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.

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 màs 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 ",çomo separador en vez de ".za que tanto bash como perl toman los números de punto flotante con un ".çomo 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 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 índices 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

    # 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 "¿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 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:

```
#! /bin/bash

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

#HIPOTESIS
# Cuando valida que no haya archivos repetidos valida que los nombres no sean iguales y los cont
# Cuando pasa el formato numerico para la parte entera corta los primeros 10 caracteres de izqui
DATADIR=../data # TODO Provisorio para pruebas: ¿DATADIR debe ser parte del entorno?
T1="$GRUPO/conf/T1.tab" # Tabla de Separadores
T2="$GRUPO/conf/T2.tab" #Tabla de Campos
DIROK="$DATADIR/ok"
DIRNOK="$DATADIR/nok"
CONTRATOS="$DATADIR/new/CONTRAT"
DIROLD="$DATADIR/old"
INTERPRETE=$(basename $0)

# Funcion para imprimir por consola y al log
# Uso: imprimir mensaje [tipo]
imprimir() {
echo $1
glog interprete $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" = $INTERPRETE ]; then (( CANTINST -= 1 )); fi
}

# Funcion para validar que no haya archivos que se procesen dos veces, para eso mira dentro de D
# Uso: validar_repetido archivo
validar_repetido(){
if [ -z $1 ]; then
```

```
return
fi

LISTAOLD=$(ls -1 $DIROLD)
ARCHIVOSOLD=( $LISTAOLD ) # Array que contiene cada linea de la lista como un elemento
for ARCHIVOOLD in ${ARCHIVOSOLD[*]}
do
# No se procesan directorios
if [ -d "$DIROLD/$ARCHIVOOLD" ]; then continue; fi

# Si son el mismo archivo entonces se devuelve true
if [ $(cmp "$DIROLD/$ARCHIVOOLD" "$1"| wc -l) = 0 ]; then
true
return
fi

done
false

}
# A partir del nombre del archivo obtiene el pais, el sistema, el anio y el mes y con eso puede
# Uso: obtener_pais_sistema_anio_mes nombrearchivo
# Almacena el pais en PAIS, el sistema en SISTEMA el anio en ANIO y el mes en MES, el separador
obtener_pais_sistema_anio_mes(){
PAIS=$(expr "$1" : '\([^-]*\)')
SISTEMA=$(expr "$1" : '[^-]*-\([^-]*\)')
ANIO=$(expr "$1" : '[^-]*-[^-]*-\([^-]*\)')
MES=$(expr "$1" : '.*-\(.*\)')
}

# Obtiene los separadores para luego poder trabajar.
# Uso: determinar_separadores
# Almacena el separador de campo en SEPCAMPO y el separador decimal en SEPDECIMAL
determinar_separadores(){
SEPCAMPO=$(grep "$PAIS-$SISTEMA" $T1| cut -d '-' -f 3)
SEPDECIMAL=$(grep "$PAIS-$SISTEMA" $T1| cut -d '-' -f 4)
}

# Obtiene los formatos y las posiciones de los campos para luego poder trabajar.
# Uso: determinar_campos
# Almacena cada posicion en NOMBRECAMPOP y cada formato en NOMBRECAMPOF
determinar_campos(){

# Fecha del contrato.
DT_FLUXP=$( grep "$PAIS-$SISTEMA-DT_FLUX-.*-1$" $T2| cut -d '-' -f 4 )
```

```
DT_FLUXF=$( grep "$PAIS-$SISTEMA-DT_FLUX-.*-1$" $T2| cut -d '-' -f 5 )

# Estado contable.
CD_STATCTBP=$( grep "$PAIS-$SISTEMA-CD_STATCTB-.*-1$" $T2| cut -d '-' -f 4 )
CD_STATCTBF=$( grep "$PAIS-$SISTEMA-CD_STATCTB-.*-1$" $T2| cut -d '-' -f 5 )

# Numero de contrato.
NO_CONTRATP=$( grep "$PAIS-$SISTEMA-NO_CONTRAT-.*-1$" $T2| cut -d '-' -f 4 )
NO_CONTRATF=$( grep "$PAIS-$SISTEMA-NO_CONTRAT-.*-1$" $T2| cut -d '-' -f 5 )

# Monto impago.
MT_IMPAGOP=$( grep "$PAIS-$SISTEMA-MT_IMPAGO-.*-1$" $T2| cut -d '-' -f 4 )
MT_IMPAGOF=$( grep "$PAIS-$SISTEMA-MT_IMPAGO-.*-1$" $T2| cut -d '-' -f 5 )

# Monto del credito.
MT_CRDP=$( grep "$PAIS-$SISTEMA-MT_CRD-.*-1$" $T2| cut -d '-' -f 4 )
