

# Índice

<b>1. Hipótesis y Aclaraciones</b>	<b>2</b>
<b>2. Problemas relevantes</b>	<b>3</b>
<b>3. Archivo README: Instructivo de instalación</b>	<b>3</b>
<b>4. Comandos Desarrollados</b>	<b>3</b>
4.1. Comando Mover . . . . .	3
4.2. Comando Instalar . . . . .	7
4.3. Comando Iniciar . . . . .	8
4.4. Comando Detectar . . . . .	14
4.5. Comando Interprete . . . . .	19
4.6. Comando Glog . . . . .	20
4.7. Comando Vlog . . . . .	22
4.8. Comando Reporte . . . . .	29
4.9. Comando X . . . . .	44
<b>5. Archivos</b>	<b>45</b>
5.1. Archivo de configuración general . . . . .	45
5.2. Archivo X . . . . .	45

---

## 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.

En el interprete la validación de que no haya archivos repetidos, se realiza verificando que los nombres no sean iguales y los contenidos tampoco. También como el enunciado pide que para los archivos de contratos generados se mantenga un formato numérico de diez caracteres para la parte entera y dos caracteres para la parte decimal, en caso de existir un número que tenga más de esos caracteres (ya que puede suceder, al haber archivos cuyos registros tienen campos numéricos de hasta dieciseis caracteres para la parte entera) para la parte entera se cortarán los primeros diez caracteres de izquierda a derecha y para la decimal se cortarán los primeros dos caracteres de izquierda a derecha del número en cuestión.

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

---

## 2. Problemas relevantes

Quizas no haya sido un problema relevante pero si una cuestión que nos demando un poco de trabajo el tema de manejar números decimales con ',' como separador en vez de '.' ya que tanto bash como perl toman los números de punto flotante con un '.' como separador, con lo cual hubo que hacer manejos para poder mantener una cierta coherencia con lo que pedia el enunciado y el lenguaje.

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 incluido 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

# Necesario para el correcto funcionamiento de glog
export LOGDIR=$movLOGDIR

# 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 Instalar

**Tipo de comando:** Solicitado

**Archivos de entrada:** El archivo de configuración, si es que existe, para una instalación anterior. Se encuentra en:

`$grupo/conf/practico.conf`

Todos los archivos binarios y de datos que seran instalados, se encuentran en

`$grupo/inst/bin/`

y

`$grupo/inst/data/`

**Archivos de salida:** ■ Un archivo de configuración que se generará a partir de la instalación. Se encontrará en

`$grupo/conf/practico.conf`

- Todos los archivos binarios y de datos que serán instalados. Se encontrarán en

---

`$BINDIR`

y

`$DATADIR.`

- Un archivo de log que tenga toda la información de registro de la instalación. Se encuentra en

`$grupo/log/instalar.log`

- Un directorio donde se alojaran los archivos de log de todos los componentes que se ejecuten. Se encontrará en

`$LOGDIR.`

**Hipotesis:** Para la instalación de los componentes necesarios, estos deben encontrarse dentro del directorio:

`$grupo/inst/bin`

Todos los archivos que se encuentren en ese directorio se consideraran como componentes a instalar. El instalador detecta una instalación previa en caso de que encuentre su respectivo archivo de configuración. En caso de encontrar un componente imprime por pantalla que se ha encontrado ese componente y no se instalará.

**Ejemplos de invocación:** `$ instalar`

De esta forma se inicia la instalacion del sistema.

**Código fuente:**

### 4.3. 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
EEXISTENTORNO=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 )

VARNECESARIAS=( \
[0]=CONFFILE
[1]=DATADIR
[2]=MAXLOG
[3]=USIZE )

# 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 \ $FILECONF = $FILECONF
    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

#### NO REMOVER. Linea reservada para asignacion de archivo de configuracion a la variable. ####
CONFFILE=../conf/practico.conf # TODO Borrar!!

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

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

# Se exportan las variables del archivo de configuracion
LINEAS=$(wc -l <$CONFFILE)
(( 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
    continue
fi

# Si se trata de las variables de un comando se las exporta sufijadas
if [ ${LINEA:0:7} = "COMAND=" ]; then
    # Los 3 primeros caracteres del nombre del
    # comando se usan como sufijo de sus variables.
    SUFIJO=${LINEA:7:3}

    # Cada comando tiene 4 lineas aparte de su nombre
    for i in 1 2 3 4
    do
        read LINEA
        # Se ve en el enunciado que los nombres tienen 6 caracteres
        NOMBRE=${SUFIJO}${LINEA:0:6}
        VALOR=${LINEA:7}
        export $NOMBRE=$VALOR
    done

    # Si no, se exporta la variable definida en la linea
    else
        export "$LINEA" #2>/dev/null
        if [ ! $? -eq 0 ]; then
            imprimir "Error de sintaxis en $CONFFILE en la linea $CUENTA"
        fi
    fi
done
} <$CONFFILE

# 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 variable de entorno $VAR, error en $CONFFILE"
    fi
done

# Necesario para el correcto funcionamiento de glog
export LOGDIR=$iniLOGDIR

# Se agregan los directorios bin al PATH para poder ejecutar las demas funciones

```

---

```

PATH="$PATH:$detBINDIR:$gloBINDIR:$iniBINDIR:$insBINDIR"
PATH="$PATH:$intBINDIR:$movBINDIR:$repBINDIR:$vloBINDIR"
export PATH
BINDIR=$iniBINDIR # TODO Para el boton de TODos, quitar!
# 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.
NECESARIOS=( \
[0]="$movBINDIR/mover" \
[1]="$gloBINDIR/glog" \
[2]="$vloBINDIR/vlog" \
[3]="$iniBINDIR/iniciar" \
[4]="$detBINDIR/detectar" \
[5]="$intBINDIR/interprete" \
[6]="$repBINDIR/reporte" \
[7]="$GRUPO/conf/p-s.tab" \
[8]="$GRUPO/conf/T1.tab" \
[9]="$GRUPO/conf/T2.tab" )

unset NOINST
# ${!a[*]} expande a todos los indices del array a.
# No se puede usar for arch in ${a[*]}, que directamente iteraría 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

# Workaroud 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

    # Workaroud 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 "¿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"
    fi
fi

```

---

```
        echo "debe ingresar \"detectar &\" en una consola con el entorno iniciado."
    fi
fi

