Índice de contenido

Hipótesis y Aclaraciones

○ *Registros*

En todos los archivos el separador de registros es el new line, no se espera encontrar un separador ";" al final.

○ *Órdenes de compra*

Existen múltiples archivos de ordenes de compra, tanto local como global. Siempre se utiliza el de número mayor (de extensión de archivo). Los demás quedan como registros históricos y no se utilizan en ningún proceso.

○ *Comando: invini*

Para saber si invonio esta corriendo, utiliza el comando auxiliar esta_corriendo. Este le indica si se esta corriendo el comando invonio que se encuentra en $grupo/comandos. Si existe otra instalación del sistema en un directorio diferente, la misma no afecta a la operación del mismo.

○ *Comando: invonio*

Sólo se procesan los archivos dentro de el directorio $grupo/arribos. Si existen subdirectorios dentro de $grupo/arribos, no serán procesados.

○ *Comando: remioc*

- Se presupone que todos los archivos (tanto de remitos como de ordenes de compra) tienen nombres válidos según corresponda.
- Cuando sobran productos, se guardan los datos en un archivo $grupo/sobrantes/sobrante.sob. Los registros tienen el siguiente formato:

Campo	Descripción
Número de orden de compra:	6 caracteres
Código de Producto a entregar:	10 caracteres
Cantidad sobrante:	Numérico, mayor que cero
Usuario de grabación:	N caracteres. Usuario que graba el registro
Fecha y hora de grabación:	N caracteres. Fecha y hora de grabación del registro

- Si se pasan por parámetro números de remito, no se da opción al usuario de elegir los remitos que quiere procesar.
- Si no se pasan parámetros, se toman todos los remitos que falten procesar.

○ *Comando: invreci*

- Todos los registros inválidos de un remito generan sólo un archivo .rech en $grupo/rechazados
- Existen varios archivos globales de ordenes de compra, sin embargo, para realizar el proceso se utilizara en todo momento, la versión mas reciente de los mismos.
- El archivo de ordenes de compra global tiene formato válido.
- Sólo se procesan los archivos dentro de el directorio $grupo/recibidos. Si existen subdirectorios dentro de $grupo/recibidos, no serán procesados.

○ *Función: modulo_mover*

- El comando admite un directorio como primer parámetro.
- Si el segundo parámetro es un directorio, verificamos que el archivo indicado en el parámetro 1 no resida en ese directorio. Si está en el mismo directorio no lo consideramos como archivo duplicado, sino que se trata del mismo archivo.

○ *Función: modulo_glog*

Cuando se invoca incorrectamente, sale con un código de error, pero no emite ningún tipo de error ni graba ningún log.

Problemas relevantes

◦ *Expansión del carácter "*" dentro de las ""*

Este problema se presentó al utilizar un for para recorrer los archivos del directorio. Al incluir el carácter "*" dentro de las "", el for toma como un único elemento la lista de los archivos contenidos separada por los espacios. Esto se solucionó colocando el carácter "*" fuera de las "".

◦ *Espacios en blanco en las rutas*

Al colocar espacios en las rutas de los directorios sobre los cuales se ejecutaba el programa, se observó que se debía "proteger" entre "" las rutas que utilizaban la variable de entorno $grupo, para evitar que los mismos produjeran errores a lo largo de la ejecución de cada uno de los comandos.

◦ *Espacios en blanco en los campos de los registros*

Al momento de probar el comando invreci, utilizando espacios en los registros, se presentó el problema de que el comportamiento del comando grep no era el esperado debido a los mismos. Por lo tanto, se procedió a reemplazar los espacios de los campos de los registros por el carácter "_", obteniéndose de este modo, el comportamiento esperado del grep.

◦ *Captura de señales para terminar un proceso*

Al desarrollar las funciones de start y stopinvonio para detener el proceso demonio, se intentó capturar la señal de kill. Luego de investigar, se observó que no era posible realizarlo de ese modo, dado que la señal de kill(9) no puede capturarse. Por lo tanto, se decidió capturar la señal de SIGTERM(15), y de este modo finalizar correctamente el comando invonio.

◦ *Problema con un flag del comando sed*

Inicialmente, para darle el formato correspondiente a los registros, se intentó utilizar el flag -i del comando sed. Al observar que no se lograba el comportamiento esperado del mismo, se decidió utilizar una variable auxiliar para modificar el formato y luego grabar el contenido de dicha variable en el archivo destino.

◦ *Error en la fórmula del cálculo de cumplimiento del enunciado*

Al desarrollar el comando occtrl utilizando la fórmula de cumplimiento dada en el enunciado se detectó un error. Al ejecutar el comando, con una orden de compra en la cual se esperaban, por ejemplo, 5(cinco) artículos y se habían recibido, por ejemplo, 0(cero); el resultado observado de la fórmula era el 100% de cumplimiento. Esto es claramente un error, dado que en el ejemplo planteado el grado de cumplimiento esperado sería del 0%. Dado esto, se decidió invertir el orden de la fórmula.

