Índice

1. Hipótesis y Aclaraciones

Si al comando Mover se le pasa como destino un directorio, deberá mover a éste el archivo de origen con su mismo nombre.

En los directorios utilizados para archivos de datos (`$DATADIR` y sus subdirectorios), no deben existir archivos ordinarios con el nombre "dup", ya que no permitiría la creación de secuencias de duplicados (para lo cual `dup` deber ser un directorio). En rigor de verdad esta situación podría manejarse, ya sea eliminando los archivos ordinarios de nombre "dup"para crear directorios en su lugar, o descartando los archivos duplicados. Consideramos que no es correcto no almacenar los duplicados, y además no es necesario tener archivos de nombre "dup", debido a que los archivos de entrada tienen nombres con estructuras definidas que no permiten a "dup"ser un archivo de entrada válido.

La correcta inicialización del entorno queda indicada mediante una variable definida a tal propósito (`$ENTORNO_INICIALIZADO`). Basta con chequear desde un comando la existencia de esta variable para asumir un entorno de trabajo válido.

La ruta relativa de `practico.conf` respecto del comando iniciar debe ser: `../conf/practico.conf`. De esta forma se asegura que iniciar pueda encontrarlo y a partir de él recibir cualquier otra información sobre rutas que necesite para continuar su correcta ejecución.

DETALLAR HIPÓTESIS ASUMIDAS SOBRE LA ESTRUCTURA DE DIRECTORIOS PARA PERMITIR A INSTALAR ENCONTRAR `practico.conf`.

2. Problemas relevantes

DETALLAR PROBLEMAS ENCONTRADOS DURANTE EL DESARROLLO Y LAS PRUEBAS.

3. Archivo README: Instructivo de instalación

Cuando esté terminado el README insertar en el `.tex` según su ruta haciendo:

```
\verbatiminput{ruta/README}
```

4. Comandos Desarrollados

4.1. Comando Mover

Tipo de comando: Solicitado

Archivos de entrada: Archivo especificado por el parámetro 1

Archivos de salida: Archivo o directorio especificado por el parámetro 2; archivo de log del comando que lo invoca: `$LOGDIR/comando.log`

Parámetros:
1. Archivo de origen a mover
2. Archivo o directorio de destino
3. (Opc.) Comando que invoca a mover, permite guardar información en su log

Ejemplos de invocación: Sea `a1` la ruta a un archivo existente, `d1` un directorio existente, invocando como:

```
$ mover a1 d1
```

el comando mueve el archivo a1 al directorio `d2`, informando que la operación es exitosa. Por mover se entiende que cambia el directorio padre del archivo origen.
Sea `a1` un archivo existente, `d2` una ruta inválida, invocando como:

```
$ mover a1 d2
```

el comando `mv`, invocado por mover, informa que el directorio de destino no existe.
Sea `a1` y `ruta/a2` archivos existentes, donde `ruta/dup` no existe, invocando como:

```
$ mover a1 ruta/a2
```

el comando crea en ruta el subdirectorio `dup`, y mueve allí el archivo `a1` con nombre `a2.1`. Este representa el primer duplicado de `a2`. Si se vuelve a invocar como:

```
$ mover ax ruta/a2
```

el comando mueve `ax` a `ruta/dup/a2.2`. Así va formando una secuencia de archivos duplicados para no permitir la sobreescritura.
Sea `a1` un archivo existente, `ruta/a1` otro archivo existente, invocando como:

```
$ mover a1 ruta
```

el comando intenta mover `a1` a `ruta/a1`; al existir este archivo, se añade un duplicado a la secuencia de `a1` (o se la inicia si no existen duplicados previos).
Sea `a3` una ruta inválida, `a4` una ruta válida o inválida, invocando como:

```
$ mover a3 a4 com1
```

el comando informa que el archivo de origen es inexistente. Además, por haber incluído el tercer parámetro, el comando guarda en `$LOGDIR/com1.log`, a través del comando `glog` todos los mensajes de información y/o error que, al igual que en las demás ejecuciones, también envía por la salida estándar.