# Botonea acerca de los TODOs sin hacer.
for arch in $(ls "$BINDIR")
do
    PARA=TO
    HACER=DO
    POR_HACER=$(grep $PARA$HACER -n <"$BINDIR/$arch");
    if [ "$POR_HACER" ]
    then
        echo "El archivo $arch tiene cosas por hacer!"
        echo "$POR_HACER"
    fi
done

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

## 4.4. 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

# Necesario para el correcto funcionamiento de glog
export LOGDIR=$detLOGDIR

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

PSTAB="$GRUPO/conf/p-s.tab"      # Tabla de Paises y Sistemas
DIROK="$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 TODO toque ACA tomas!!!!
    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
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
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 [ $CANTINST -lt 1 ]; then
        interprete &
        intPID=$!
        wait
        COD=$?
        if [ $COD -eq 0 ]; then
            imprimir "PID de interprete: $intPID"
        else
            imprimir "Error de interprete #$COD: ${ERRDESC[$?]}"
        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

```

## 4.5. Comando Interprete

**Tipo de comando:** Solicitado

**Archivos de entrada:**

1. Archivos de Practico Recibidos en \$DATADIR/ok
2. Tabla de Separadores \$DATADIR/conf/T1.tab
3. Tabla de Campos \$DATADIR/conf/T2.tab

**Archivos de salida:**

1. Archivo de Contratos de Préstamos Personales  
\$DATADIR/new/CONTRAT.<pais>

- 
2. Archivos (duplicados) Rechazados `$DATADIR/nok/<nombre del archivo>`
  3. Archivos de Practico Procesados `$DATADIR/old/<pais>-<sistema>-<año>-<mes>`
  4. Log `$DATADIR/log/interprete.log`

**Ejemplos de invocacin:** El comando solo puede invocarse como

```
$ interprete
```

de esta manera, lo que hace es procesar los archivos que se encuentran en el directorio `$DATADIR/ok`, procesarlos y generar los archivos de contrato, con los cuales se llega a un formato estandar, denominados `$DATADIR/new/CONTRAT.<pais>` en el directorio `$DATADIR/new`. Los archivos ya procesados los deja en el directorio `$DATADIR/old`, y los repetidos en `$DATADIR/nok`.

**Código fuente:**

## 4.6. Comando Glog

**Tipo de comando:** Solicitado

**Archivos de salida:** Los archivos correspondientes a los logs registrados. Estos archivos se encontrarán en el directorio:

```
$LOGDIR
```

**Parámetros:** El primer parametro es de caracter obligatorio. Indica el comando que esta generando un log.

El segundo parametro es de caracter opcional. Indica el tipo de mensaje que puede ser informativo (I), un error severo (SE), un warning (W) o un error(E).

El tercer mensaje es de caracter obligatorio. Indica el mensaje que se desea imprimir en el log.

**Consideraciones:** De acuerdo a lo solicitado en el enunciado, existe un tamaño máximo para el tamaño de los archivos de logs. Una vez que se supera este limite, el programa realiza un truncamiento del archivo dejando las ultimas 50 lineas. Pese a que es altamente improbable,

---

podría suceder que con estas 50 líneas se supere el tamaño máximo del archivo. Por lo tanto, una vez que ya se ha realizado el truncamiento se verifica que el tamaño de dichas líneas no supere el tamaño máximo establecido. En caso de superarlo, se dejaría al archivo vacío para que recupere la consistencia.

**Ejemplos de invocación:** Existen dos formas distintas de invocar al comando: pasándole 2 o 3 parametros.

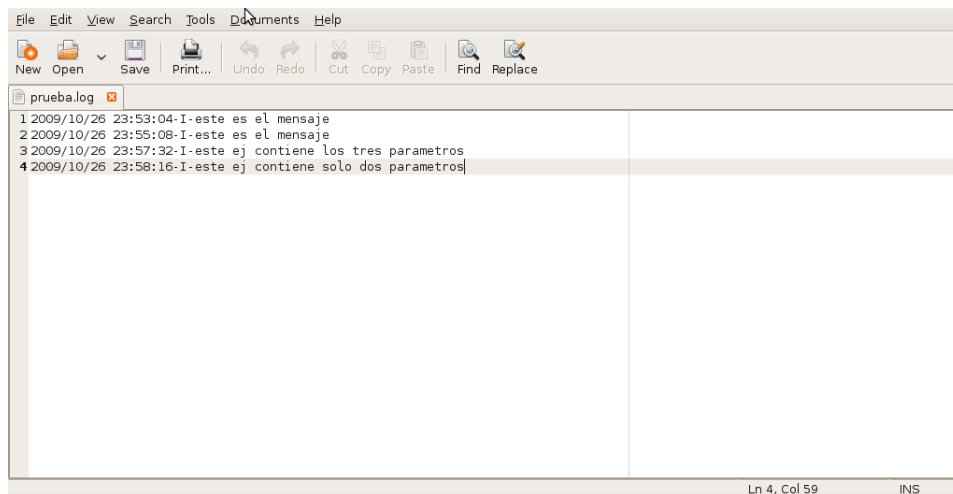
```
glog <Comando> <Tipo de Mensaje> <Mensaje>
glog <Comando> <Mensaje>
```

Se puede observar un ejemplo de la invocación en la captura siguiente.



```
File Edit View Terminal Tabs Help
tomas@tomas-laptop:~/Desktop/7508g1/bin$ glog prueba I "este ej contiene los tr
s parametros"
tomas@tomas-laptop:~/Desktop/7508g1/bin$ glog prueba "este ej contiene solo dos
parametros"
tomas@tomas-laptop:~/Desktop/7508g1/bin$
```

Se genera un archivo llamado *prueba.log* con el siguiente contenido:



**Código fuente:**

## 4.7. Comando Vlog

**Tipo de comando:** Solicitado

**Archivos de entrada:** Archivos de log( ubicados en el LOGDIR).

**Archivos de salida:** standar output.

**Ejemplo de invocación:** El comando vlog puede invocarse como

```
$ vlog
```

Así solo, el comando muestra por salida estandar todos los archivos de logs. Obviamente esto no es lo deseable en la mayoría de los casos por lo que este comando nos ofrece varios filtros para visualizar el archivo de log de cada uno de los comandos. De esta forma, se pueden filtrar los logs por comando, por fecha, hora y tipo de mensaje. Cabe destacar que en la varios filtros poseen la propiedad de hacerlo por rango. Por ejemplo, generalmente es bastante deseable que se filtren los sucesos que sucedieron entre tantos días y a la vez entre un rango de horas determinadas. Todo esto es posible con esta herramienta. A continuación se muestran unos ejemplos con screenshots acerca de como usarla. En

---

la siguiente figura se puede ver el comando -h ( help). En ella se muestran las distintas opciones para poder filtrar los logs, y alguna opción más.