◦ *Problema para invocar el invreci sólo si no estaba ya ejecutándose*

Al tratar de invocar el comando invreci desde el comando invonio, se debió validar que dicho comando no este en ese momento en ejecución. Esto presentó algunos problemas, como por ejemplo, el no poder validarlo y que se invoque infinitas veces el proceso invreci sin importar si se encontraba ejecutándose o no. Para solucionarlo, se procedió a validarlo utilizando el comando ps y el flag eo pid (para obtener sólo los pid's de los procesos ejecutándose) y luego un grep con el PID del invreci que se lanzó anteriormente; observar que al lanzarse el comando invreci siempre se guarda el PID con el que se ejecutó.

◦ *Finalización del último campo del registro con ";"*

Al desarrollar el comando de validación de registros (invreci) se presentó el problema de no saber si el último campo de los registros finalizaba con un ";" (al igual que el resto de los campos). Para solucionar este problema, se tomó como convención que el mismo no finaliza con ";".

◦ *Parseo de parámetros de invocación del comando occtrl*

Al desarrollar este comando se presentó el problema de parsear los parámetros para la invocación del mismo. Este problema se presentó a causa de que la cantidad de parámetros es muy variable, con muchas combinaciones posibles y aceptándose también, la invocación del mismo sin parámetros (utilizando parámetros por defecto). Esto se solucionó estableciendo como convención que primero debe pasarse como parámetro la salida y luego la cantidad de ordenes de compra a procesar(rango,simple,todas,etc). También, para simplificar la lectura e interpretación del código perl se optó por utilizar el módulo Switch y de este modo evitar anidar varios if's de modo sucesivo.

◦ *Error en la fórmula de pendientes en occtrl del enunciado*

Al realizar el cálculo de pendientes, según la fórmula del enunciado, se detectó un error en la misma. Dado que, en esta, se calcula el pendiente como la cantidad total menos la cantidad remanente; lo cual es a nuestro entender incorrecto, dado que el pendiente es directamente la cantidad remanente de artículos.

Archivo README: Instructivo de Instalación

Para la instalación de la aplicación se deben seguir los siguientes pasos:

