PROGRAMACIÓN INTEGRATIVA

TUTORIAL SHELL SCRIPTING

En el siguiente documento se presenta un tutorial de cómo programar un pequeño *script* que, para cada uno de los usuarios que se encuentren en `/etc/passwd`, compruebe que el tamaño de su directorio no excede en bytes el tamaño indicado por línea de comandos. En caso de que el tamaño de directorio de alguno de los usuarios exceda este límite, se imprimirá un mensaje de error.

La estructura del tutorial se divide en 1) obtención de parámetros por línea de comandos, 2) implementación de la lógica de programa y 3) resultado final.

1. Obtención de parámetros

Como hemos comentado, a nuestro *script* vamos a tener que pasarle un argumento que es obligatorio. Una buena práctica para evitar un comportamiento anómalo, sería que el *script* se ejecutase si y sólo si obtiene el argumento deseado. Esto se podría hacer de forma posicional o usando opciones `getopts`. Con todo, otra buena práctica sería poner un valor por defecto en caso de que el usuario se haya olvidado de pasar este argumento. Una posible solución sería.

```
#!/bin/bash
MAXSIZE=16777216 # 16MiB
if [ $# != 1 ]; then
    echo "Error: wrong number of parameters"; exit 1;
else
    MAXSIZE=$1
fi
```

De esta manera, le decimos al usuario que el número de argumentos obligatorios no es el correcto. En caso de argumentos con alias, bash permite usar `getopts`, que es una forma fácil de parsear los argumentos de entrada utilizando un loop `while-do`. Con todo, la mayor limitación de `getopts` es que no existe una forma fácil de utilizar argumentos "largos" (e.g. `--verbose` en vez de `-v`). Un ejemplo de uso para nuestro caso:

```
#!/bin/bash

while getopts "s:" opt; do
```

```
  case $opt in
    s) MAXSIZE=$OPTARG ;;
    \?) echo "Invalid option: -$OPTARG" >&2 ;;
  esac
done
```

Una nueva posible mejora sería utilizar el comando `getopt` (no confundir con `getopts`). El problema que supone en cuanto a compatibilidad es que depende de que la herramienta esté instalada en el sistema, pero su ventaja es que extiende la funcionalidad de `getopts` y permite parsear opciones en su versión "larga". En el material de ampliación de la sesión 2 se describe con mayor detalle su uso.

2. Lógica del script

Una vez que nuestro script es capaz de entender los parámetros de entrada, el siguiente paso es implementar la funcionalidad requerida. El algoritmo se divide en los siguientes pasos:

1. Buscar todos los usuarios en `/etc/passwd`. Esto se puede realizar iterando línea a línea el fichero.

2. Recuperar el tamaño del directorio de cada usuario. Para ello habría que ejecutar el comando `du` en cada directorio de usuario.

3. Comprobar que el tamaño no excede el valor pasado al usuario. En caso de que exceda, imprimir por pantalla un mensaje de alerta.

Por ello, es necesario conocer la estructura del fichero `/etc/passwd`, para saber qué campos son los que debemos obtener. A modo de resumen, este fichero tiene una línea para cada usuario con campos separados por dos puntos (:). En nuestro caso nos interesa el nombre de usuario, que es el primer campo. La lista completa del significado de todos los campos se puede consultar en [1].

De esta manera, lo que tendremos que hacer es iterar línea a línea el fichero y recuperar el primer campo.

```
IFS=$'\n';
for l in $(cat /etc/passwd); do
    user=$(cut -d ":" -f 1 <<< $l);
    ...
done
```

Hemos utilizado la variable especial definida en POSIX `IFS` (Internal Field Separator, §2.4.1.3) para indicar que el separador de palabra es el salto de línea, de manera que el bucle `for` itere a nivel de línea. También es interesante la sintaxis `$'...'`. Sabemos que podemos utilizar las comillas simples para tratar el contenido que concierne de forma literal, pero si añadimos $ básicamente permitimos que bash pueda interpretar correctamente carácteres de escape como el salto de línea ($\backslash n$) o tabuladores ($\backslash t$).

De la misma forma, hemos recuperado el usuario utilizando el comando `cut` con el delimitador `:`, y recuperado el primer valor (`-f 1`). Es destacable el uso de la sintaxis `HERESTRING` (<<<), que permite pasar como string la línea leída al comando.