Código fuente:

```
#!/bin/bash

# Codigos de error
ENOARGS=65
ENOFILE=66

ENOENTORNO=14

# Funcion para imprimir por consola y al log
# Uso: imprimir mensaje [tipo]
imprimir() {
    echo $1

    if [ $USAR_LOG ]; then
        glog "$COMANDO" $2 "$1"
    fi
```

```
}

# Funcion para determinar el orden de secuencia
det_sec() {
    SEC=1

    for ARCH in "$DIRDEST/dup/$(basename $DEST)".*
    do
        # Se extrae la secuencia de los duplicados existentes
        TMP=$(expr "$ARCH" : '.*\(\.[0-9]*\)')
        TMP=${TMP:1}

        # Se incrementa en uno el valor actual de la secuencia
        if [ "$TMP" -ge "$SEC" ] &>/dev/null; then
            SEC=$((TMP+1))
        fi
    done

    imprimir "Mover: Duplicado de $(basename "$DEST") numero $SEC" I
}

if [ ! $ENTORNO_INICIALIZADO ]; then
    imprimir "Mover: Entorno no inicializado"
    exit $ENOENTORNO
fi

# Se chequean parametros obligatorios
if [ $# -lt 2 -o $# -gt 3 ]; then
    imprimir "Uso: ‘basename $0‘ origen destino [comando]" SE
    exit $ENOARGS
fi

if [ $# -eq 3 ]; then
    USAR_LOG=1
    COMANDO=$3
fi

# Si el destino es igual al origen no se debe hacer nada
if [ "$1" = "$2" ]; then
    imprimir "Mover: Archivo de destino igual a archivo de origen" W
    exit $ENOARGS
fi

# Se verifica que el origen exista
if [ ! -e "$1" ]; then
```

```
    imprimir "Mover: $1 no es un archivo de origen valido" E
    exit $ENOFILE
fi

DEST="$2"

# Si el destino es un directorio se asume copiar con igual nombre
if [ -e "$2" -a -d "$2" ]; then
    DEST="${2%/}/$(basename "$1")"
fi

# Si no es duplicado se mueve y se termina la ejecucion
if [ ! -e "$DEST" ]; then
    mv "$1" "$DEST"
    if [ $# -eq 3 ]; then
        imprimir "Mover: exitoso de $1 a $DEST" I
    fi
    exit 0

# Si es duplicado se gestiona la secuencia
else
    DIRDEST="$(dirname "$DEST")"

    # Si no existe se crea el directorio de duplicados
    if [ ! -d "$DIRDEST"/dup ]; then
        mkdir "$DIRDEST"/dup
    fi

    # Se determina el numero de secuencia siguiente
    det_sec
    # Se mueve la version que se queria copiar a /dup
    mv "$1" "$DIRDEST/dup/$(basename "$DEST").$SEC"
fi

exit 0
```

4.2. Comando Iniciar

Tipo de comando: Solicitado

Archivos de entrada: El archivo de configuración `practico.conf`

Archivos de salida: El archivo de log `$LOGDIR/iniciar.log`

Ejemplo de invocación: El comando iniciar debe invocarse como

```
$ . iniciar
```

El comando chequeará la presencia en el archivo de configuración `practico.conf` de las variables indispensables para el correcto funcionamiento de todos los comandos. Solicitará parámetros necesarios para el comando detectar: cantidad máxima de ciclos de ejecución y tiempo de espera entre ciclos, expresado en minutos. También dará la posibilidad de ejecutar detectar a continuación, o bien recibir indicaciones de cómo hacerlo manualmente. La ejecución de este comando deja inicializado el entorno de trabajo para la sesión de bash actual. Cualquier otra sesión de bash en que se quieran utilizar los comandos del trabajo, no representará un entorno válido si no se ejecuta en ella el comando iniciar.