1. Insertar el dispositivo de almacenamiento con el contenido del tp (pen drive, cd, etc).

2. Copie el archivo grupo08.tgz a algún dispositivo local.

3. Ejecute el comando: tar xzf grupo08.tgz

4. El programa ya está listo para usarse. Si así lo desea, puede eliminar el archivo grupo08.tgz.

Diagrama de Procesos

- ***Diagrama de Proceso: Comando inovio***

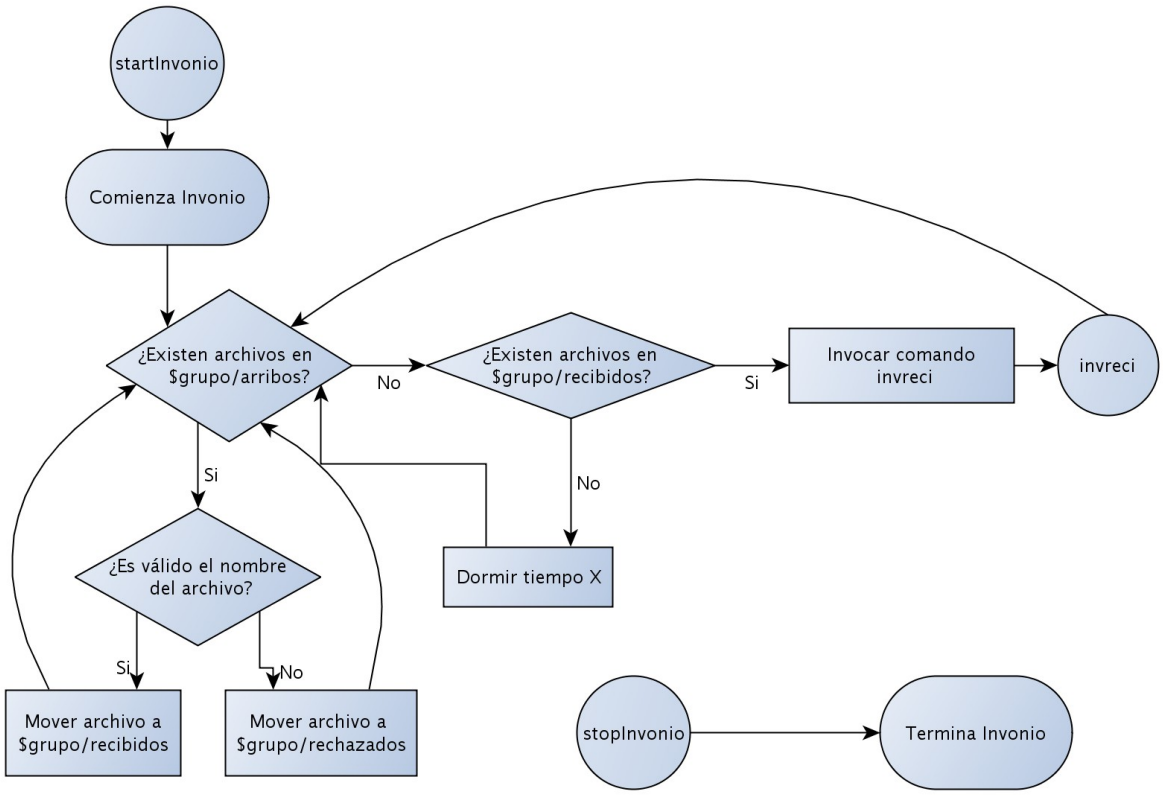

Diagrama de Proceso 1: Comando Invonio

- ***Diagrama de Proceso: Comando invreci***

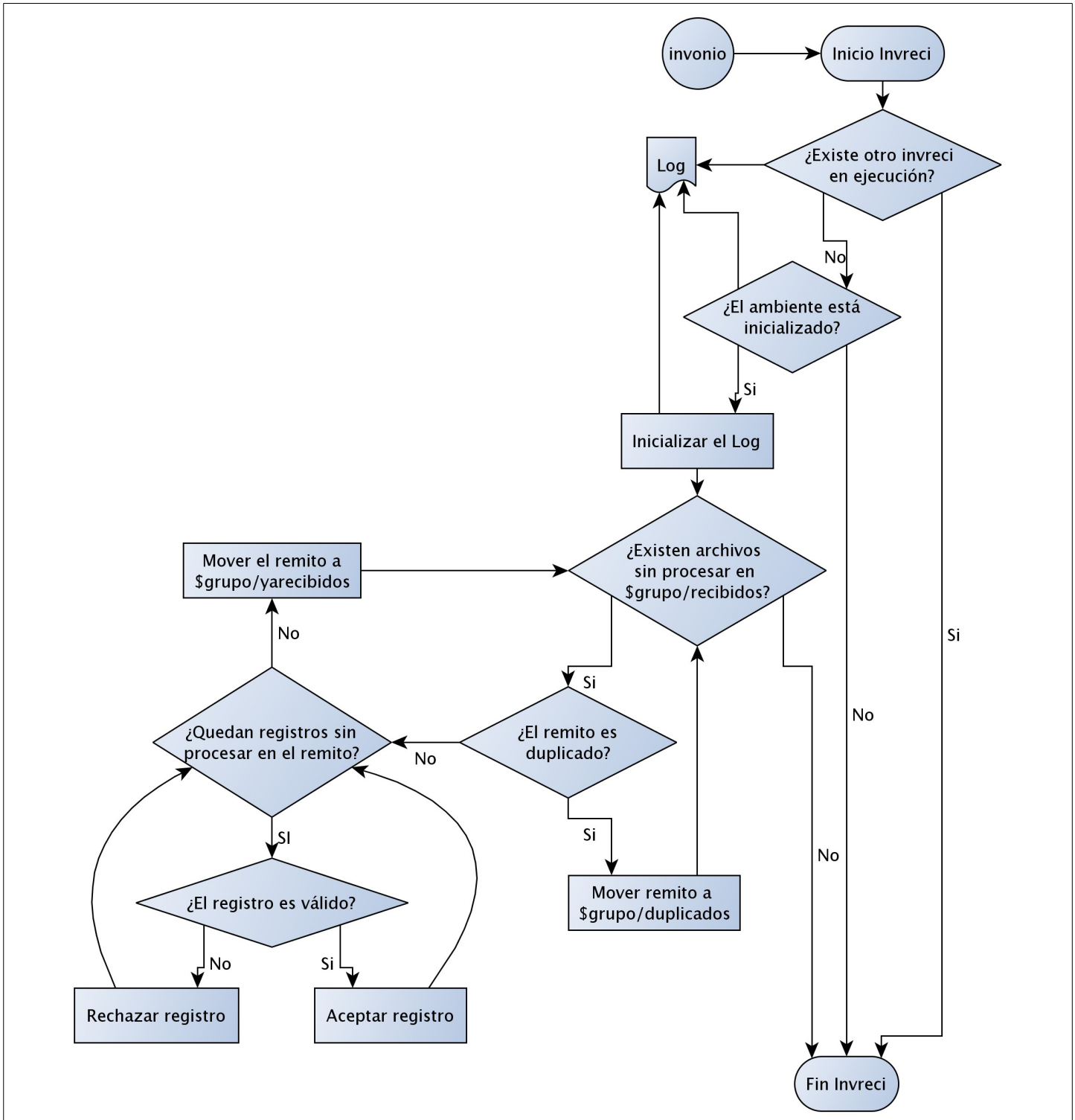

Diagrama de Proceso 2: Comando Invreci

- ***Diagrama de Proceso: Comando remioc***

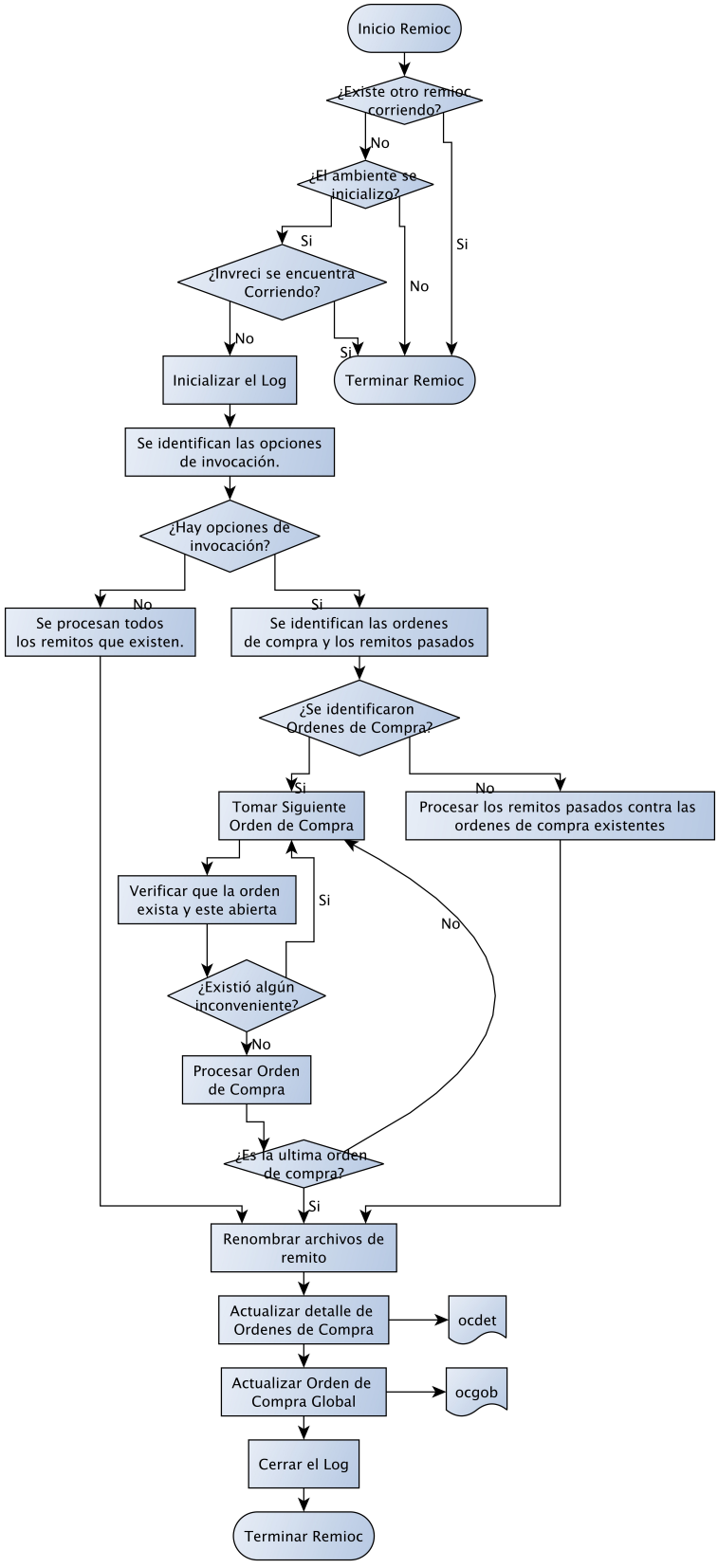

Diagrama de Proceso 3: Comando Remioc