```
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$ vlog -h
Modo de uso: vlog [OPCION]... [ARGUMENTO]
[ARGUMENTO] puede ser una valor o un rango.
Ejemplo vlog -m [10-43] ; vlog -m 14
Por default, sin ningun parametro muestra todos los logs de todos los parametros
los argumentos posibles son:
-H      filtra por hora.
-m      filtra por minutos
-D      filtra por dias
-M      filtra por mes
Los restantes no adminten rango
-A      filtra por anio
-l      filtra por tipo de mensaje de log( SE, E, I, W )
-c      filtra por tipo de comando
-b      no muestra de que archivos son los match
-h      muestra esta ayuda
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$
```

Figura 1: Help del comando vlog

En las posteriores figuras se puede observar el log del comando iniciar, variando la cantidad de parámetros para obtener búsquedas más precisas.

```
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$ vlog -c iniciar
2009/10/20 22:50:40-SE-El archivo indispensable /home/esteban/src/tpsisop/conf/practico.conf no esta presente
2009/10/20 22:50:40-SE-Imposible iniciar el entorno: Instalacion fallida o corrupta
2009/10/25 09:20:07-I-Inicializando entorno
2009/10/25 09:20:14-I-Entorno inicializado correctamente, iniciar terminado
2009/10/26 10:31:11-I-Inicializando entorno
2009/10/26 10:31:15-I-Entorno inicializado correctamente, iniciar terminado
2009/10/26 11:00:10-W-Invocacion con el entorno ya inicializado
2009/10/26 14:41:45-I-Inicializando entorno
2009/10/26 14:41:51-I-Entorno inicializado correctamente, iniciar terminado
2009/10/26 16:02:25-W-Invocacion con el entorno ya inicializado
2009/10/26 16:02:25-I-PID de detectar: 8927
2009/10/26 16:02:55-W-Invocacion con el entorno ya inicializado
2009/10/26 16:02:55-I-PID de detectar: 8927
2009/10/26 16:51:59-W-Invocacion con el entorno ya inicializado
2009/10/26 16:51:59-I-PID de detectar: 8927
2009/10/26 17:03:21-I-Inicializando entorno
2009/10/26 17:03:30-I-Entorno inicializado correctamente, iniciar terminado
2009/10/26 21:48:11-I-Inicializando entorno
2009/10/26 21:48:15-I-Entorno inicializado correctamente, iniciar terminado
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$
```

Figura 2: vlog filtrando solo por comando(en este caso el comando iniciar)

```
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$ vlog -c iniciar -l I -m [40-55]
2009/10/26 14:41:45-I-Inicializando entorno
2009/10/26 14:41:51-I-Entorno inicializado correctamente, iniciar terminado
2009/10/26 16:51:59-I-PID de detectar: 8927
2009/10/26 21:48:11-I-Inicializando entorno
2009/10/26 21:48:15-I-Entorno inicializado correctamente, iniciar terminado
pablin@pablin-laptop:~/Escritorio/sistop/7508g1/bin$
```

Figura 3: vlog con filtros más exhaustivos. Filtros: comando(-c), tipo de mensaje(-l) y por rango de minutos (-m)

### Código fuente:

```
#!/bin/bash

#definicion de constantes
TRUE=1
FALSE=0
ENOENTORNO=14
#fin definicion constantes

# Necesario para el correcto funcionamiento de glog
export LOGDIR=$vloLOGDIR
```



---

```
#####
# DEFINICION FUNCIONES AUXILIARES                                     #
#####

es_rango() {
if [ $(echo $1 | grep '\[.\{1,2\}-.\{1,2\}\]') ]; then
return $TRUE
fi
return $FALSE
}

#devuelve TRUE si el primer parametro esta entre el parametro 2 y el 3.
dentro_rango(){
if [[ $( expr $2 \<= $1 ) == $TRUE ]]; then
if [[ $( expr $1 \<= $3 ) == $TRUE ]];then
return $TRUE
fi
fi
return $FALSE
}

# setea rango1 rango12 rango2 rango22 segun el formato [**-**] donde * es un numero.
# casos a evaluar [12-23] [11-1] [1-23] [1-2].
get_rango() {
rango1=$( expr substr $1 3 1)
if [[ $rango1 != - ]]; then
rango1=$( expr substr $1 2 1)
rango12=$( expr substr $1 3 1)
rango2=$( expr substr $1 6 1)
if [[ $rango2 == ] ]];then
rango2=0 # agrego un cero para tener decena = 0
rango22=$( expr substr $1 5 1)
else
rango2=$( expr substr $1 5 1)
rango22=$( expr substr $1 6 1)
fi
else
rango1=0 # agrego un cero para tener decena = 0
rango12=$( expr substr $1 2 1)
rango2=$( expr substr $1 5 1)
if [[ $rango2 == ] ]];then
rango2=0
rango22=$( expr substr $1 4 1)
else
```

---

```

rango2=$( expr substr $1 4 1)
rango22=$( expr substr $1 5 1)
fi
fi
}

```

```

#crea en la variable rango_valor una expresion regular para el rango dado
#en el primer argumento. Ejemplo [1-30] --> [01-30] ; [12-20] --> [12-20]
#Tambien chequea que los valores esten dentro del rango $2 y $3
#tenemos esto [ rango1 rango12 rango2 rango22 ] Luego para la reg exp tenemos 3 casos
construir_rango() {
get_rango $1
dentro_rango $rango1$rango12 $2 $3
temp1=$?
dentro_rango $rango2$rango22 $2 $3
temp2=$?
if [[ $temp2 != $TRUE ]] || [[ $temp1 != $TRUE ]]; then
echo "rango invalido"
exit 0
fi

temp=$( expr $rango1 = $rango2 )

if [[ $temp == 1 ]];then
rango_valor="($rango1[$rango12-$rango22])"
return 0
fi

temp=$( expr $rango2 - $rango1)
if [[ $temp == 1 ]]; then
rango_valor="($rango1[$rango12-9]|$rango2[0-$rango22])"
else
#caso mas general.
temp1=$( expr $rango1 + 1)
temp2=$( expr $rango2 - 1)
rango_valor="($rango1[$rango12-9]|[$temp1-$temp2][0-9]|$rango2[0-$rango22])"
fi
}

```

