Administración de Sistema Operativos

Monitorización de cambios en tiempo real (inotify)

Jose Ricardo Peña Seco
Francisco Javier Dominguez Suarez
Daniel Hernández Sánchez

Introducción

Para el desarrollo de esta práctica hemos realizado dos scripts, ambos consistirán en la monitorización de una dirección pasada como parámetro, cada uno de los script realizan esta misma acción pero cada uno desarrollado de manera distinto, el primero de estos será una versión mínima de monitorización, seguido de una segundo script de monitorización ampliado, control de versiones y con diversas funcionalidades extra.

Realización Primer Script

Para poder realizar lo que explicamos en la introducción debemos de emplear el módulo de inotify para capturar de forma asíncrona los eventos asociados a la dirección que pasaremos como parámetro y guardaremos toda esta información en un fichero llamado en este caso fichero_registro .

Las primeras líneas de nuestro código consisten en la detección de errores y creación de ficheros para evitar errores, comprobando el número de parámetros pasados, el tipo del parámetro, además de comprobar que nuestro fichero de registro exista y si no lo creamos y por último de redirigir el parámetro introducido al output de errores por defecto (stderr). Aquí podemos ver la parte de corrección de errores:

```
#función que redirecciona el parámetro que le pasemos al stderr
error(){
      echo $1>&2
      exit 1
}

#revisar si el número de argumentos es igual a dos
if [[ $# -ne 2 ]]
then
      error "usage: script [directory] [register_file]"
fi

#comprobamos que exista el fichero de logs
if [[ ! -f $2 ]]; then
      touch $2
fi
```

```
#comprobamos que el directorio exista
if [[ ! -d $1 ]]
then
    error "Incorrect parameter [directory]: Is not a directory"
fi
```

Por otro lado tenemos la parte donde realizamos la monitorización, para realizarlo hemos decidido usar inotifywait. Con este podemos monitorizar cambios en un fichero particular o un directorio, incluso recursivamente sobre los subdirectorios que se encuentren por debajo lo cual es exactamente lo que necesitamos en este caso, aunque no vamos a usarlo recursivamente ya que a nosotros se nos pide monitorizar un directorio concreto y no sus subdirectorios, pero aun así con inotifywait podríamos hacerlo también.

Usaremos la opción -m por una parte para especificar que monitorice el directorio que le pasemos como parámetro. Además de la opción -e para filtrar los eventos que queremos monitorizar, en este caso, create y modify para cuando se cree un fichero o se edite uno de los ya existentes.

Usando una tubería vamos a introducir los datos de los eventos ocurridos en un while en segundo plano que leerá directorio, evento, fichero y calculará la fecha usando la función date con formato “%H:%M %d/%m/%y”, siendo así por ejemplo una fecha: 16:55 30/12/2022.

Para finalizar el script guardaremos los datos de directorio, evento, fichero y fecha redirigiendo la salida de la función echo al fichero_registro con un >>.

```
#usamos el comando inotifywait de inotify para realizar acciones cada
vez que haya un cambio en el directorio en segundo plano.
inotifywait -m $1 -e create,modify | while read DIRECTORY EVENT FILE;
do
    DATE=$(date +"%H:%M %d/%m/%y")
    echo $DATE, $DIRECTORY, $EVENT, $FILE >> $2
done &

echo "Changes observer initialized with PID: $!"
```

Segundo Script:

Una vez realizada la explicación del primer script, pasaremos a repasar el código del segundo script. En este se nos ha solicitado que haciendo uso de la monitorización que vimos en el primer script creamos un sistema de control de versiones minimalista. Nosotros hemos creado una utilidad llamada pic (de picture) que nos servirá para guardar una serie de snapshots de nuestro directorio con los cambios leídos por inotify. Además se han implementado una serie de funciones para trabajar con este sistema. Cómo por ejemplo recuperar el estado del directorio en un punto, ver una lista de los cambios que se han realizado etc… Cómo podemos apreciar a continuación nos hemos inspirado en git para la realización de este trabajo.