- ***Diagrama de Proceso: Comando occtrl***

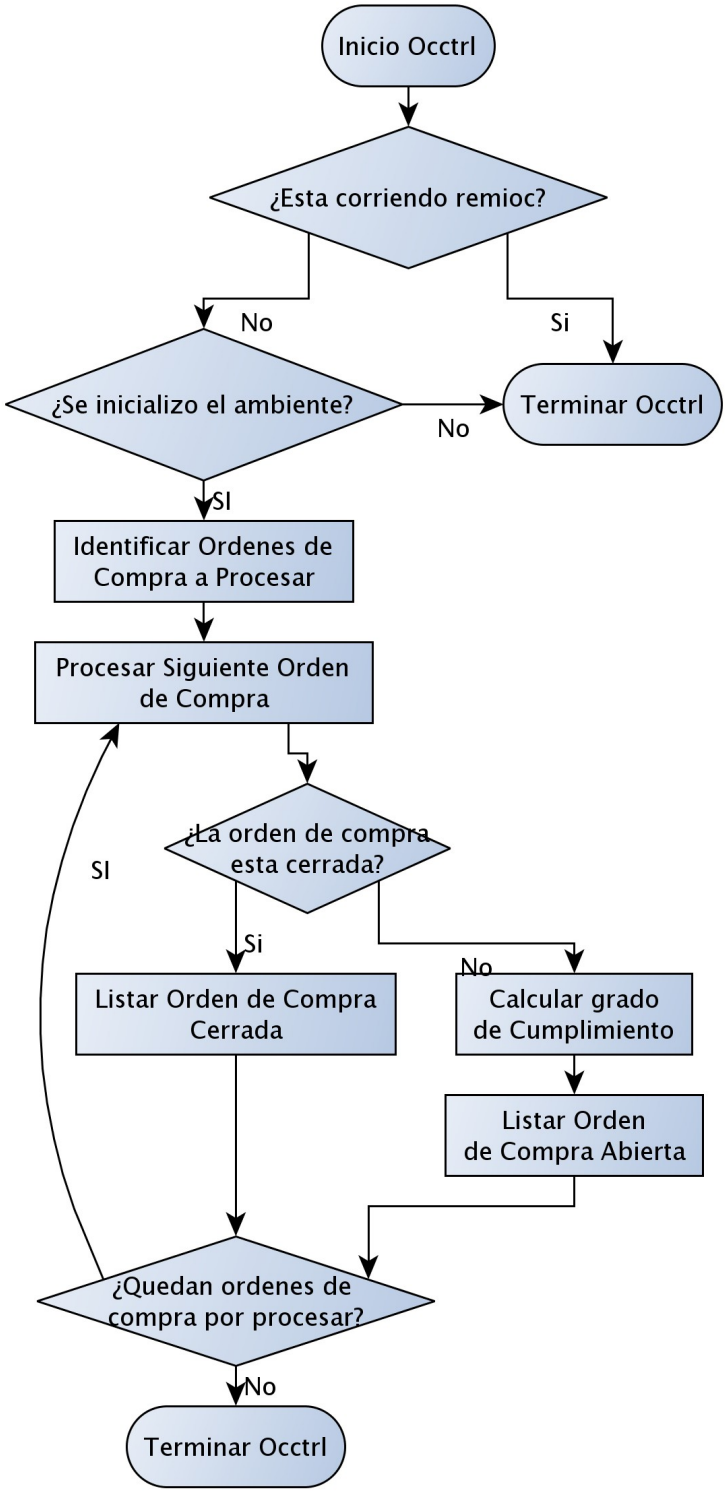

Diagrama de Proceso 4: Comando Occtrl

- ***Diagrama de Proceso: Proceso General***

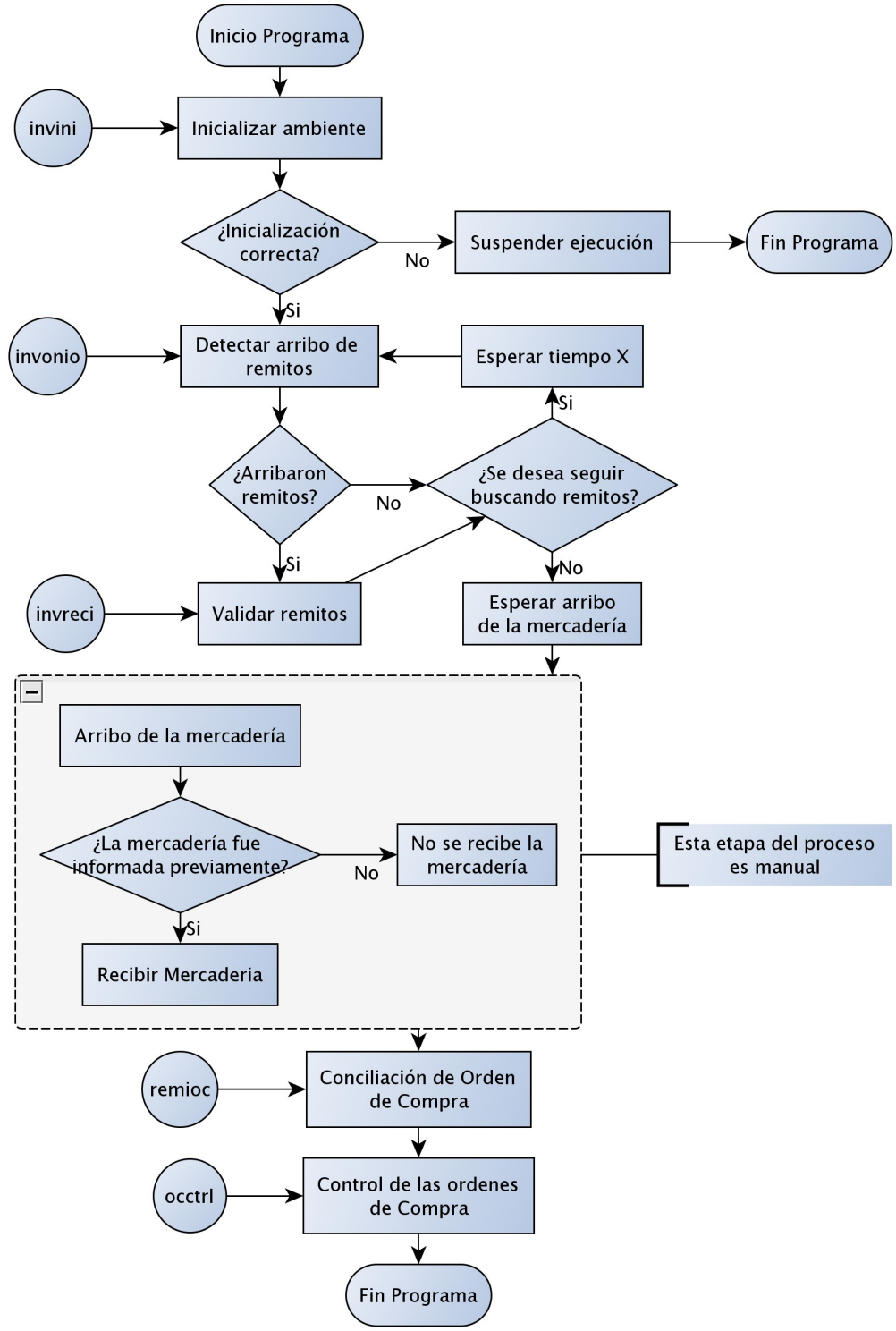

Diagrama de Proceso 5: General

Hoja de Ruta para la Corrección

Se recomienda la lectura del archivo README para realizar una correcta instalación del programa.

