
    if test $dflag = y ; then
        checkdirs $f
    fi
    if test $sflag = y ; then
        checkshells $f
    fi
done
```

Hay formas mejores de hacer lo que hemos hecho. Pero lo que importa es que hemos podido hacerlo con las herramientas que tenemos. Por ejemplo, la práctica totalidad de los shells modernos incluyen *getopts(1)* para ayudar a procesar opciones y, además, tienes *getopt(1)* en cualquier caso para la misma tarea. Sólo con eso, ya podemos simplificar en gran medida nuestro script.