Comandos:

Para comenzar examinando el código en mayor detalle, vamos a analizar los comandos y opciones que ofrece nuestro script, por eso, empezaremos a examinar nuestro case, que a partir de nuestro primer parámetro de entrada nos va a redirigir a la función que necesitemos en cada caso, si $1 no está entre las opciones, te llevará a la función default() que analizaremos más tarde.

```
case "$1" in
    -v | --version )
        version
    ;;
    -h | --help )
        base_help
    ;;
    watch )
        watch
    ;;
    log )
        log
    ;;
    snap )
        snap
    ;;
    status )
        status
```

```
    ;;
    goto )
        goto
    ;;
    init )
        init $2
    ;;
    help )
        help $2
    ;;
    * )
        default
    ;;
esac
```

En primer lugar, tenemos la opción versión (-v o –version), la cual consiste en la impresión por pantalla de la versión del script que estamos usando.

```
version(){
    echo "pic version 0.0.1"
}
```

En el caso de la siguiente opción, tenemos -h o –help, el cual nos ayudará a entender mejor cómo usar el script con una breve descripción de cada opción, obviamente si utilizamos la opción help, en lugar -h obtendremos una ayuda mucho más amplia y que explicaremos más adelante.

```
base_help(){
    echo "Usage:"
    echo ""
    echo "  pic [--version] [--help] <command> [<args>]"
    echo ""
    echo "These are the implemented commands:"
    echo ""
    echo "  init    Create a control version repository"
    echo "  watch   Make the control version work in current directory"
    echo "  log     Show the changes history of the directory"
    echo "  status  See changes in the directory"
    echo "  snap    Create a snapshot with current changes"
    echo "  goto    Recover the state of the directory in a given
snapshot"
```

```
    echo "  help    get help with a concrete command"
    echo ""
    echo "You can use pic --help or pic --version to get additional
information"
}
```

Además, tenemos la opción init, que será la primera que debemos utilizar, ya que esta se encarga de iniciar el control de versiones en un directorio pasado por parámetro creando un repositorio pic, aunque hasta que no realicemos la opción watch (que por detrás usa inotify), no empezará de verdad a trackear los cambios del directorio y por ende poder hacer control de versiones. La implementación consiste primero en revisar mediante la función is_repository_error que se encargará de comprobar si nuestro parámetro es un repositorio pic, a partir de ahí, simplemente crearemos los ficheros necesarios para el control de versiones y el resto de opciones e imprimimos un mensaje por pantalla si todo ha salido bien.

```
init(){
    repo=$1
    is_repository_error $repo
    mkdir $repo/.pic
    mkdir $repo/.pic/versions
    touch $repo/.pic/logs
    touch $repo/.pic/snap
    echo "Initialized a pic repository in $repo"
}

is_not_repository_error(){
    if [[ ! -d $1/.pic ]]
    then
        error "$1 is not a pic repository"
    fi
}
```

Nuestra función watch() está desarrollada de manera muy parecida al primer script, ya que esta parte se va a encargar mediante la función inotifywait de controlar los cambios en el directorio pasado por parámetro exactamente como en el primer script, pero en este caso también tenemos en cuenta la eliminación de ficheros, ya que en este caso al realizar un control de versiones,

también deberemos de tener en cuenta esto. Además, de guardar estos cambios esta vez en el fichero creado de /.pic/snap que nos servirá más adelante con otras funciones y la comprobación mediante la función is_not_repository_error .

```
watch(){
    is_not_repository_error .
    inotifywait -m . -e create,modify,delete | while read DIRECTORY
EVENT FILE; do
            DATE=$(date +"%H-%M-%d-%m-%y")
            echo $DATE:$DIRECTORY:$EVENT:$FILE >> ./.pic/snap
        done &
    echo "Started watching changes in $repo with PID: $!"
}
```

Además tenemos la opción status, que en este caso consiste de un simple cat del fichero que creamos previamente con los cambios que se han realizado en el directorio.

```
status(){
    is_not_repository_error .
    #implement -file optional parameter
    echo "The changes of the directory are:"
    cat ./.pic/snap
}
```