MT_CRDF=$( grep "$PAIS-$SISTEMA-MT_CRD-.*-1$" $T2| cut -d '-' -f 5 )

# Otras sumas del cliente.
MT_OTRSUMDCP=$( grep "$PAIS-$SISTEMA-MT_OTRSUMDC-.*-1$" $T2| cut -d '-' -f 4 )
MT_OTRSUMDCF=$( grep "$PAIS-$SISTEMA-MT_OTRSUMDC-.*-1$" $T2| cut -d '-' -f 5 )

# Monto de interes devengado.
MT_INDEP=$( grep "$PAIS-$SISTEMA-MT_INDE-.*-1$" $T2| cut -d '-' -f 4 )
MT_INDEF=$( grep "$PAIS-$SISTEMA-MT_INDE-.*-1$" $T2| cut -d '-' -f 5 )

# Monto de interes no devengado.
MT_INNODEP=$( grep "$PAIS-$SISTEMA-MT_INNODE-.*-1$" $T2| cut -d '-' -f 4 )
MT_INNODEF=$( grep "$PAIS-$SISTEMA-MT_INNODE-.*-1$" $T2| cut -d '-' -f 5 )

}


# Lee las fechas y las transforma en el formato requerido
# Uso: transformar_fecha linea
# Deja la fecha del contrato en DT_FLUX
transformar_fecha(){

# Lee la fecha de la linea extraida
DT_FLUXL=$( echo $1| cut -d $SEPCAMPO -f $DT_FLUXP)

# Separo segun la convencion dada
# Los primeros 6 caracteres son el formato en si
FORMATO_DT_FLUX=$( echo $DT_FLUXF| cut --characters=1-6)
# Los ultimos caracteres son la longitud
```

```
LONGITUD_DT_FLUX=$( echo $DT_FLUXF| cut --characters=7-$((${#DT_FLUXF}-1)) )

# Valida que no se exceda ni se tenga menor longitud que la pedida, si es asi termina la ejecuci
if [ $LONGITUD_DT_FLUX -ne ${#DT_FLUXL} ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Transforma la fecha al formato correspondiente
if [ "$FORMATO_DT_FLUX" = "ddmmyy" ] ; then
if [ $LONGITUD_DT_FLUX = 8 ]; then
DT_FLUX="$( echo $DT_FLUXL| cut --characters=1-2)/$( echo $DT_FLUXL| cut --characters=3-4)/$( ec
else
DT_FLUX="$( echo $DT_FLUXL| cut --characters=1-2)/$( echo $DT_FLUXL| cut --characters=4-5)/$( ec
fi
else
if [ $LONGITUD_DT_FLUX = 8 ]; then
DT_FLUX="$( echo $DT_FLUXL| cut --characters=7-8)/$( echo $DT_FLUXL| cut --characters=5-6)/$( ec
else
DT_FLUX="$( echo $DT_FLUXL| cut --characters=9-10)/$( echo $DT_FLUXL| cut --characters=6-7)/$( e
fi
fi
}

# Lee los alfanumericos y los transforma en el formato requerido
# Uso: transformar_alfanumericos linea
# Deja el estado contable en CD_STATCTB y el numero de contrato en NO_CONTRAT
transformar_alfanumericos(){

# Lee el estado contable de la linea extraida
CD_STATCTBL=$( echo $1| cut -d $SEPCAMPO -f $CD_STATCTBP)

# Lee el numero de contrato de la linea extraida
NO_CONTRATL=$( echo $1| cut -d $SEPCAMPO -f $NO_CONTRATP)

# El caracter entre el $ y el . es la longitud maxima, obtiene la longitud maxima de ambos campo
LONGITUD_CD_STATCTB=$( echo $CD_STATCTBF| cut -d '$' -f 2| cut -d '.' -f 1)
LONGITUD_NO_CONTRAT=$( echo $NO_CONTRATF| cut -d '$' -f 2| cut -d '.' -f 1)

# Valida que no se superen las longitudes maximas, si es asi termina la ejecucion con un error d
if [ ${#CD_STATCTBL} -gt $LONGITUD_CD_STATCTB ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi
if [ ${#NO_CONTRATL} -gt $LONGITUD_NO_CONTRAT ]; then
```

```
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Transforma el estado contable al formato correspondiente
if [ "$CD_STATCTBL" = "SNA" ] || [ "$CD_STATCTBL" = "SNIM" ]; then
CD_STATCTB="SANOS"
fi
if [ "$CD_STATCTBL" = "DTCA" ] || [ "$CD_STATCTBL" = "DTXA" ]; then
CD_STATCTB="DUDOSOS"
fi
if [ "$CD_STATCTBL" = "CTX" ]; then
CD_STATCTB="CTX"
fi

# Transforma el numero de contrato al formato correspondiente
NO_CONTRAT=$( echo $NO_CONTRATL )


}

# Lee los numericos y los transforma en el formato requerido
# Uso: transformar_numericos linea
# Deja el monto de credito en MT_CRD, el monto impago en MT_IMPAGO, el monto de otras sumas del
transformar_numericos(){

# Lee el monto de credito de la linea extraida
MT_CRDL=$( echo $1| cut -d $SEPCAMPO -f $MT_CRDP)