```

#construye una expresion regular y aloca el resultado en reg_expr. El argumento
# 2 y 3 son la cota minima y maxima que pueden tomar los valores.
construir_expresion_regular() {
val=$1 #suponemos no es rango, ni varios valores

```

---

```

es_rango $val
if [[ $? == $TRUE ]]; then
construir_rango $val $2 $3
reg_expr=$rango_valor
else
reg_expr=$val
fi
}

# Muestra un help de como usar el comando.
uso() {
echo "Modo de uso: vlog [OPCION]... [ARGUMENTO]  "
echo "[ARGUMENTO] puede ser una valor o un rango. "
echo "Ejemplo vlog -m [10-43] ; vlog -m 14"
echo "Por default, sin ningun parametro muestra todos los logs de todos los parametros"
echo "los argumentos posibles son: "
echo "-H filtra por hora."
echo "-m filtra por minutos"
echo "-D filtra por dias"
echo "-M filtra por mes"
echo "Los restantes no adminten rango"
echo "-A filtra por anio"
echo "-l filtra por tipo de mensaje de log( SE, E, I, W )"
echo "-c filtra por tipo de comando"
echo "-b no muestra de que archivos son los match"
echo "-h muestra esta ayuda"
}

#####
# FIN DEFINICION FUNCIONES AUXILIARES                                     #
#####

#chequeamos que este inicializado el entorno.
if [ ! $ENTORNO_INICIALIZADO ]; then
echo "Glog: Entorno no inicializado"
exit $ENOENTORNO
fi

# Si el directorio de logs no existe sale del programa.
if [ ! -e "$LOGDIR" ]; then
echo "No hay ningun log c"
#mkdir -p $LOGDIR
fi

#####

```

---

```

#Estos son los parametros que podemos usar:
#Para el tipo de log: ES, E, I W
#hora: se puede pasar varias horas separadas o un intervalo de horas.
#ejemplo -H [1,2,12]    o    -H [3-8]
#minutos: es exactamente igual a horas.
#se utiliza con -m
#fecha: aca lo hacemos por anio, mes y dia y con las mismas propiedades que hora
#se utiliza con -D -M y -A respectivamente pero anio no acepta rango.
#comando: tipo de comando a buscar en el glog
#se utiliza con -c
#####

#Por default las variables pueden ser cualquier cosa.
tipolog=.*
anio=...
mes=..
dia=..
hora=..
minuto=..

MOSTRARARCHIVO=

while getopts \l:H:m:D:M:A:c:hb" OPTION
do
case $OPTION in

l)
PARAM_TIPO_LOG=$OPTARG
if [[ $PARAM_TIPO_LOG != [IWE] ]] && [ $PARAM_TIPO_LOG != SE ]; then
echo "tipo de log invalido. Opciones: SE I W E"
exit 0
fi
tipolog=$PARAM_TIPO_LOG

;;

H)
PARAM_HORA=$OPTARG
construir_expresion_regular $PARAM_HORA 0 60
hora=$reg_expr
;;

m)
PARAM_MINUTO=$OPTARG
construir_expresion_regular $PARAM_MINUTO 0 60

```

---

```

minuto=$reg_expr
    ;;

D)
    PARAM_DIA=$OPTARG
    construir_expresion_regular $PARAM_DIA 0 31
    dia=$reg_expr
    ;;

M)
    PARAM_MES=$OPTARG
    construir_expresion_regular $PARAM_MES 0 12
    mes=$reg_expr
    ;;

A)
    PARAM_ANIO=$OPTARG
    anio=$PARAM_ANIO
    ;;

c)
    PARAM_COMANDO=$OPTARG
    ;;

b)
    MOSTRARARCHIVO="-h"
    ;;