Código fuente:

```
#! /bin/bash

# Codigos de error
EEXISTEENTORNO=14
ENOINSTALADO=15

# Funcion para imprimir por consola y al log
# Uso: imprimir mensaje [tipo]
imprimir() {
    echo $1
    if [ $ENTORNO_INICIALIZADO ]; then
        glog "$INICIAR" $2 "$1"
    fi
}

# Variables que deben estar definidas en practico.conf
VARNECESARIAS=( \
[0]=BINDIR \
[1]=LOGDIR \
[2]=DATADIR \
[3]=MAXLOG \
[4]=GRUPO )

# El entorno puede inicializarse una sola vez en cada sesion de bash
if [ $ENTORNO_INICIALIZADO ]; then

    imprimir "Invocacion con el entorno ya inicializado" W
```

```
    echo "Valores de las variables de entorno indispensables"
    for VAR in ${VARNECESARIAS[*]}
    do
        echo \$$VAR = $(eval echo \$$VAR)
    done
    echo \$ARCHCONF = $ARCHCONF
    echo \$CANLOOP = $CANLOOP
    echo \$TESPERA = $TESPERA

    echo

    # Se muestra el PID de detectar, si es que esta corriendo
    DETPID=$(expr "$(ps | grep detectar)" : '\ \([^\ ]*\)\ .*')
    if [ ! -z "$DETPID" ]; then
        imprimir "PID de detectar: $DETPID"
    fi

    return $EEXISTEENTORNO
fi

# la ubicacion de practico.conf relativa a iniciar es fija
export ARCHCONF=../conf/practico.conf

# El archivo de configuracion practico.conf debe existir
if [ ! -e $ARCHCONF ] || [ -d $ARCHCONF ]; then
    imprimir "Archivo de configuracion $ARCHCONF inexistente, instalacion corrupta"
    return $ENOINSTALADO
fi

# Se exportan las variables del archivo de configuracion
LINEAS=$(wc -l <$ARCHCONF)
(( LINEAS += 1 ))
CUENTA=0
{ # Se procesa el archivo de configuracion linea por linea
while [ $CUENTA -lt $LINEAS ]
do
    (( CUENTA += 1))

    # Se lee linea por linea el archivo de configuracion
    read LINEA
    # Si la linea esta vacia o comentada se saltea
    if [ ${#LINEA} -lt 1 -o "${LINEA:0:1}" = "#" ]; then
        (( REGISTROS -= 1 ))
        continue
    fi
```

```
    # Se exporta la variable definida en la linea
    export "$LINEA" #2>/dev/null
    if [ ! $? -eq 0 ]; then
        imprimir "Error de sintaxis en $ARCHCONF en la linea $CUENTA"
    fi
done
} <$ARCHCONF

# Se chequea que el archivo practico.conf este completo
for VAR in ${VARNECESARIAS[*]}
do
    # Referencia indirecta a la variable cuyo nombre es $VAR
    if [ -z "$(eval echo \$$VAR)" ]; then
        imprimir "No existe la de entorno $VAR"
    fi
done

# Se agrega el directorio bin al PATH para poder ejecutar las demas funciones
export PATH="$PATH:$BINDIR"
# Se exporta esta variable para poder invocar a glog. Si la ejecucion
#+de iniciar es erronea ENTORNO_INICIALIZADO se "unsetea".