Para el resto de la explicación, supondremos que el directorio de instalación es */home/usuario/grupo*.

Para inicializar el sistema, debe posicionarse en el directorio de instalación:

```
# cd /home/usuario/grupo/comandos
```

A continuación, se debe ejecutar el siguiente comando:

```
# . invini
```

Esto inicializa algunas variables de entorno necesarias para la correcta ejecución de los módulos restantes. Asimismo, se lanza automáticamente un proceso *invonio* (en caso de que no este corriéndose aún).

A partir de este momento, cada archivo que exista dentro del directorio *arribos*, será procesado por el sistema y tratado acorde a su nombre y contenido.

En cualquier momento pueden utilizarse los comandos *remioc* (para procesar remitos contra ordenes de compra) y *occtrl* (para controlar los resultados del proceso).

Por ejemplo, para procesar las ordenes de compra 000001, 000002 y 000003, se debe ejecutar:

```
# remioc 000001 000002 000003
```

También se puede elegir procesar por número de remito. Por ejemplo para procesar los remitos 12345678 y 12345679:

```
# remioc 12345678 12345679
```

Por último, también se puede elegir procesar todos los remitos disponibles aún no procesados:

```
# remioc
```

Para verificar el estado de las ordenes de compra se utiliza el comando *occtrl*. Por ejemplo para verificar el estado de todas las órdenes de compra se utiliza de la siguiente manera:

```
# occtrl
```

Si se quiere verificar una orden de compra en particular, por ejemplo 000001

```
# occtrl -single 000001
```

También se pueden comprobar ordenes de compra por rango:

```
# occtrl -range 000001 123456
```

Si por algún motivo se desea frenar la ejecución del programa que procesa los *arribos*, se debe utilizar el comando *stopinvonio*. Para volver a ejecutar el proceso, se provee el comando *startinvonio*.

Casos de prueba incluidos

Se incluyen casos de prueba para verificar el correcto funcionamiento del sistema.

Para generar los casos de prueba, es necesario ejecutar los scripts generarCasosXXXXX donde XXXXX corresponde al nombre del comando para el cual se desea generar las pruebas (Remioc, Occtrl, Invonio, etc). También se provee un script *generarCasosPruebasGeneral* que genera archivos que permiten realizar una prueba integral del sistema. Dichos scripts se encuentran dentro del subdirectorio Pruebas.

Por ejemplo:

```
# cd /home/usuario/grupo/pruebas
# ./generarCasosPruebasGeneral
```

Comandos Solicitados

- ***Nombre del comando: invini***

Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplos de invocación:

- Sin inicializar previamente el ambiente(primera vez que se ejecuta el invini)

```
user@Fiuba:~$ . invini
     Inicialización de Ambiente Concluida.
       Ambiente: grupo=/home/user/workspace/7508/TP/trunk/TP/Pruebas/comandos/..
           PATH=/home/user/workspace/7508/TP/trunk/TP/Pruebas/comandos:/sbin:/usr/local/bin
:/usr/bin:/bin:/usr/games:/usr/lib/java/bin:/usr/lib/kde4/libexec:/opt/kde3/lib/qt3/bin:/opt/kde3/
bin:/usr/lib/qt/bin:/usr/share/texmf/bin:.
Demonio corriendo bajo el no.: 3226
```

- Una vez que invini ya fue ejecutado

```
user@Fiuba:~$ . invini
Inicialización de Ambiente No fue exitosa. El sistema ya esta inicializado.
```