Ahora, utilizaremos el comando `du` para contar el espacio que ocupa el directorio de cada usuario utilizando la expresión `~$user`:

```
...
output=$(eval du -s \~$user); # aqui podemos anhadir 2> /dev/null
                              # para evitar mensajes de error por permisos
size=$(cut -d $'\t' -f 1 <<< $output);
...
```

Se utiliza `eval` para permitir alterar el orden normal de las sustituciones realizadas por el shell. Si ejecutásemos:

```
du -s ~$user
```

el shell intentaría resolver primero la virgulilla (ver Sección 2.5.5), provocando un error al no encontrar un usuario llamado '$user'. `eval` es un comando especial del shell que realiza las sustituciones pertinentes (virgulillas, paramétricas, de variables, de comandos, aritméticas) y reinicia nuevamente el proceso de ejecución descrito en la Sección 2.5.5. Utilizamos:

```
eval du -s \~$user
```

De este modo, en una primera ejecución (la de `eval`) el intérprete reconoce \~ como un caracter normal, y realiza la sustitución de `$user`. Tras ejecutar `eval` se realiza la ejecución de su resultado:

```
du -s ~usuario
```

Que era exactamente el resultado que estábamos buscando. La salida de `du -s <directory>` devuelve el tamaño del directorio dado en bytes. Así, de nuevo, utilizamos `cut` para parsear y recuperar la primera parte de la salida, que contiene este valor deseado.

Antes de comprobar si el tamaño es menor que el dado por línea de comandos, sería conveniente asegurarnos de que $output no es vacío. Una forma sencillo de comprobarlo es utilizando el test `-z`, que devuelve `true` si la variable es una cadena vacía, o está sin declarar o ambas:

```
...
output=...
if [ -z $output ]; then
    continue
fi;
```

Por último, sería necesario realizar la comprobación del tamaño y, en caso de que el tamaño sea mayor, imprimir un mensaje de error.

```
if [ $size -qt $MAX ]; then
cat <<EOF
Estimado $user,

Hemos detectado que su directorio HOME ubicado en $(eval echo \~$user)
supera el tamanho maximo permitido de $MAX bytes. Por favor, resuelvalo
a la mayor brevedad posible.
EOF
fi;
```

Donde hemos utilizado un HEREDOC (ver Sección 2.5.1.1) para imprimir la salida. En un sistema real se podría enviar un correo con herramientas como `mailx`, lo que se deja como ejercicio para el estudiante.

3. *Putting It All Together*[1]

```
#!/bin/bash

SCRIPTNAME="script-getopt.sh"
SCRIPTVERSION="v0.1"

# Print usage
usage() {
  echo -n "$SCRIPTNAME [OPTION]...
main options are described below.

 Options:
  -v,--verbose     Verbose output
  -d,--debug       Debug output
  -s,--size        Max size allowed in bytes
  -h,--help        Display this help and exit
  --version     Displays versions and exits"
}

version() {
  echo -n "$SCRIPTNAME $SCRIPTVERSION (C) 2019. No warranty."
}

# DEPENDENCY: getopt command tool
TEMP=‘getopt -o vds:h --long verbose,debug,size:,help,version \
             -n "$SCRIPTNAME" -- "$@"‘
# Note the quotes around ‘$TEMP’: they are essential!
eval set -- "$TEMP"

# default values for options
VERBOSE=false
DEBUG=false
MAXSIZE=16777216
while true; do
  case "$1" in
    -v | --verbose ) VERBOSE=true; shift ;;
    -d | --debug ) DEBUG=true; set -x; shift ;;
    -s | --size ) MAXSIZE="$2"; shift 2 ;;
    -h | --help ) usage; exit 1 ;;
    --version ) version; exit 1 ;;
    -- ) shift; break ;;
    * ) break ;;
  esac
done

# main loop
IFS=$’\n’;
for l in $(cat /etc/passwd); do
  user=$(cut -d ":" -f 1 <<< $l);
  size=$(eval du -s \~$user 2> /dev/null);
  s=$(cut -d $’\t’ -f 1 <<< $size);
  if [ -z $size ]; then
    continue
  fi;
  if [ $s -gt $MAXSIZE ]; then
    echo "[ERROR]   $user directory exceeds size $MAXSIZE"
  elif [ $VERBOSE == true ]; then
    echo "[SUCCESS] $user directory does not exceed size $MAXSIZE"
  fi;
done
```

[1]Con esta frase Hennessy y Patterson acaban muchos de los capítulos en “la biblia” del hardware Computer Architecture: A Quantitative Approach

Referencias

[1] nixCraft - Linux and Unix tutorials for new and seasoned sysadmin. *Understanding /etc/passwd File Format*. `https://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/`.