?)
    uso
    exit 0
    ;;
esac
done

if [ $PARAM_COMANDO ]; then
    egrep $MOSTRARARCHIVO "$anio/$mes/$dia $hora:$minuto:..-$tipolog" "$LOGDIR/$PARAM_COMANDO.log"
else
    egrep $MOSTRARARCHIVO "$anio/$mes/$dia $hora:$minuto:..-$tipolog" "$LOGDIR/"*
fi

```

## 4.8. Comando Reporte

**Tipo de comando:** Solicitado

- 
- Archivos de entrada:**
1. Archivo de Contratos de Préstamos Personales ubicado en `$grupo/datadir/new/CONTRAT.<pais>`
  2. Archivo Maestro Préstamos Personales Impagos ubicado en `$grupo/datadir/mae/PPI.mae`
  3. Tabla de Paises y Sistemas ubicadas en `$grupo/conf/p-s.tab`

**Archivos de salida:** Los archivos de salida correspondientes a este comando, se generan si el usuario lo especifica.

1. Log `$grupo/logdir/reporte.log`
2. Listados `$grupo/datadir/list/<nombre listado>.<id>`
3. Archivo de Modificaciones de Contratos `$grupo/datadir/new/MODIF.<pais>`

**Ejemplos de invocación:** El comando solo puede invocarse como

`$ reporte`

de esta manera, se carga un menu interactivo en donde el usuario puede:

1. Cargar parámetros de la consulta a realizar
2. Activar o desactivar la grabación de los listados de consulta
3. Activar o desactivar la grabación de modificaciones de contrato
4. Realizar la consulta en base a los parámetros ingresados por teclado

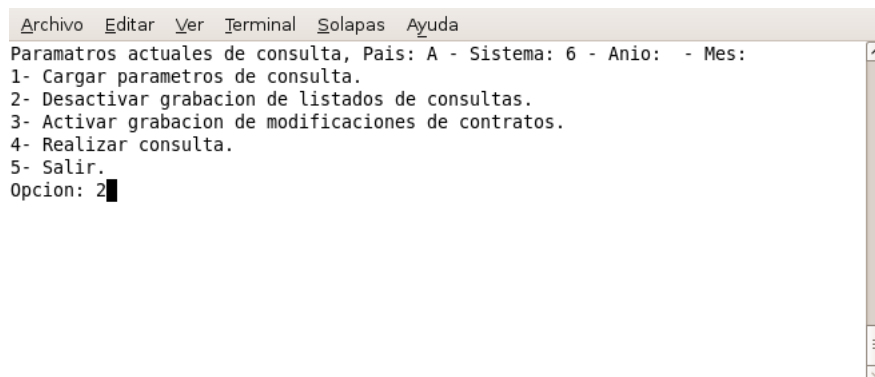
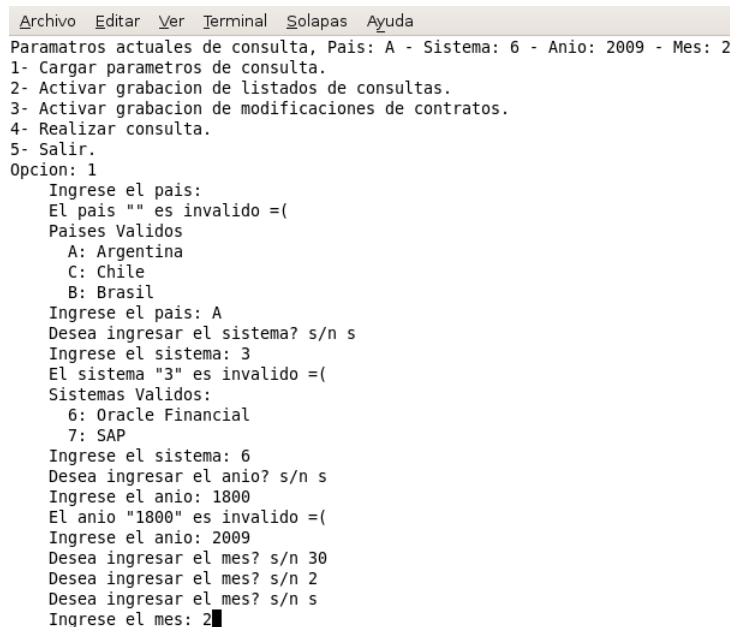


Figura 4: Menú reporte

---

El reporte cuenta con un menú con cinco opciones disponibles, como se puede ver en la imagen anterior. La carga de parámetro se realiza por teclado y el único obligatorio es el nombre del país. La aplicación permite activar y desactivar la grabación de los listados y modificaciones en archivos. Realizar consulta impreme por pantalla los listados correspondientes y una tabla con las modificaciones. En caso de haber activo la grabación en archivos, se realiza la escritura. La última opción, como su nombre lo indica, nos permite abandonar la aplicación.



```
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
Paramatros actuales de consulta, Pais: A - Sistema: 6 - Anio: 2009 - Mes: 2
1- Cargar parametros de consulta.
2- Activar grabacion de listados de consultas.
3- Activar grabacion de modificaciones de contratos.
4- Realizar consulta.
5- Salir.
Opcion: 1
Ingrese el pais:
El pais "" es invalido =(
Países Validos
A: Argentina
C: Chile
B: Brasil
Ingrese el pais: A
Desea ingresar el sistema? s/n s
Ingrese el sistema: 3
El sistema "3" es invalido =(
Sistemas Validos:
6: Oracle Financial
7: SAP
Ingrese el sistema: 6
Desea ingresar el anio? s/n s
Ingrese el anio: 1800
El anio "1800" es invalido =(
Ingrese el anio: 2009
Desea ingresar el mes? s/n 30
Desea ingresar el mes? s/n 2
Desea ingresar el mes? s/n s
Ingrese el mes: 2
```

Figura 5: Carga de parámetros

Esta imagen corresponde a la carga de los parámetros de consulta y a la validación de los mismos. En primer lugar se ingresan el país y el sistema. La validación de estos parámetros se realiza consultando la tabla de países y sistemas ubicada en `$grupo/conf/p-s.tab`. Luego se ingresa el año que, para superar el proceso de validación, debe ser superior a 2000. Por último se ingresa el mes, el cual debe ser un número entre 1 y 12.

Archivo Editar Ver Terminal Solapas Ayuda										
LISTADO										
Contratos comunes sanos con identico Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			4		SANO		SANO		11712,06
Contratos comunes dudosos con identico Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			3		DUDOSO		DUDOSO		12398,19
Contratos comunes sanos con diferente Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			7		SANO		SANO		7959,66
Contratos comunes dudosos con diferente Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			2		DUDOSO		DUDOSO		6518,11
Contratos comunes con diferente estado con identico Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			1		SANO		DUDOSO		10427,86
Contratos comunes con diferente estado con diferente Monto Restante:										
Pais	Sistema	Anio	Mes	Cant	Con	Est	Cont	Est	Mae	Monto
A	6			4		SANO		DUDOSO		1867,37
A	6			7		DUDOSO		SANO		5952,13
MODIFICACIONES										
Pais	Sistema	Anio	Mes	Num	Cont	Estado	Monto			
A	6	2009		2	101A00153000	SANO	3455.9			
A	6	2009		2	101A00153000	SANO	3455.9			
A	6	2008		5	101B00162100	SANO	5727.26			
A	6	2008		12	201A00005900	SANO	6204.39			
A	6	2008		7	201A00188600	SANO	2931.87			
A	6	2009		3	201A00019500	SANO	2244.74			
A	6	2008		10	201A00107800	SANO	1228.28			
A	6	2008		11	201A00018400	SANO	3049.83			
A	6	2009		3	101A00019500	SANO	1954.34			
A	6	2009		6	201A00009400	SANO	1712.87			
A	6	2009		2	101A00153000	SANO	3455.9			
A	6	2009		5	201A00064300	SANO	181.35			
A	6	2008		10	101A00107800	SANO	837.93			
A	6	2008		9	201A00120000	SANO	5307.49			
A	6	2008		11	101A00018400	SANO	2555.71			
A	6	2009		8	201A00254700	SANO	559.65			
Presione Enter para continuar										

Figura 6: Ejemplo consulta

La siguiente imagen corresponde a la consulta país: A sistema: 6. No hay restricción sobre el resto de los parámetros. La salida de los listados en archivos se produce en 6 ficheros distintos, llamados listadosX.jid, en donde “X” e “id” denominan al tipo de consulta y usuario que la realizo, respectivamente.

### Código fuente:

```
#!/usr/bin/perl

use Switch;

# Variables global.
$usId = $ENV{"USER"};
$pais = "A";
$sistema = 6;
$grabarListados = 0;
$grabarModificaciones = 0;
$LISTDIR = $ENV{"DATADIR"}."/list/";
```



---