# Lee el monto impago de la linea extraida
MT_IMPAGOL=$( echo $1| cut -d $SEPCAMPO -f $MT_IMPAGOP)

# Lee el monto de otras sumas del cliente de la linea extraida
MT_OTRSUMDCL=$( echo $1| cut -d $SEPCAMPO -f $MT_OTRSUMDCP)

# Lee el monto de interes devengado de la linea extraida
MT_INDEL=$( echo $1| cut -d $SEPCAMPO -f $MT_INDEP)

# Lee el monto de interes no devengado de la linea extraida
MT_INNODEL=$( echo $1| cut -d $SEPCAMPO -f $MT_INNODEP)

# Longitudes maximas monto de credito
LONGMAXE_MT_CRD=$( echo $MT_CRDF| cut -d '.' -f 1| cut --characters=7- )
LONGMAXD_MT_CRD=$( echo $MT_CRDF| cut -d '.' -f 2 )
```

```
# Longitudes maximas monto impago
LONGMAXE_MT_IMPAGO=$( echo $MT_IMPAGOF| cut -d '.' -f 1| cut --characters=7- )
LONGMAXD_MT_IMPAGO=$( echo $MT_IMPAGOF| cut -d '.' -f 2 )

# Longitudes maximas monto otras sumas
LONGMAXE_MT_OTRSUMDC=$( echo $MT_OTRSUMDCF| cut -d '.' -f 1| cut --characters=7- )
LONGMAXD_MT_OTRSUMDC=$( echo $MT_OTRSUMDCF| cut -d '.' -f 2 )

# Longitudes maximas monto devengado
LONGMAXE_MT_INDE=$( echo $MT_INDEF| cut -d '.' -f 1| cut --characters=7- )
LONGMAXD_MT_INDE=$( echo $MT_INDEF| cut -d '.' -f 2 )

# Longitudes maximas monto no devengado
LONGMAXE_MT_INNODE=$( echo $MT_INNODEF| cut -d '.' -f 1| cut --characters=7- )
LONGMAXD_MT_INNODE=$( echo $MT_INNODEF| cut -d '.' -f 2 )

# Obtiene la parte entera y decimal del monto de credito y valida que no se superen las longitud
E_MT_CRD=$( echo $MT_CRDL| cut -d $SEPDECIMAL -f 1 )
D_MT_CRD=$( echo $MT_CRDL| cut -d $SEPDECIMAL -f 2 )
if [ ${#E_MT_CRD} -gt $LONGMAXE_MT_CRD ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi
if [ ${#D_MT_CRD} -gt $LONGMAXD_MT_CRD ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Obtiene la parte entera y decimal del monto impago y valida que no se superen las longitudes m
E_MT_IMPAGO=$( echo $MT_IMPAGOL| cut -d $SEPDECIMAL -f 1 )
D_MT_IMPAGO=$( echo $MT_IMPAGOL| cut -d $SEPDECIMAL -f 2 )
if [ ${#E_MT_IMPAGO} -gt  $LONGMAXE_MT_IMPAGO ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi
if [ ${#D_MT_IMPAGO} -gt $LONGMAXD_MT_IMPAGO ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Obtiene la parte entera y decimal del monto de otras sumas y valida que no se superen las long
E_MT_OTRSUMDC=$( echo $MT_OTRSUMDCL| cut -d $SEPDECIMAL -f 1 )
D_MT_OTRSUMDC=$( echo $MT_OTRSUMDCL| cut -d $SEPDECIMAL -f 2 )
if [ ${#E_MT_OTRSUMDC} -gt $LONGMAXE_MT_OTRSUMDC ]; then
echo "Interprete: Error de formato."
```

```
exit $ENOFORMATO
fi
if [ ${#D_MT_OTRSUMDC} -gt $LONGMAXD_MT_OTRSUMDC ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Obtiene la parte entera y decimal del monto devengado y valida que no se superen las longitude
E_MT_INDE=$( echo $MT_INDEL| cut -d $SEPDECIMAL -f 1 )
D_MT_INDE=$( echo $MT_INDEL| cut -d $SEPDECIMAL -f 2 )
if [ ${#E_MT_INDE} -gt $LONGMAXE_MT_INDE ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi
if [ ${#D_MT_INDE} -gt $LONGMAXD_MT_INDE ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Obtiene la parte entera y decimal del monto no devengado y valida que no se superen las longit
E_MT_INNODE=$( echo $MT_INNODEL| cut -d $SEPDECIMAL -f 1 )
D_MT_INNODE=$( echo $MT_INNODEL| cut -d $SEPDECIMAL -f 2 )
if [ ${#E_MT_INNODE} -gt $LONGMAXE_MT_INNODE ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi
if [ ${#D_MT_INNODE} -gt $LONGMAXD_MT_INNODE ]; then
echo "Interprete: Error de formato."
exit $ENOFORMATO
fi

# Si el entero o el decimal estan vacios se los pone en cero
if [ "$E_MT_CRD" = "" ]; then
E_MT_CRD=0
fi
if [ "$D_MT_CRD" = "" ]; then
D_MT_CRD=0
fi

if [ "$E_MT_IMPAGO" = "" ]; then
E_MT_IMPAGO=0
fi
if [ "$D_MT_IMPAGO" = "" ]; then
D_MT_IMPAGO=0
fi
```

```
