12.Parámetros

Como se vió en los capítulos anteriores, el formato general para la invocación de programas en UNIX es

command options files

Donde command es el nombre de un comando, options es cualquier opción que se necesite especificar, y files es una lista de archivos opcionales sobre los que el comando podría operar. En el siguiente ejemplo:

$ ls -l *.doc

Acá ls es el comando, -l es alguna opción, y *.doc es la lista de archivos sobre la cual ls trabaja.

Dado que esta es la forma estándar de la la mayoría de los UNIXes, nuestros scripts deberían adherir a este formato.

Existen 2 formas comunes de manejar opciones pasadas a un script:

- Manejar las opciones en forma manual usando la sentencia case
- Manejar las opciones usando el comando getopts

En el caso de pocas opciones, la primer opción es fácil de implementar y funciona correctamente. Cuando un script permite muchas opciones y combinaciones el comando getopts brinda la mayor flexibilidad.

Variables Especiales

El shell define varias variables especiales que son relevantes para el manejo de opciones. Además, algunas variables brindan el estado de los comandos ejecutados por el script.

Tabla 12.1 Variables Especiales del Shell

Variable	Descripción
$0	El nombre del comando que se está ejecutando.
$n	Estas variables corresponden a los argumentos con los cuales se invocó el script. En este caso n es un entero positivo correspondiente a la posición del argumento.(el 1er argumento es $1, el segundo $2, y así sucesivamente).
$#	El número de argumentos.
$*	Todos los argumentos. Si el script recibe 2 argumentos es equivalente a $1 y $2.
$@	Todos los argumentos. Si el script recibe 2 argumentos es

	equivalente a $1 y $2. Preserva el quoteado.
$?	El estado de terminación del último comando ejecutado.
$$	El número de proceso del shell actual.
$!	El número de proceso del último comando background.

Uso de $0

Esta variable es comunmente usada para determinar el comportamiento de un script que puede ser invocado con más de un nombre.

```
#!/bin/sh

case $0 in
        *listtar) TARGS="-tvf $1" ;;
        *maketar) TARGS="-cvf $1.tar $1" ;;
esac

tar $TARGS
```

Se puede utilizar este script para listar el contenido de un archivo tar o crear un archivo tar basado en el nombre suministrado como parámetro.

Se puede llamar al script mytar y hacer 2 links simbólicosllamados listtar y maketar:

```
$ ln -s mytar listtar
$ ln -s mytar maketar
```

Sentencias de “Uso”

Otro uso común es usar $0 en la sentencia de uso de un script, esto es un mensaje corto indicando al usuario como invocar el script correctamente. Todos los scripts utilizados por más de un usuario deberían incluir este mensaje.

En general , el la sentencia de uso es algo como:

```
echo "Usage: $0 [options][files]"
```

Si se el script mytarmytar, la inclusión de una sentencia de uso sería importante que indique cómo debe utilizarse el script en caso de error. Para implementar esto se debe cambiar el case de la siguiente manera:

```
case $0 in
        *listtar) TARGS="-tvf $1" ;;
        *maketar) TARGS="-cvf $1.tar $1" ;;
        *) echo "Usage: $0 [file|directory]"
                exit 0
                ;;
```

```
esac
```

Con esta modificación, si el script es invocado sólo con mytar, se ve el siguiente mensaje:

```
Usage: mytar [file|directory]
```

A pesar que el mensaje describe el uso correcto del script, no dice cual fue el error. Hay 2 métodos posibles para solucionar esto:

- Hardecodear los nombres válidos en la "sentencia de uso"
- Cambiando el script para usar los argumentos para decidir en que modo debe correr

Para demostrar el uso de opciones, la siguiente sección muestra el segundo método.

Opciones y Argumentos

La opciones se pasan el la línea de comandos para cambiar el comportamiento de un script o programa. Por ejemplo, la opción -a del comando ls cambia el comportamiento del comando ls para que liste todos los archivos. Esta sección muestra como usar opciones para cambiar el comportamiento de de scripts.

A menudo se puede ver que se llamen argumentos a las opciones. La diferencia es sutil. Un argumento de un comando son todos los strings que aparecen en la línea de comando después del nombre del comando, mientras que las opciones son sólo aquellos argumentos que cambian el comportamiento del comando.

```
$ ls -aF autos
```

El comando es ls, y los argumentos son -aF y autos. Las opciones del comando ls son -aF.