```

$archivoListadosTotales = $LISTDIR."ListadosTotales.txt";

# Países y sistemas válidos.
%paísesValidos;
%sistemasValidos;

# Valida que el entorno esté inicializado y aborta la ejecución con error en
# caso de ser así.
sub validarEntorno {

    if (!$ENV{"ENTORNO_INICIALIZADO"}) {
        print "Entorno no inicializado\n";
        exit 1;
    }
}

sub glog {
    my ($mensaje, $tipo) = @_;
    #print "Reporte: $mensaje\n";
    'glog reporte "$tipo" "$mensaje"'
}

sub glogAndExit {
    my ($mensaje, $tipo, $exitCode) = @_;
    glog ($mensaje, $tipo);
    exit $exitCode;
}

# Inicializa las variables globales %paísesValidos y %sistemasValidos a partir del
# archivo p-s.tab.
sub inicializarPaísesSistemasValidos {

    my $psTabDir = $ENV{"GRUPO"}."/conf/p-s.tab";
    open (PSTAB, $psTabDir) ||
        glogAndExit ("No se pudo abrir el archivo $psTabDir", "SE", 1);

    my (%países, %sistemas);
    while (my $linea = <PSTAB>) {
        chomp ($linea);
        my @psReg = split("-", $linea);
        $países {$psReg[0]} = $psReg[1];
        $sistemas{$psReg[2]} = $psReg[3];
    }

    close (PSTAB);
}

```

---

```

    %paísesValidos = %países;
    %sistemasValidos = %sistemas;
}

sub mostrarPaísesValidos {
    print "    Países Validos \n";
    foreach my $idPaís (keys(%paísesValidos)) {
        print "        $idPaís: ".$paísesValidos{$idPaís}."\n";
    }
}

sub mostrarSistemasValidos {
    print "    Sistemas Validos:\n";
    foreach my $idSist (keys(%sistemasValidos)) {
        print "        $idSist: ".$sistemasValidos{$idSist}."\n";
    }
}

# No valida nada.
sub validacionNula{

    return $_[0];
}

sub validarPaís {

    my $país = $_[0];
    return exists ($paísesValidos{$país}) ? $país : "";
}

sub validarSistema{

    my $sistema = $_[0];
    return exists ($sistemasValidos{$sistema}) ? $sistema : "";
}

# Se permiten todos los años mayores a 1900.
sub validarAño{

    my $año = int($_[0]);
    return $año >= 2000 ? $año : "";
}

```

---

```

# Valida que un mes dado esté entre 1 y 12 y lo retorna.
sub validarMes{

    my $mes = int(@_[0]);
    return $mes >= 1 && $mes <= 12 ? $mes : "";
}

# Solicita al usuario que cargue un parámetro con un dado nombre y una dada
# función de validación.
sub cargaParametro{

    my ($parametro, $validacion, $mostrarValidos) = @_;
    my ($respuesta, $respuestaValidada);

    while (!$respuestaValidada){

        print "    Ingrese el $parametro: ";
        $respuesta = <STDIN>;
        chomp($respuesta);
        $respuestaValidada = &$validacion($respuesta);
        if (!$respuestaValidada) {
            print "    El $parametro \"$respuesta\" es invalido =(\\n";
            if ($mostrarValidos) {
                &$mostrarValidos();
            }
        }
    }
    return $respuesta;
}

# Permite la carga de parametros opcionales como el sistema, anio y mes.
sub cargaParametroOpcional{

    my ($parametro, $validacion, $mostrarValidos) = @_;
    my $opcion;
    my $respuesta;

    while($opcion ne "s" && $opcion ne "n"){

        print "    Desea ingresar el $parametro? s/n ";
        $opcion = <STDIN>;
        chop($opcion);
    }
}

```

---

```

    if ($opcion eq "n"){
        return $respuesta;
    }

    return &cargaParametro($parametro,$validacion, $mostrarValidos);
}

# Cargar parametros de consulta (Pais, Sistema, Anio y Mes)
sub cargarParametrosDeConsulta{

    my $pais = &cargaParametro("pais","validarPais", "mostrarPaisesValidos");
    my $sistema = &cargaParametroOpcional("sistema","validarSistema",
                                           "mostrarSistemasValidos");

    my $anio;
    my $mes;

    if ($anio = &cargaParametroOpcional("anio", "validarAnio")){
        $mes = &cargaParametroOpcional("mes", "validarMes");
    }

    return $pais, $sistema, $anio, $mes;
}

sub filtroArchivoMaestro{

    #print "filtro maestro @_ \n";
    my ($linea, $filtroPais, $filtroSistema, $filtroAnio, $filtroMes) = @_;
    my @registro = split("-", $linea);
    my ($pais, $sistema, $anio, $mes) = @registro[0..3];

    # Filtros!
    my $filtroOk = (! $filtroPais || $pais eq $filtroPais) &&
                  (! $filtroSistema || $sistema == $filtroSistema) &&
                  (! $filtroAnio || $anio == $filtroAnio) &&
                  (! $filtroMes || $mes == $filtroMes);

    my $nContrato = $registro[7];
    return $filtroOk, $nContrato;
}

sub filtroArchivoContratos{

    #print "filtro contrato @_ \n";
    my ($linea, $filtroSistema, $filtroAnio, $filtroMes) = @_;

```

---

```

my @registro = split("-", $linea);
my ($sistema, $anio, $mes) = @registro[0..2];

# Filtros!
my $filtroOk = (!$filtroSistema || $sistema == $filtroSistema) &&
               (!$filtroAnio    || $anio    == $filtroAnio)    &&
               (!$filtroMes      || $mes      == $filtroMes);

my $nContrato = $registro[3];
return $filtroOk, $nContrato;
}

# Filtra el archivo maestro a partir de los parámetros de consulta (pais,
# sistema, año y mes) y retorna un hash con los contratos filtrados, de la
# forma: clave = número de contrato, valor = <estado>-<monto>
sub filtrarArchivo {

    my ($fileName, $funcionFiltro) = @_ [0,1];
    my (@filtros) = @_ [2..$#_];