if [ "$E_MT_OTRSUMDC" = "" ]; then
E_MT_OTRSUMDC=0
fi
if [ "$D_MT_OTRSUMDC" = "" ]; then
D_MT_OTRSUMDC=0
fi

if [ "$E_MT_INDE" = "" ]; then
E_MT_INDE=0
fi
if [ "$D_MT_INDE" = "" ]; then
D_MT_INDE=0
fi

if [ "$E_MT_INNODE" = "" ]; then
E_MT_INNODE=0
fi
if [ "$D_MT_INNODE" = "" ]; then
D_MT_INNODE=0
fi

# Forma los numeros en el formato para sumar
MT_CRD=$E_MT_CRD.$D_MT_CRD
MT_IMPAGO=$E_MT_IMPAGO.$D_MT_IMPAGO
MT_OTRSUMDC=$E_MT_OTRSUMDC.$D_MT_OTRSUMDC
MT_INDE=$E_MT_INDE.$D_MT_INDE
MT_INNODE=$E_MT_INNODE.$D_MT_INNODE

# Obtiene el monto restante
MT_RESTANTE=$(echo "$MT_CRD + $MT_IMPAGO + $MT_INDE -$MT_OTRSUMDC" | bc -l)
# Lo separa en parte decimal y entera
E_MT_RESTANTE=$( echo $MT_RESTANTE| cut -d '.' -f 1 )
D_MT_RESTANTE=$( echo $MT_RESTANTE| cut -d '.' -f 2 )
# Si el entero o el decimal estan vacios se los pone en cero
if [ "$E_MT_RESTANTE" = "" ]; then
E_MT_RESTANTE=0
fi
if [ "$D_MT_RESTANTE" = "" ]; then
D_MT_RESTANTE=0
fi

# Transforma cada monto al formato correspondiente (corta los primeros 10 caracteres de la parte
MT_CRD=$( echo $E_MT_CRD| cut --characters=1-10 ),$( echo $D_MT_CRD| cut --characters=1-2 )
MT_IMPAGO=$( echo $E_MT_IMPAGO| cut --characters=1-10 ),$( echo $D_MT_IMPAGO| cut --characters=1
```

```
MT_OTRSUMDC=$( echo $E_MT_OTRSUMDC| cut --characters=1-10 ),$( echo $D_MT_OTRSUMDC| cut --charac
MT_INDE=$( echo $E_MT_INDE| cut --characters=1-10 ),$( echo $D_MT_INDE| cut --characters=1-2 )
MT_INNODE=$( echo $E_MT_INNODE| cut --characters=1-10 ),$( echo $D_MT_INNODE| cut --characters=1
MT_RESTANTE=$( echo $E_MT_RESTANTE| cut --characters=1-10 ),$( echo $D_MT_RESTANTE| cut --charac
}

# Va recorriendo linea por linea del archivo pasado como parametro y genera un nuevo registro en
# Uso: procesar archivo
procesar(){

REGISTROS=$(wc -l <$1)
CUENTA=0
GRABADOS=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

transformar_fecha $LINEA
transformar_alfanumericos $LINEA
# Si el contrato tiene estado contable CTX se descarta, en cualquier otro caso se almacena
if [ "$CD_STATCTB" != "CTX" ]; then
transformar_numericos $LINEA
# Si el monto restante es mayor a 0 se guarda sino se descarta, no funciona bien
if [ $( echo $MT_RESTANTE| cut --characters=1) != '-' ]; then
echo "$SISTEMA-$ANIO-$MES-$NO_CONTRAT-$DT_FLUX-$CD_STATCTB-$MT_CRD-$MT_IMPAGO-$MT_INDE-$MT_INNOD
(( GRABADOS += 1 ))
fi
fi
(( CUENTA += 1 ))

done
} < $1
}

# Solo se permite una instancia activa de este script.
contar_instancias $INTERPRETE
if [ $CANTINST -gt 1 ]; then
```

```
imprimir "Interprete ya esta corriendo" W
exit 0
fi

imprimir "Interprete iniciado" I

# Valida que el ambiente este inicializado.
if [ ! $ENTORNO_INICIALIZADO ]; then
echo "Interprete: Entorno no inicializado"
exit $ENOENTORNO
fi

#Cuenta la cantidad de archivos
CANTARCH=$(ls $DIROK| wc -l)
imprimir "Inicio de Interprete, Cantidad de Archivos: $CANTARCH" I


LISTAOK=$(ls -1 $DIROK)
ARCHIVOSOK=( $LISTAOK ) # Array que contiene cada linea de la lista como un elemento
# Para cada elemento del directorio se realiza el procesamiento necesario
for ARCHIVOOK in ${ARCHIVOSOK[*]}
do
# No se procesan directorios
if [ -d "$DIROK/$ARCHIVOOK" ]; then continue; fi

# Si el archivo ya se proceso entonces se pasa al directorio nok, sino se trabaja con el mismo
if validar_repetido $DIROK/$ARCHIVOOK; then
mover "$DIROK/$ARCHIVOOK" $DIRNOK interprete
imprimir "Duplicado: $ARCHIVOOK" I
else
obtener_pais_sistema_anio_mes $ARCHIVOOK
determinar_separadores
determinar_campos
procesar $DIROK/$ARCHIVOOK
mover "$DIROK/$ARCHIVOOK" $DIROLD interprete
imprimir "Registros de Input: $CUENTA. Registros  de Output: $GRABADOS" I
fi
done

imprimir "Fin de interprete" I
```