export ENTORNO_INICIALIZADO=1

imprimir "Inicializando entorno"

# Se verifica la instalacion: presencia de tablas y ejecutables
# Array de nombres de archivos indispensables. Ejecutables:
NECESARIOS=( [0]="$GRUPO/bin/mover" \
[1]="$GRUPO/bin/glog" \
[2]="$GRUPO/bin/vlog" \
[3]="$GRUPO/bin/iniciar" \
[4]="$GRUPO/bin/detectar" \
[5]="$GRUPO/bin/interprete" \
[6]="$GRUPO/bin/reporte" \
[7]="$GRUPO/conf/p-s.tab" \
[8]="$GRUPO/conf/T1.tab" \
[9]="$GRUPO/conf/T2.tab" \
[10]="$GRUPO/conf/practico.conf" \
)

unset NOINST
# ${!a[*]} expande a todos los Ãndices del array a.
# No se puede usar for arch in ${a[*]}, que directamente iterarÃa en todos los
# elementos del array porque los elementos pueden contener espacios.
```

```
for i in ${!NECESARIOS[*]}
do
    if [ ! -e "${NECESARIOS[i]}" ]; then
        imprimir "El archivo indispensable ${NECESARIOS[i]} no esta presente" SE
        NOINST=1
    fi
done

if [ $NOINST ]; then
    imprimir "Imposible iniciar el entorno: Instalacion fallida o corrupta" SE
    unset ENTORNO_INICIALIZADO
    return $ENOINSTALADO
fi

# Se solicita por pantalla la cantidad de ciclos de detectar
while [[ "$CANLOOP" -lt 1 ]] 2>/dev/null
do
    echo -n "Cantidad de Ciclos de DETECTAR? (100 ciclos) "
    read CANLOOP

    if [ ${#CANLOOP} -eq 0 ]; then
        CANLOOP=100
    fi

    # Workaround para obligar a que sea un entero
    let CANLOOP="$CANLOOP"*1 2>/dev/null
done

# Se solicita por pantalla el tiempo de espera por ciclo de detectar
while [[ "$TESPERA" -lt 1 ]] 2>/dev/null
do
    echo -n "Tiempo de espera entre ciclos? (1 minuto) "
    read TESPERA

    if [ ${#TESPERA} -eq 0 ]; then
        TESPERA=1
    fi

    # Workaround para obligar a que sea un entero
    let TESPERA="$TESPERA"*1 2>/dev/null
done

# Se cambia la unidad de TESPERA de minutos a segundos
(( TESPERA *= 60 ))
```

```
# Se exportan las variables necesarias para ejecutar detectar
export CANLOOP
export TESPERA

# Se da la opcion de iniciar detectar si no se esta ejecutando
if [ $(ps -e | grep detectar | wc -l) -eq 0 ]; then
    while [[ "$OPCDETECTAR" != [snSN] ]]
    do
        echo -n "Â¿Desea efectuar la activacion de DETECTAR? (S/N) "
        read OPCDETECTAR
    done

    if [[ "$OPCDETECTAR" = [sS] ]]; then
        detectar >/dev/null v&
        imprimir "Demonio corriendo bajo el no.: $!"
    else
        echo "Para realizar la activacion del comando DETECTAR en segundo plano"
        echo "debe ingresar \"detectar &\" en una consola con el entorno iniciado."
    fi
fi

imprimir "Entorno inicializado correctamente, iniciar terminado"
```