    # Hash con registros filtrados del archivo correspondiente.
    my %filtrado;

    open(ARCHIVO, $fileName) ||
        glogAndExit ("No se pudo abrir $fileName", "SE", 1);

    my $linea;
    while ($linea = <ARCHIVO>){

        chomp($linea);
        # Filtros!
        my ($pasaFiltro, $nContrato) = &$funcionFiltro($linea, @filtros);
        if ($pasaFiltro){
            $filtrado{$nContrato} = &convNotacionPunto($linea);
        }
    }

    close(ARCHIVO);

    return %filtrado;
}

sub procesarConsulta{

    my @filtros = @_ [0..3];

```

---

```

my @entradas = @_[4..$#_];
my @entrada;
my ($consulta,$cantidadContratos,$montoMaestro,$montoContrato);

$cantidadContratos = $#entradas + 1;

foreach my $elemento (@entradas) {

    @entrada = split("-", $elemento);
    $montoMaestro += @entrada[3];
    $montoContrato += @entrada[2];
}

if ($cantidadContratos > 0) {
    return join("-", @filtros, $cantidadContratos, @entrada[0,1], $montoContrato,
        $montoMaestro);
}
else {
    return "";
}
}

sub crearConsultaFormateada{
    # print "entro a imprimir consulta\n";
    my $consultaSinFormato=$_[0];
    # print "consulta sin formato : " . $consultaSinFormato;
    chomp($consultaSinFormato);
    my @entrada= split("-", $consultaSinFormato);
    my $linea;

    foreach my $campo (@entrada){
        $linea = $linea . sprintf("%9s %s", $campo, "|");
    }
    $linea = $linea . "\n";
    return $linea;
}

sub crearModificacionFormateada(){

    my $rArrayModificacion = $_[0];

    foreach $modificacionSinFormato (@$rArrayModificacion){

```

---

```

    chomp($modificacionSinFormato);
    my @entrada= split("-", $modificacionSinFormato);
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $pais, "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[0], "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[1], "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[2], "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[3], "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[5], "|";
    $modificacionFormateada= $modificacionFormateada . sprintf '%12s %s', $entrada[11], "|";
    $modificacionFormateada= $modificacionFormateada . "\n";
}

return $modificacionFormateada;
}

sub convNotacionPunto{
    my $nroOriginal= $_[0];
    my @partes=split(",", $nroOriginal);
    my $numero =join(".", @partes);
    return $numero;
}

sub convNotacionComa{
    my $nroOriginal= $_[0];
    my @partes=split(/\./, $nroOriginal);
    my $numero=join(",", @partes);
    return "$numero";
}

sub crearEncabezadoConsulta{
    # $linea = sprintf '%42s %s', "Parametros", "|";
    my $linea = sprintf '%9s %s', "Pais", "|";
    $linea = $linea . sprintf '%9s %s', "Sistema", "|";
    $linea = $linea . sprintf '%9s %s', "Anio", "|";
    $linea = $linea . sprintf '%9s %s', "Mes", "|";
    $linea = $linea . sprintf '%9s %s', "Cant Con", "|";
    $linea = $linea . sprintf '%9s %s', "Est Cont", "|";
    $linea = $linea . sprintf '%9s %s', "Est Mae", "|";
    $linea = $linea . sprintf '%9s %s', "Monto Con", "|";
    $linea = $linea . sprintf '%9s %s', "Monto Mae", "|";
    $linea = $linea . sprintf "\n";
    return $linea;
}

sub crearEncabezadoModificaciones{

```

---

```

    #$linea = sprintf '%42s %s',"Parametros","|";
    my $linea = sprintf '%12s %s',"Pais","|";
    $linea = $linea . sprintf '%12s %s',"Sistema","|";
    $linea = $linea . sprintf '%12s %s',"Anio","|";
    $linea = $linea . sprintf '%12s %s',"Mes","|";
    $linea = $linea . sprintf '%12s %s',"Num Cont","|";
    $linea = $linea . sprintf '%12s %s',"Estado","|";
    $linea = $linea . sprintf '%12s %s',"Monto","|";
    $linea = $linea . sprintf "\n";
    return $linea;
}

sub procesarModificacion{

    my ($rCons,$rPpiFiltrado,$rArrayModificaciones) = @_;
    my $modificacion;

    my ($seg, $min, $hora, $dia, $mes, $anho, @zape) = localtime(time);
    # Los anios comienzan en 1900. Los meses van de 0 a 11.
    my $fecha = $dia."/".($mes+1)."/".($anho+1900);

    my @cons = @$rCons;
    my %ppiFiltrado = %$rPpiFiltrado;
    my (@entradaConsulta,@entradaMaestro);

    foreach my $elemento (@cons){

        @entradaConsulta = split("-", $elemento);
        my $nContrato = @entradaConsulta[4];

        @entradaMaestro = split("-", $ppiFiltrado{$nContrato});

        my $consultaActual = join("-", @entradaMaestro[1,2,3], $nContrato, @entradaMaestro[8,6,10..14],
                                     @entradaConsulta[3], $fecha, $usId)."\n";
        push(@$rArrayModificaciones, $consultaActual);
        $modificacion = $modificacion.$consultaActual;
    }

    return $modificacion;
}

sub realizarConsulta{

    my @filtrosPPI = @_;
    my @filtrosContrato = @_[1..$#_];

```



---

```

my $pais = $_[0];

my $ppiDir      = $ENV{"DATADIR"}."/mae/PPI.mae";
my $contratosDir = $ENV{"DATADIR"}."/new/CONTRAT.$pais";

my %ppiFiltrado =
    &filtrarArchivo($ppiDir,      "filtroArchivoMaestro",  @filtrosPPI);
my %contratosFiltrado =
    &filtrarArchivo($contratosDir, "filtroArchivoContratos", @filtrosContrato);

my (@consA, @consB, @consC, @consD, @consE1, @consE2, @consF1, @consF2);

foreach my $nContrato (keys(%contratosFiltrado)) {

    if (exists $ppiFiltrado{$nContrato}){

        my $lineaContrato = $contratosFiltrado{$nContrato};
        my $lineaMaestro = $ppiFiltrado{$nContrato};

        my @arrayMaestro = split("-", $lineaMaestro);
        my @arrayContrato = split("-", $lineaContrato);

        # Calcula el monto restante.
        my ($MT_CRD, $MT_IMPAGO, $MT_INDE, $MT_OTRUMDC) =
            @arrayMaestro[10,11,13,14];

        my $montoMaestro = $MT_CRD + $MT_IMPAGO + $MT_INDE - $MT_OTRUMDC;
        my $estadoMaestro = @arrayMaestro[6];

        my ($estadoContrato, $montoContrato) = @arrayContrato[5,11];