Listado del código del comando:

```
#!/bin/bash
export grupo=`pwd`/..

export PATH=`pwd`:$PATH

error_inicializacion(){
    echo "Inicialización de Ambiente No fue exitosa. $@"

}

inicializar(){
    if [ ! -z "$SISTEMA_INICIALIZADO" ]
    then
       error_inicializacion "El sistema ya esta inicializado."
       return 1
    fi

    if [ ! -f "$grupo/oc/ocgob."?? ]
    then
       error_inicializacion "No existe el archivo global de ordenes de compra."
       return 1
    fi

    if [ ! -f "$grupo/oc/ocdet."?? ]
    then
       error_inicializacion "No existe el archivo de descripción de ordenes de compra."
```

```
      return 1
    fi

    for i in modulo_mover modulo_glog invonio invreci startinvonio stopinvonio
    do
      if [ ! -e "$grupo/comandos/$i" ]
      then
          error_inicializacion "No existe el archivo $i, necesario para la correcta ejecución del
programa."
          return 1
      fi
    done

    for i in remioc occtrl
    do
      if [ ! -e "$grupo/comandos/$i" ]
      then
          error_inicializacion "No existe el archivo $i. Esta funcionalidad no estará disponible."
          return 1
      fi
    done
    return 0
}

iniciar_invonio(){

    estaCorriendo "invonio"

    if [ $? -ne 0 ]
    then
      bash "$grupo/comandos/invonio" &
      PID_INVONIO=$!;
      return 0
    else
      local LINE_PID=`grep "PID=" "$grupo/locks/.lock_invonio_corriendo"`
      local PID=${LINE_PID:4}
      error_inicializacion "Invonio ya esta corriendo bajo el PID $PID."
      return 1
    fi
}

getFechaYHora(){
    echo -n `date +"%Y/%m/%d %H:%M:%S"`
    return $?
}

getUsuario(){
    echo -n `id -un`
    return $?
}

bloquearProceso(){
    if [ $# -ne 1 ]
    then
      return 1
    fi
    mkdir -p "$grupo/locks"

    local nombre=`basename "$1"`
    local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"

    if [ -e "$ARCHIVO_LOCK" ]
    then
      return 2
    fi
```

```
    # Creo un archivo lock oculto con el numero de pid del proceso
    echo PID=$$ > "$ARCHIVO_LOCK"

    return 0
}

desbloquearProceso(){
    if [ $# -ne 1 ]
    then
       return 1
    fi
    local nombre=`basename "$1"`
    local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"

    rm -rf "$ARCHIVO_LOCK" > /dev/null
    return $?
}

estaCorriendo(){
    if [ $# -ne 1 ]
    then
       return 1
    fi

    local nombre=`basename "$1"`
    local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"

    if [ -e "$ARCHIVO_LOCK" ]
    then
       return 0
    fi

    return 2
}

export -f getFechaYHora
export -f getUsuario
export -f bloquearProceso
export -f desbloquearProceso
export -f estaCorriendo

inicializar

if [ $? -eq 0 ]
then
    export SISTEMA_INICIALIZADO="OK"
    . "$grupo"/comandos/modulo_glog
    . "$grupo"/comandos/modulo_mover

    iniciar_invonio
    if [ $? -eq 0 ]
    then

       echo "Inicialización de Ambiente Concluida."
       echo "Ambiente: grupo=$grupo"
       echo "          PATH=$PATH"
       echo "Demonio corriendo bajo el no.: $PID_INVONIO"
    else
       echo "Inicialización de Ambiente Concluida."
       echo "Ambiente: grupo=$grupo"
       echo "          PATH=$PATH"
       echo "No se corrió invonio. Ya estaba ejecutandose."
    fi
fi
```

- ***Nombre del comando: startinvonio***

Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplos de invocación:

- Sin ejecutar el invonio previamente

```
user@Fiuba:~$ ./startinvonio
==> Demonio corriendo bajo el Nº.:<5279>
============================================================
Fecha actual: 20100508
------------------------------------------------------
El nombre del archivo es: 00000001.20100101
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000002.20100215
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000003.20100415
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000004.20100215
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000005.20100202
El archivo se coloco en la carpeta de recibidos
El PID del comando invreci es: 5311

Inicio de Invreci: Archivos a procesar : <5>
```

- Con el proceso invonio ya ejecutándose

```
user@Fiuba:~$ ./startinvonio
Ya existe un proceso invonio corriendo.
```

Listado del código del comando:

```
#!/bin/bash

#Si NO esta inicializado el ambiente
if [ -z $SISTEMA_INICIALIZADO ]
then
    echo Error: El ambiente no fue inicializado
    exit 1
fi

# Verifico que no exista un proceso invonio corriendo
estaCorriendo invonio
if [ $? -ne 0 ]
then
    ./invonio&
else
    echo "Ya existe un proceso invonio corriendo."
fi
```

○ *Nombre del comando: stopinvonio*

Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplos de invocación:

- Con el proceso invonio corriendo

```
user@Fiuba:~$ ./stopinvonio
##################################
 ++ Señal SIGTERM para Invonio ++
##################################
============================================================
Fin del proceso invonio Nº.:<4654>
```

- Con el proceso invonio sin ejecutarse

```
user@Fiuba:~$ ./stopinvonio
No existe ningun proceso invonio corriendo.
```

Listado del código del comando:

```
#!/bin/bash

#Si NO esta inicializado el ambiente
if [ -z $SISTEMA_INICIALIZADO ]
then
    echo Error: El ambiente no fue inicializado
    exit 1
fi

# Verifica que exista un proceso invonio corriendo
estaCorriendo invonio
if [ $? -eq 0 ]
then
       LINE_PID=`grep "PID=" "$grupo/locks/.lock_invonio_corriendo"`
       PID=${LINE_PID:4}
       kill -15 $PID
else
    echo "No existe ningun proceso invonio corriendo."
fi
```

- ### *Nombre del comando: invonio*

Archivos de Input:	Los remitos que se encuentran en la carpeta $grupo/arribos.
Archivos de Output:	Los remitos aceptados se ubican en la carpeta $grupo/recibidos. Los remitos rechazados se colocan en la carpeta $grupo/rechazados.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplos de invocación:

- Sin ejecutar el invonio previamente

```
user@Fiuba:~$ ./invonio
==> Demonio corriendo bajo el Nº.:<5279>
===========================================================
Fecha actual: 20100508
-------------------------------------------------------
El nombre del archivo es: 00000001.20100101
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000002.20100215
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000003.20100415
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000004.20100215
El archivo se coloco en la carpeta de recibidos
El nombre del archivo es: 00000005.20100202
El archivo se coloco en la carpeta de recibidos
El PID del comando invreci es: 5311

Inicio de Invreci: Archivos a procesar : <5>

================================================
```