Trabajando con Argumentos

Para ilustrar el uso de opciones, se cambia el script mytar para que use su primer argumento, $1, como argumento de opción y $2 como el archivo tar a leer o crear.

Para implementar este cambio se modifica el case:

```
USAGE="Usage: $0 [-c|-t] [file|directory]"

case "$1" in
        -t) TARGS="-tvf $2" ;;
        -c) TARGS="-cvf $2.tar $2" ;;
        *) echo "$USAGE"
```

```
            exit 0
            ;;
esac
```

Los 3 grandes cambios son

- Todas las referencias a $1 se cambiaron a $2 dado que el 2do argumento es ahora el nombre del archivo.
- listtar se reemplazó por -t.
- maketar se reemplazó por -c.

Ahora cuando se corre mytar produce la siguiente salida:

```
Usage: ./mytar [-c|-t] [file|directory]
```

Para crear un tar con el directorio autos with this version, use the command

```
$ ./mytar -c autos
```

Para listar el contenido del archivo tar, autos.tar, se usa el comando

```
$ ./mytar -t autos
```

Uso de basename

El mensaje muestra el path completo con el cual el script fue invocado, pero lo que se necesita realmente es el nombre del script. Esto se puede corregir utilizando el comando basename.

El comando basename toma un path absoluto o relativo y retorna el nombre del archivo o directorio. Por ejemplo:

```
$ basename /usr/bin/sh
```

imprime:

```
sh
```

Usando basename, se puede cambiar la variable $USAGE en el script mytar:

```
USAGE="Usage: `basename $0` [-c|-t] [file|directory]"
```

Esto produce la siguiente salida:

```
Usage: mytar [-c|-t] [file|directory]
```

Problemas Comunes en el Manejo de Argumentos

Ahora que el script mytar usa opciones para cambiar el modo en que se que ejecuta el script, hay otro problema que resolver. Que debe hacerse si el

segundo argumento, $2, no se provee?

No hay que preocuparse por lo que pasa con el 1er argumento , $1, porque el case maneja esto con la cláusula default, *.

La forma más simple de chequear que se pasó el número necesario de argumentos es verificar si el número de argumentos pasados, $#, concuerda con el número requerido.

```
#!/bin/sh

USAGE="Usage: `basename $` [-c|-t] [file|directory]"

if [ $# -lt 2 ] ; then
        echo "$USAGE"
        exit 1
fi

case "$1" in
        -t) TARGS="-tvf $2" ;;
        -c) TARGS="-cvf $2.tar $2" ;;
        *) echo "$USAGE"
                exit 0
                ;;
esac

tar $TARGS
```

Manejando Archivos Adicionales

El script mytar está casi finalizado, pero aún se pueden hacer algunas mejoras. Por ejemplo, sólo maneja con el primer archivo pasado como argumento, y no valida si el argumento es realmente un archivo.

Se puede agregar el procesamiento de todos los archivos pasados como argumento usando la variable especial $@.

```
case "$1" in
        -t) TARGS="-tvf"
                for i in "$@" ; do
                        if [ -f "$i" ] ; then tar $TARGS "$i" ; fi ;
                done
                ;;
        -c) TARGS="-cvf $2.tar $2" ;
                tar $TARGS
                ;;
        *) echo "$USAGE" ;
                exit 0
                ;;
esac
```

El mayor cambio es que la opción -t del case ahora incluye el for que itera sobre los argumentos y verifica si cada uno es un archivo. Si el argumento es un archivo, se invoca tar con dicho archivo.

Algunos Problemas Menores

Aún quedan algunos problemas menores que solucionar. Mirando en detalle el script, se puede ver que todos los argumentos pasados al script, incluyendo el 1er argumento, $1, son considerados como archivos. Dado que el primer argumento es utilizado como flag, no debería considerarse.

Para remover el primer argumento de la lista, se utiliza el comando shift. Un cambio similar debe hacerse al modo de creación.