La función snap se encarga de registrar y almacenar los cambios realizados en el directorio durante una sesión de trabajo, esta información se encuentra en el fichero snap que contiene los cambios realizados en el directorio. Su objetivo es similar al objetivo de un Commit en Git, una vez se ha ejecutado el comando watch, se puede llamar a la función snap para dejar constancia en el fichero de logs de los archivos modificados, además se guardará en el directorio versions una copia de tipo tar de los archivos que han cambiado usando como etiqueta la fecha y hora del snap realizado, además se guardarán en el fichero deletions los archivos eliminados durante ese snap.

```
snap(){
    is_not_repository_error .
    # Se almacena la fecha de la captura y se obtiene la información
    del fichero snap
    DATE=$(date +%Y%m%d%H%M%S)
    list=$(cat ./.pic/snap | cut -d: -f4 | sort -u | grep -v "^\.")
```

```
    if [ -z"$list" ]
    then echo "No hay cambios que registrar"
    exit 1
    fi

    # Se obtiene la lista de ficheros borrados
    cat /dev/null > ./.pic/deletions
    cat ./.pic/snap | cut -d: -f3,4 | grep '^DELETE'| cut -d: -f2 |
sort -u >> ./.pic/deletions

    # Los archivos modificados se utilizan como parámetros del comando
    tar
    input="./.pic/deletions"
    for i in $list; do
            input="$input $i"
        done
    tar czf ./.pic/versions/$DATE.tgz $input 1 > /dev/null 2> /dev/null

    #Se escribe en el ./.pic/log el snap y los archivos modificados
    echo "========================================" >> ./.pic/logs
    echo "Snap with ID: $DATE.tgz added to versions " >> ./.pic/logs
    echo "  Files changed:" >> ./.pic/logs
    for i in $list; do
            echo "    ._$i" >> ./.pic/logs
        done

    # Se muestra un mensaje por la consola del snap creado y su ID
    echo "Created a snap with ID: $DATE"
    echo "  Files changed:"
    for i in $list; do
        aux=$(grep "$i" ./.pic/deletions)
        if [ "$i" == "$aux" ]; then
            echo "    --"$i""
        else
            echo "    ++"$i""
        fi
        done

    # Se vacía el fichero snap
    cat /dev/null > ./.pic/snap
}
```

Por otro lado, también hemos implementado la función de log(), para conocer las diferentes versiones del directorio que tenemos (este comando es equivalente al git log de git). Esta función hace simplemente un cat del .pic/logs ya que en el comando snap hemos ido guardando las diferentes versiones en formato de texto dentro de ese fichero.

```
log(){
    #implement -file optional parameter
    cat ./.pic/logs
}
```

El goto es un comando que nos servirá para recuperar el directorio en un determinado periodo de tiempo. Para ello simplemente le pasaremos una fecha en este formato YYYYMMDDTHH:MM:SS y si existe un snap nos llevará el directorio a ese estado. Para ello recursivamente iremos recorriendo los diferentes tars que hemos creado en el snap, descomprimiendolos hasta que lleguemos al snap pedido. De esa forma el directorio se quedará en el estado de esa fecha concreta.

```
goto(){
    #queda por implementar
    echo "Is not implemented yet"
}
```

Cabe destacar que al igual que el primer script tenemos partes de detección de errores, aunque en este caso en forma de funciones las cuales ya hemos analizado una por una a medida que las hemos ido utilizando para programar los comandos. Sin embargo, en este script hemos desarrollado un caso de help() para ayudar a comprender mejor el código y el uso de sus opciones. La opción de help, contiene a su vez de otro case que nos permite mediante el parámetro de entrada introducido con el help() decidir que vamos a imprimir por pantalla, cada una de las opciones explica el funcionamiento y uso de cada una de las opciones, además de comprobar que si no es ninguna de las opciones que hemos desarrollado en pic y mandar el mensaje correspondiente a este error.

```
help(){
    case "$1" in
        init )
```

```
            echo "Manual of init:"
            echo ""
            echo "Description:"
            echo ""
            echo "init is used to make a directory a pic repository."
            echo "If you want to track the changes of a directory you
first must"
            echo "create inside a pic repository."
            echo "this will create a .pic folder in your dir which is
used internally"
            echo "to save all the snapshots and internal stuff"
            echo ""
            echo "Usage:"
            echo ""
            echo "  pic init [directory]"
            echo ""
        ;;
        watch )
            echo "Manual of watch:"
            echo ""
            echo "Description:"
            echo ""
            echo "watch is used to start tracking changes of a
repository"
            echo "if you modify your repository without watching it the
control version"
            echo "will not track the changes of the directory"
            echo ""
            echo "Usage:"
            echo ""
            echo "  pic watch"
            echo ""
            echo "By default gets the current directory"
        ;;
        log )
            echo "Manual of log:"
            echo ""
            echo "Description:"
            echo ""
            echo "Log is used to see the history of changes of the
current directory"
            echo ""
            echo "Usage:"
```

```
            echo ""
            echo "  pic log [-file <file>]"
            echo ""
            echo "The current directory must be a pic repository."
        ;;
        goto )
            echo "Manual of goto:"
            echo ""
            echo "Description:"
            echo ""
            echo "Change the current directory state to a past
snapshot"
            echo ""
            echo "Usage:"
            echo ""
            echo "  pic goto <file> <date>"
            echo ""
            echo "The current directory must be a pic repository."
        ;;
        snap )
            echo "Manual of snap:"
            echo ""
            echo "Description:"
            echo ""
            echo "Create a snapshot with the changes of the directory"
            echo ""
            echo "Usage:"
            echo ""
            echo "  pic snap [-m <message>]"
            echo ""
            echo "The current directory must be a pic repository."
        ;;
        status )
            echo "Manual of status:"
            echo ""
            echo "Description:"
            echo ""
            echo "See the current changes in the directory"
            echo ""
            echo "Usage:"
            echo ""
            echo "  pic status"
            echo ""
```

```
            echo "The current directory must be a pic repository."
        ;;
        * )
            error "No manual entry for that command or the command does
not exist"
        ;;
    esac
}
```