4.5. 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:

4.6. 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.m`
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

Código fuente:

```
#! /usr/bin/perl

use Switch;

# Variables global.
$usId = "yo";
$pais = "A";
$sistema = 6;
$grabarListados = 0;
$grabarModificaciones = 0;
$archivoListados = "Listados.txt";
$archivoModificaciones = "Modificaciones.txt";

# Paises y sistemas válidos.
%paisesValidos;
%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;
  }
}

# Inicializa las varibles gobales %paisesValidos y %sistemasValidos a partir del
# archivo p-s.tab.
sub inicializarPaisesSistemasValidos {

  # TODO obtener el nombre de archivo a partir de la varible de entorno DATADIR.
  my $filename = "../conf/p-s.tab";
  open (PSTAB, $filename) || die "No se pudo abrir el archivo $filename";

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

```
    $sistemas{$psReg[2]} = $psReg[3];
  }

  close (PSTAB);

  %paisesValidos = %paises;
  %sistemasValidos = %sistemas;
}

sub mostrarPaisesValidos {
  print "    Paises Validos \n";
  foreach my $idPais (keys(%paisesValidos)) {
    print "      $idPais: ".$paisesValidos{$idPais}."\n";
  }
}

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


# No valida nada.
sub validacionNula{

  return @_[0];
}

# TODO
sub validarPais {

  my $pais = @_[0];
  return exists ($paisesValidos{$pais}) ? $pais : "";
}

# TODO
sub validarSistema{

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

# Se permiten todos los anios mayores a 1900.
```

```
sub validarAnio{

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

# 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: ";
    $respuesta = <STDIN>;
    chop($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;