4.3. Comando Detectar

Tipo de comando: Solicitado

Archivos de entrada: Archivos ubicados en el directorio `$DATADIR`

Archivos de salida: Archivos de `$DATADIR` que cumplen con el criterio de validez de nombres

Ejemplos de invocación: El comando solo puede invocarse como

```
$ detectar
```

de esta manera, lo que hace es verificar la validez de los archivos ubicados en el directorio `$DATADIR`, para así colocar los correctos como archivos de entrada del intérprete en `$DATADIR/ok`, y los incorrectos en `$DATADIR/nok`.

Código fuente:

```
#! /bin/bash

ENOENTORNO=14
ENODATADIR=15
ENODIR=16
ENOFORMATO=17

ERRDESC=( \
[$ENOENTORNO]="Entorno no inicializado" \
[$ENODATADIR]="Directorio de entrada inexistente" \
[$ENODIR]="Directorio inexistente" \
[$ENOFORMATO]="Error de formato" )

PSTAB="$BINDIR/../conf/p-s.tab"    # Tabla de Paises y Sistemas
DIROK="$DATADIR/ok"                # Directorio de aceptados
DIRNOK="$DATADIR/nok"              # Directorio de rechazados

# Funcion para imprimir por consola y al log
# Uso: imprimir mensaje [tipo]
imprimir() {
    echo $1
    glog "detectar" $2 "$1"
}

# Funcion para contar las instancias de un proceso dado su nombre
# Uso: contar_instancias proceso
# Almacena la cantidad contada en CANTINST
contar_instancias() {
    if [ -z $1 ]; then
        return
    fi

    # De los procesos existentes, se cuentan las instancias del buscado
    CANTINST=$(ps -e | grep "$1" | wc -l)
    # Debe restarse uno porque el subshell abierto incrementa la cantidad de instancias
    if [ "$1" = detectar ]; then (( CANTINST -= 1 )); fi
}

# Funcion para leer la tabla de archivos y sistemas
leer_paises_y_sistemas() {

    REGISTROS=$(wc -l <"$PSTAB")
    CUENTA=0
    # Se procesa cada linea de la tabla
    {
```

```
    while [ $CUENTA -lt $REGISTROS ]
    do
        # Se lee un registro de la tabla
        read LINEA
        # Si la linea esta vacia o comentada se saltea
        if [ ${#LINEA} -lt 1 -o "${LINEA:0:1}" = "#" ]; then
            (( REGISTROS -= 1 ))
            continue
        fi

        # Se extrae el codigo de pais
        PAISES[$CUENTA]=$(expr "$LINEA" : '\([^-]*\)')
        # Se extrae el codigo de sistema
        SISTEMAS[$CUENTA]=$(expr "$LINEA" : '[^-]*-[^-]*-\([^-]\)')

        (( CUENTA += 1 ))
    done
    } < "$PSTAB"
}

# Funcion para validar los nombres de archivos de entrada
# Uso: validar_nombre archivo
# Formato de nombre de archivo: <pais>-<sistema>-<anio>-<mes>
validar_nombre() {
    if [ -z $1 ]; then
        return
    fi

    PAIS=$(expr "$1" : '\([^-]*\)')
    SISTEMA=$(expr "$1" : '[^-]*-\([^-]*\)')
    ANIO=$(expr "$1" : '[^-]*-[^-]*-\([^-]*\)')
    MES=$(expr "$1" : '.*-\(.*\)')

    # Se comprueba la validez del anio
    ANIOACTUAL="$(date +%Y)"
    MESTOPE=12
    if [ "$ANIO" -lt 2000 -o "$ANIO" -gt "$ANIOACTUAL" ]; then
        false
        return
    fi
    # Si el anio es el actual, el mes tope es el actual
    if [ "$ANIO" -eq "$ANIOACTUAL" ]; then
        MESTOPE=$(date +%m)
    fi
    # Se comprueba la validez del mes
```

```
    if [ "$MES" -lt 1 -o "$MES" -gt "$MESTOPE" ]; then
        false
        return
    fi

    # Se verifica que el pais y el sistema sean validos
    CUENTA=0
    while [ $CUENTA -lt $REGISTROS ]
    do
        # Si existe el par Pais-Sistema, el archivo es valido
        if [ "$PAIS" = "${PAISES[$CUENTA]}" \
            -a "$SISTEMA" = "${SISTEMAS[$CUENTA]}" ]; then
            true
            return
        fi
        (( CUENTA += 1 ))
    done

    false
}

if [ ! $ENTORNO_INICIALIZADO ]; then
    echo "Detectar: Entorno no inicializado"
    exit $ENOENTORNO
fi

# Solo se permite una instancia activa de este script.
contar_instancias detectar
if [ $CANTINST -gt 1 ]; then
    imprimir "Detectar ya esta corriendo" W
    exit 0
fi

imprimir "Detectar iniciado" I