Otro problema es que lo que debe hacer el script cuando la operación falla. En el caso de la operación de listado, si el tar no puede listar el contenido de un archivo, sería razonable saltear el archivo e imprimir el error.

Dado que el shell setea la variable $? con el código de terminación del comando más reciente, se puede utilizar para determinar si el tar falló o no.

Resolviendo estos problemas, queda el script:

```
#!/bin/sh

USAGE="Usage: `basename $0` [-c|-t] [files|directories]"

if [ $# -lt 2 ] ; then
        echo "$USAGE" ;
        exit 1 ;
fi

case "$1" in
        -t) shift ; TARGS="-tvf" ;
                for i in "$@" ; do
                        if [ -f "$i" ] ; then
                                FILES=`tar $TARGS "$i" 2>/dev/null`
                                if [ $? -eq 0 ] ; then
                                        echo ; echo "$i" ; echo "$FILES"
                                else
                                        echo "ERROR: $i not a tar file."
                                fi
                        else
                                echo "ERROR: $i not a file."
                        fi
                done
                ;;
        -c) shift ; TARGS="-cvf" ;
                tar $TARGS archive.tar "$@"
```

```
        ;;
    *) echo "$USAGE"
        exit 0
        ;;
esac

exit $?
```

Parsing de Opciones

Hay 2 métodos comunes de manejar el parseo de opciones pasadas a un script. En el primero, se puede tratar manualmente con las opciones utilizando como se vio la sentencia case. El segundo método, tratado en esta sección, es el uso del comando getopts.

La sintaxis del comando getopts command es

getopts option-string variable

Donde option-string es un string con los caracteres simples con todas las opciones que getopts debe considerar, y variable es el nombre de la variable a la cual la opción debe asignarse.

La forma en que getopts parsea las opciones pasadas en la línea de comando es

1. getopts examina todos lo argumentos de la línea de comando, buscando los que comienzan con el carácter –.
2. Cuando se encuentra un argumento que comienza con el caracter -, luego compara el carácter que precede al - con los caracteres pasados en option-string.
3. Si encuentra uno que concuerda, la variable variable se setea con la opción: de otra forma, variable es seteada al caracter ?.
4. Los pasos 1 a 3 se repiten hasta que todas las opciones han sido consideradas.
5. Cuando ternina el parseo, getopts retorna un código distinto de 0. Esto permite utilzarlo fácilmente en loops. Además, cuando getopts finaliza, setea la variable OPTIND al índice del último argumento.

Otro funcionalidad de getopts es la capacidad de indicar opciones que requieren parámetros adicionales. Esto se puede conseguir agregando el caracter : luego de la opción en option-string. En este caso, depués de que la opción es parseada, el parámetros adicional es asignado a la variable OPTARG.

Uso de getopts

Para comenzar a entender como trabaja getopts y como tratar con opciones, se verá un script que simplifica la generación del hash md5 de un archivo.

El comando md5sum, es un que entre otras cosas se utiliza para generar el hash de archivos y verificar su integridad.

Primero se examinará la interfaz del script para facilitar su comprensión. Este script debería ser capaz de aceptar las siguientes opciones:

- -f para indicar el nombre del archivo de entrada
- -o para indicar el nombre del archivo de salida
- -v para indicar que el script debe ser verbose

El comando getopts necesario para implementar estos requerimientos es

```
getopts e:o:v OPTION
```

Esto indica que todas las opciones excepto -v requieren un parámetro adicional. Las variables que se necesitan para esto son

- VERBOSE, que guarda el valor del flag verbose. Por defecto es false.
- INFILE, que guarda el nombre del archivo de entrada.
- OUTFILE, que guarda el nombre del archivo de salida. Si este valor no se especifica, md5sum utiliza el nombre del archivo de entrada agregandole la extensión .md5.

El loop que implementa esto es

```
VERBOSE=false

while getopts f:o:v OPTION ;
do
        case "$OPTION" in
                f) INFILE="$OPTARG" ;;
                o) OUTFILE="$OPTARG" ;;
                v) VERBOSE=true ;;
                \?) echo "$USAGE" ;
                        exit 1
                        ;;
        esac
done
```

Ahora que se manejó el parseo de opciones, se necesita aún manejar otras condiciones de error. Por ejemplo que sucede si el archivo de entrada no se especifica?