  # TODO Usar glog?
  open(ARCHIVO,$fileName) || die ("No se pudo abrir $filename\n");

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

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

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

  my ($seg, $min, $hora, $dia, $mes, $anho, @zape) = localtime(time);
  my $fecha = $dia."/".$mes."/".$anho;

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

  foreach my $elemento (@cons){

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

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

    $modificacion = $modificacion.join("-", @entradaMaestro[1,2,3,8,6,10,11,12,13,14], @entradaC
  }

  return $modificacion;
}

sub realizarConsulta{

  my @filtrosPPI = @_;
  my @filtrosContrato = @_[1..$#_];
  my $pais = $_[0];
  my $key;

  # TODO Obtener las rutas de los archivos a partir de la variable de entorno
  # DATADIR.
  my %ppiFiltrado = &filtrarArchivo("PPI.mae","filtroArchivoMaestro",@filtrosPPI);
  my %contratosFiltrado = &filtrarArchivo("CONTRAT.$pais","filtroArchivoContratos",@filtrosContr
  my (@consA, @consB, @consC, @consD, @consE1, @consE2, @consF1, @consF2);

  foreach $key (keys(%contratosFiltrado)) {

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

    if ($lineaMaestro){

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

      # Calcula el monto restante.
      my ($MT_CRD, $MT_IMPAGO, $MT_INDE, $MT_OTRSUMDC) = @arrayMaestro[10,11,13,14];
      my $montoMaestro = $MT_CRD + $MT_IMPAGO + $MT_INDE - $MT_OTRSUMDC;
      my $estadoMaestro = @arrayMaestro[6];

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

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

      if (($estadoMaestro eq "SANO") && ($estadoContrato eq "SANO")){
        $igualMonto ? push(@consA,$lineaConsulta) : push(@consC,$lineaConsulta);
```

```
      }
      elsif (($estadoMaestro eq "DUDOSO") && ($estadoContrato eq "DUDOSO")){
      $igualMonto ? push(@consB,$lineaConsulta) : push(@consD,$lineaConsulta);
      }
      elsif (($estadoMaestro eq "SANO") && ($estadoContrato eq "DUDOSO")){
      $igualMonto ? push(@consE2,$lineaConsulta) : push(@consF2,$lineaConsulta);
      }
      elsif (($estadoMaestro eq "DUDOSO") && ($estadoContrato eq "SANO")){
      $igualMonto ? push(@consE1,$lineaConsulta) : push(@consF1,$lineaConsulta);
      }

   }
   else{
# TODO llamas al glog. ("No existe el numero de contrato en el archivo maestro" SE)
   }
  }

  # Listados.
  print "LISTADO\n";
  my $consultaA = "Contratos comunes sanos con identico Monto Restante: \n".&procesarConsulta(@f
  print "$consultaA\n";

  my $consultaB = "Contratos comunes dudosos con identico Monto Restante: \n".&procesarConsulta(
  print "$consultaB\n";

  my $consultaC = "Contratos comunes sanos con diferente Monto Restante: \n".&procesarConsulta(@
  print "$consultaC\n";

  my $consultaD = "Contratos comunes dudosos con diferente Monto Restante: \n".&procesarConsulta
  print "$consultaD\n";

  my $consultaE = "Contratos comunes con diferente estado con identico Monto Restante: \n".
  &procesarConsulta(@filtrosPPI,@consE1)."\n".&procesarConsulta(@filtrosPPI,@consE2)."\n";
  print "$consultaE\n";

  my $consultaF = "Contratos comunes con diferente estado con diferente Monto Restante: \n".
  &procesarConsulta(@filtrosPPI,@consF1)."\n".&procesarConsulta(@filtrosPPI,@consF2)."\n";
  print "$consultaF\n";

  if ($grabarListados){
    open(LISTADOS,">$archivoListados");
    print LISTADOS $consultaA;
    print LISTADOS $consultaB;
    print LISTADOS $consultaC;
    print LISTADOS $consultaD;
```

```
    print LISTADOS $consultaE;
    print LISTADOS $consultaF;
    close(LISTADOS);
  };

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

  if ($grabarModificaciones){
    open(MODIFICACIONES,">$archivoModificaciones");
    print MODIFICACIONES $modicaciones;
    close(MODIFICACIONES);
  }

  print "Presione Enter para continuar";
  my $enter = <STDIN>;
}

sub menu{

  my $salir = 0;

  while (!$salir){

    system("clear");
    print "Paramatros 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();
```

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)