Tercer script:

Por último, al haber realizado un script mejorado de la segunda parte no queríamos quedarnos sin abarcar algunas de las tareas marcadas en la práctica, es decir, que hemos desarrollado un tercer script siguiendo las indicaciones dadas en la práctica a rajatabla.

Al igual que desarrollamos en los demás script, nuestro código comienza con corrección de errores, en este caso, la función error() que ya hemos comentado en los anteriores script, encargándose de redirigir nuestro contenido a la salida por defecto de los errores (stderr). Además, comprobamos que los parámetros pasados sean el número deseado, después de realizar esta comprobaciones, recorremos los directorios pasados y comprobamos que sean directorios y también verificar que la carpeta versions exista, si esta no existe, nuestro código se encargara de crearlo.

```
#!/bin/bash

# Monitorización de cambios en tiempo real (icron/inotify):
#
#   Se trata de llevar un registro automático de versiones de los
archivos de un directorio. Se elaborará
#   un script que reciba como argumento uno o varios directorios. Sobre
cada directorio, mediante
#   inotify se detectarán las modificaciones en todos sus ficheros, que
se irán guardando en una carpeta
#   ".versiones" del directorio. La versión del archivo A se guardará
como
#   A.YYYYMMDDTHH:MM:SS (o sea, el nombre del archivo, seguido de una
marca de tiempo con
```

```
#   formato estándar). Se valorará positivamente que se elabore alguna
utilidad para recuperar el estado
#   del directorio en una fecha y hora determinadas.
#
# Uso:
#
#    script [directory...]
#
#    en caso de que alguno de los parametros no sea un directorio
saltara error
#

#función que redirecciona el parámetro que le pasemos al stderr
error(){
    echo $1>&2
    exit 1
}

#revisar si el número de argumentos es mayor a 1
if [[ $# -lt 1 ]]
then
    error "usage: script [directory...]"
fi
```

Una vez visto que todo está en orden, podemos pasar a realizar el control de versiones, en este caso usaremos el inotify de la misma forma que lo hemos hecho en los anteriores script, pero esta vez en nuestra *pipe* vamos a cambiar el algoritmo del while. En este caso, al tratarse de un control de versiones, debemos guardar el fichero en su última versión cada vez que pase un evento en nuestra carpeta de versión, usando cp. El resto del algoritmo será igual al de nuestro primer script.

```

#recorremos toda la lista de argumentos
for x in $@
do

    #revismos que el argumento sea un directorio
    if [[ ! -d $x ]]
    then
        error "Parameter must be a directory $x"
    fi

    #sino existe la carpeta .versions la creamos
```

```
    if [[ ! -d $x/.versions ]]
    then
        mkdir $x/.versions
    fi

    #usamos el comando inotifywait de inotify para realizar acciones
cada vez que haya un cambio en el directorio en segundo plano.
    inotifywait -m $x -e create,modify | while read DIRECTORY EVENT
FILE; do
        DATE=$(date +%Y%m%d%H:%M:%S)
        cp $DIRECTORY/$FILE $x/.versions/$FILE.$DATE
    done &

    echo "Changes observer in directory $x initialized with PID: $!"
done
```