La respuesta simple es salir cuando hay un error, pero con poco trabajo, se puede hacer el script más amigable al usuario. Si se usa el hecho de que getopts asigna el valor de la última opción a OPTIND, se puede tener un script que asume que el primer argumento después de este es el archivo de entrada. Si no hay argumentos adicionales, se debe salir. El chequeo de error consiste en

```
shift `echo "$OPTIND - 1" | bc`

if [ -z "$1" -a -z "$INFILE" ] ; then
        echo "ERROR: No se especificó el archivo de entrada."
        exit 1
fi

if [ -z "$INFILE" ] ; then INFILE="$1" ; fi
```

En este caso el shift se utiliza para descartar los argumentos pasados al script excepto el último procesado por getopts. Después de desplazar los argumentos, se verifica que el nuevo $1 contiene algún valor. Si no, imprime y termina. De otra forma, asigna a INFILE el nombre del archivo especificado por $1.

En el caso de que la opción -o no se especifique, también se debe setear el nombre del archivo de salida. Se puede utilizar substitución de parámetros:

```
: ${OUTFILE:=${INFILE}.md5}
```

En este ejemplo, el nombre del archivo de salida se setea a el nombre del archivo de entrada con la extensión .uu, si no se especifica el archivo de salida. Notar que se usa el comando : para prevenir que el shell ejecute el resultado de la substitución.

Cuando se aseguró que todos los argumentos son correctos, se ejecuta la acción:

```
md5sum $INFILE > $OUTFILE ;
```

Se debe chequear si el archivo de entrada es realmente un archivo antes de ejecutar el comando, con lo cual queda

```
if [ -f "$INFILE" ] ; then md5sum $INFILE > $OUTFILE ; fi
```

A este punto el script quedó funcional, sólo resta incluir el reporte para el caso de la opción verbose. Hay que cambiar la sentencia if a lo siguiente:

```
if [ -f "$INFILE" ] ; then
        if [ "$VERBOSE" = "true" ] ; then
                echo "Generando hash md5 de $INFILE en $OUTFILE... \c"
        fi
        md5sum $INFILE > $OUTFILE ; RET=$? ;

        if [ "$VERBOSE" = "true" ] ; then
                MSG="Failed" ;
                if [ $RET -eq 0 ] ; then MSG="Done." ; fi
                echo $MSG
        fi
```

```
fi
```

El script completo queda:

```
#!/bin/sh

USAGE="Usage: `basename $0` [-v] [-f] [filename] [-o] [filename]";
VERBOSE=false
while getopts f:o:v OPTION ; do
        case "$OPTION" in
                f) INFILE="$OPTARG" ;;
                o) OUTFILE="$OPTARG" ;;
                v) VERBOSE=true ;;
                \?) echo "$USAGE" ;
                        exit 1
                        ;;
        esac
done

shift `echo "$OPTIND - 1" | bc`
if [ -z "$1" -a -z "$INFILE" ] ; then
        echo "ERROR: Input file was not specified."
        exit 1
fi

if [ -z "$INFILE" ] ; then INFILE="$1" ; fi

: ${OUTFILE:=${INFILE}.md5}
if [ -f "$INFILE" ] ; then
        if $VERBOSE ; then
                echo "uuencoding $INFILE to $OUTFILE... \c"
        fi
        md5sum $INFILE $INFILE > $OUTFILE ; RET=$?
        if $VERBOSE ; then
                MSG="Failed" ; if [ $RET -eq 0 ] ; then MSG="Done." ; fi
                echo $MSG
        fi
fi

exit 0

easy-md5 ch11.doc
easy-md5 -f ch11.doc
easy-md5 -f ch11.doc -o ch11.md5
```

Resumen

En este capítulo se vio como tratar con argumentos y opciones en un script.

Específicamente se vieron los siguientes métodos:

- Manejar argumentos y opciones manualmente con un case
- Manejar argumentos y opciones con getopts

A lo largo de los ejemplos se vio la implementación y el racional detrás de cada método. Además, se vieron varias variables especiales referidas a argumentos y ejecución de comandos.

Como se verá más adelante el uso de opciones incrementa la flexibilidad y reusabilidad de los scripts de forma significativa.