        $lineaConsulta = join("-", $estadoContrato, $estadoMaestro,
                                   $montoContrato, $montoMaestro, $nContrato);
        $igualMonto = ($montoMaestro == $montoContrato);

        my $estados = "$estadoMaestro $estadoContrato";
        switch ($estados) {
            case "SANO SANO"      { $igualMonto ? push(@consA, $lineaConsulta)
                                             : push(@consC, $lineaConsulta); }
            case "DUDOSO DUDOSO" { $igualMonto ? push(@consB, $lineaConsulta)
                                             : push(@consD, $lineaConsulta); }
            case "SANO DUDOSO"   { $igualMonto ? push(@consE2, $lineaConsulta)
                                             : push(@consF2, $lineaConsulta); }
            case "DUDOSO SANO"   { $igualMonto ? push(@consE1, $lineaConsulta)
                                             : push(@consF1, $lineaConsulta); }
        }
    }
}

```

---

```

        else { &glog("Estado inválido: $estados", "E"); }
    }
}
else{
    &glogAndExit ("No existe el numero de contrato $nContrato en el archivo maestro",
        "E");
}
}

# Listados.
print "\nLISTADO\n";
my @consultasFormateadas;
my $consulta;
my $consultaAux;
my $encabezadoConsulta=&crearEncabezadoConsulta();

$consultaSinFormato = &procesarConsulta(@filtrosPPI,@consA)."\n";
$descConsulta = "Contratos comunes sanos con identico Monto Restante: \n";
$consulta = &crearConsultaFormateada($consultaSinFormato);
push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta);

$consultaSinFormato= &procesarConsulta(@filtrosPPI,@consB)."\n";
$descConsulta = "Contratos comunes dudosos con identico Monto Restante: \n";
$consulta = &crearConsultaFormateada($consultaSinFormato);
push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta);

$descConsulta = "Contratos comunes sanos con diferente Monto Restante: \n";
$consultaSinFormato=&procesarConsulta(@filtrosPPI,@consC)."\n";
$consulta = &crearConsultaFormateada($consultaSinFormato);
push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta);

$descConsulta = "Contratos comunes dudosos con diferente Monto Restante: \n";
$consultaSinFormato=&procesarConsulta(@filtrosPPI,@consD)."\n";
$consulta = &crearConsultaFormateada($consultaSinFormato);
push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta);

$descConsulta = "Contratos comunes con diferente estado con identico Monto Restante: \n";
$consultaSinFormato= &procesarConsulta(@filtrosPPI,@consE1)."\n";
$consulta = &crearConsultaFormateada($consultaSinFormato);
$consultaSinFormato= &procesarConsulta(@filtrosPPI,@consE2)."\n";
$consultaAux = &crearConsultaFormateada($consultaSinFormato);
push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta.$consultaAux);

$descConsulta = "Contratos comunes con diferente estado con diferente Monto Restante: \n";
$consultaSinFormato=&procesarConsulta(@filtrosPPI,@consF1)."\n";

```

---

```

    $consulta = &crearConsultaFormateada($consultaSinFormato);
$consultaSinFormato=&procesarConsulta(@filtrosPPI,@consF2)."\n";
    $consultaAux = &crearConsultaFormateada($consultaSinFormato);
    push(@consultasFormateadas,$descConsulta.$encabezadoConsulta.$consulta.$consultaAux);

    if ($grabarListados){
        open(LISTADOS,">>$archivoListadosTotales");
        foreach my $string (@consultasFormateadas){
            print LISTADOS &convNotacionComa($string);
        }
        close(LISTADOS);

        my $indice = 0;
        foreach my $idList ("A","B","C","D","E","F"){
            open(arch , ">>".$LISTDIR."list".$idList."".$$usId);
            print arch $consultasFormateadas[$indice];
            close(arch);
            $indice++;
        }
    }

    foreach my $string (@consultasFormateadas){
        print &convNotacionComa($string);
    }

    # Modificaciones.
    print "\nMODIFICACIONES\n";
    @arrayModificaciones;
    my $modificaciones = &procesarModificacion(\@consC,\%ppiFiltrado,\@arrayModificaciones).
        &procesarModificacion(\@consE2,\%ppiFiltrado,\@arrayModificaciones).
        &procesarModificacion(\@consF2,\%ppiFiltrado,\@arrayModificaciones);

    my $modificacionesFormateadas = &crearModificacionFormateada(\@arrayModificaciones);

    print &crearEncabezadoModificaciones;
    print $modificacionesFormateadas;

    if ($grabarModificaciones){
        $archivoModificaciones = $ENV{"DATADIR"}."new/MODIF.$pais";
        open(MODIFICACIONES,">>$archivoModificaciones");
        print MODIFICACIONES convNotacionComa($modificaciones);
        close(MODIFICACIONES);
    }

    print "Presione Enter para continuar";

```

---

```

    my $enter = <STDIN>;
}

sub menu{

    my $salir = 0;

    while (!$salir){

        system("clear");
        print "Parametros actuales de consulta, \Pais: $pais - ".
            "Sistema: $sistema - Anio: $anio - Mes: $mes\n";
        print "1- Cargar parametros de consulta.\n";
        print "2- ".$grabarListados ? "Desactivar" : "Activar").
            " grabacion de listados de consultas.\n";
        print "3- ".$grabarModificaciones ? "Desactivar" : "Activar").
            " grabacion de modificaciones de contratos.\n";
        print "4- Realizar consulta.\n";
        print "5- Salir.\n";
        print "Opcion: ";
        $opcion = <STDIN>;
        chomp($opcion);

        switch ($opcion) {

            case 1 { ($pais,$sistema,$anio,$mes) = &cargarParametrosDeConsulta(); }
            case 2 { $grabarListados = !$grabarListados }
            case 3 { $grabarModificaciones = !$grabarModificaciones }
            case 4 { &realizarConsulta($pais,$sistema,$anio,$mes) }
            case 5 { $salir=1 }
        }
    }
    if (rand() < 0.001) { print "Sos el elegido!\n"; }
}

# Bloque principal.
&validarEntorno();
&inicializarPaisesSistemasValidos();
&menu();

```

## 4.9. 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 escaparse 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)