# El directorio de entrada de archivos debe existir
if [ ! -d "$DATADIR" ]; then
    imprimir "El directorio de entrada $DATADIR no existe" SE
    exit $ENODATADIR
fi

# Los directorios donde se moveran los archivos, de no existir, se crean
if [ ! -d "$DIROK" ]; then
    # Si existe un archivo con el nombre de uno de los directorios
    #+se esta ante un error severo.
```

```
    if [ -e "$DIROK" ]; then
        imprimir "El directorio de aceptados $DIROK no es un directorio" SE
        exit $ENODIR
    fi
    mkdir "$DIROK"
fi
if [ ! -d "$DIRNOK" ]; then
    if [ -e "$DIRNOK" ]; then
        imprimir "El directorio de aceptados $DIROK no es un directorio" SE
        exit $ENODIR
    fi
    mkdir "$DIRNOK"
fi

CICLOS=0

# Se lee por unica vez la tabla de paises y sistemas,
#+por lo tanto no puede cambiar durante la ejecucion
leer_paises_y_sistemas

# Bucle principal del demonio
while [ 1 ]
do
    (( CICLOS += 1 ))
    imprimir "Ciclo #$CICLOS"

    # Se procesan los archivos del directorio de entrada
    LISTA=$(ls -1 "$DATADIR")
    ARCHIVOS=( $LISTA ) # Array que contiene cada linea de la lista como un elemento
    # Para cada elemento del directorio se realiza el procesamiento necesario
    for ARCHIVO in ${ARCHIVOS[*]}
    do
        # No se procesan directorios
        if [ -d "$DATADIR/$ARCHIVO" ]; then continue; fi

        # Segun la validez del nombre, se acepta o rechaza el archivo
        if validar_nombre $ARCHIVO; then
            mover "$DATADIR/$ARCHIVO" "$DIROK" detectar
            imprimir "Recibido: $ARCHIVO"
        else
            mover "$DATADIR/$ARCHIVO" "$DIRNOK" detectar
            imprimir "Rechazado: $ARCHIVO"
        fi
    done
```

```
    # Variable que indica si entre este ciclo y el siguiente el
    #+demonio duerme o no. Si hay archivos en el directorio de
    #+validos, no se dormira para permitir su procesamiento inmediato.
    DORMIR=1
    # Si hay archivos de entrada validos se invoca al interprete
    LISTA=$(ls -1 "$DIROK")
    ARCHIVOS=( $LISTA )
    if [ ${#ARCHIVOS[*]} -gt 0 ]; then
        DORMIR=0
        # Solo puede correr una instancia del interprete
        contar_instancias "interprete"
        if [ $CANTINST -lt 1 ]; then
            #interprete
            if [ $? -eq 0 ]; then
                imprimir "PID de interprete: $!"
            else
                imprimir "Error de interprete #$?: ${ERRDESC[$?]}"
            fi
        fi
    fi

    # Se sale al alcanzar la cantidad maxima de ciclos de ejecucion
    if [ "$CICLOS" -ge "$CANLOOP" ]; then
        imprimir "Cantidad maxima de ciclos alcanzada" I
        exit 0
    fi

    if [[ $DORMIR ]]; then sleep $TESPERA; fi
done
```

4.4. Comando X

Tipo de comando: [Solicitado — Auxiliar]

Justificación: (de su uso si es auxiliar)

Archivos de entrada:

Archivos de salida:

Parámetros: (si tiene)

Opciones: (si tiene)

Ejemplos de invocación:

Código fuente:

5. Archivos

5.1. Archivo de configuración general

Nombre: `$GRUPO/practico.conf`

Estructura: Este archivo consta de líneas que pueden ser de alguno de los siguientes tipos:

- En blanco
- Comentario, comenzando la línea con `#` sin espacios previos
- Definición de una variable en formato bash: `VARIABLE=VALOR`. El valor puede contener espacios internos, si debe tener espacios al comienzo deben escapearse o encomillarse, aunque se recomienda especialmente no utilizar espacios en las rutas.

Este archivo debe tener como mínimo una serie de variables indispensables que el comando instalar generará. Por tal motivo se recomienda que en caso de modificación no se eliminen variables previamente creadas, solo se las modifique y se agreguen variables nuevas.

5.2. Archivo X

Nombre: Su nombre

Estructura: Descripción de la estructura

Justificación: (de su uso si no es requerido por el enunciado)