- Con el proceso invonio ya ejecutándose

```
user@Fiuba:~$ ./invonio
Ya existe un proceso invonio corriendo.
```

Listado del código del comando:

```
#!/bin/bash

# Salidas: 1 : Error de ambiente no inicializado

# Constantes
MAX_REMITO=99999999
MIN_REMITO=00000000
TIEMPO=2
PRIMERA=1

# Funciones
function handler_sigterm(){
	echo
	echo \#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#
	echo " ++ Señal SIGTERM para Invonio ++ "
	echo \#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#\#
```

```
      terminar=1
}

# Catcher
trap "handler_sigterm" SIGTERM

#Si NO esta inicializado el ambiente
if [ -z $SISTEMA_INICIALIZADO ]
then
    echo Error: El ambiente no fue inicializado
    exit 1
fi

# ++ Programa INVONIO ++ #
echo
echo "==> Demonio corriendo bajo el N°.:"\<$$\>
echo ============================================================

# Creo un archivo lock oculto con el numero de pid del proceso
bloquearProceso "$0"
if [ $? -ne 0 ]
then
    echo Ya existe un proceso invonio corriendo, se termina la ejecucion.
    Glog invreci "Ya existe un proceso invonio corriendo, se termina la ejecucion." W
    exit 1
fi

# Almaceno la fecha actual
aaaa=`date +%Y`
mm=`date +%m`
dd=`date +%d`
echo Fecha actual: $aaaa$mm$dd
echo  ------------------------------------------------------

terminar=0

while [ $terminar -eq 0 ]
do
  for path_archivo in "$grupo/arribos/"*
  do
    if [ -f "$path_archivo" ]
    then
         archivo=${path_archivo##*/}
         echo "El nombre del archivo es: $archivo"

         #Verifico el formato del nombre del archivo
         echo $archivo | grep "^[0-9]\{8\}\.[0-9]\{8\}$" -q

         if [ $? = 0 ]
         then
                error_remito=0
                remito=${archivo%.*}
                # Verifico el remito
                if [ $remito -ge $MAX_REMITO ] || [ $remito -le $MIN_REMITO ]
                then
                       error_remito=1
                       echo El remito se encuentra fuera de los limites
                fi
                # Verifico la fecha
                # Formato fecha aaaammdd
                error_fecha=0
                fecha=${archivo#*.}
                anio=${fecha:0:4}
                mes=${fecha:4:2}
                dia=${fecha:6:2}
                if [ $anio -lt "0000" ] || [ $anio -gt "9999" ]
```

```
                then
                       error_fecha=1
                       echo El anio del archivo es un anio invalido
                fi

                if [ $mes -le "00" ] || [ $mes -gt "12" ]
                then
                       error_fecha=1
                       echo El mes del archivo es un mes invalido
                fi

                # Valida que la cantidad de dias de la fecha sea correcta para el mes del que se
trata
                #31 dias: enero,marzo,mayo.julio,agosto,octubre,diciembre
                #30 dias: abril,junio,setpiembre,noviembre
                #29 dias: febrero si el a?o es divisible por 400 y no por 100, sino 28 dias.

                case $mes in
                [01,03,05,07,08,10,12]) if [ $dia -gt "31" ]
                                        then
                                            error_fecha=1
                                            echo La fecha del archivo es una fecha invalida : 31 dias
                                        fi;;
                [04,06,09,11]) if [ $dia -gt "30" ]
                               then
                                     error_fecha=1
                                     echo La fecha del archivo es una fecha invalida : 30 dias
                               fi;;
                02) if [ $(( $anio % 400 )) -eq "0" ] && [ $(( $anio % 100 )) -ne "0" ]
                    then
                       if [ $dia -gt "29" ]
                       then
                           error_fecha=1
                           echo La fecha del archivo es una fecha invalida : Febrero 29 dias
                       fi
                    elif [ $dia -gt "28" ]
                    then
                         error_fecha=1
                         echo La fecha del archivo es una fecha invalida : Febrero 28 dias
                    fi;;
                esac

                if [ $anio -gt $aaaa ]
                then
                       error_fecha=1
                       echo La fecha es mayor a la actual [año mayor al actual]
                else
                       if [ $anio -eq $aaaa ] &&  [ $mes -gt $mm ]
                       then
                              error_fecha=1
                              echo La fecha es mayor a la actual [igual año, mes mayor al actual]
                       else
                              if [ $anio -eq $aaaa ] &&  [ $mes -eq $mm ] && [ $dia -gt $dd ]
                              then
                                     error_fecha=1
                                     echo La fecha es mayor a la actual [igual año y mes, dia mayor
al actual]
                              fi
                       fi
                fi

                # Si existe algun error en el nombre del archivo lo muevo a la carpera
/grupo/rechazados,
                # sino /grupo/aceptados
                if [ $error_remito -eq "1" ] || [ $error_fecha -eq "1" ]
                then
```

```
                        Mover "$grupo/arribos/$archivo" $grupo/rechazados/ invonio
                        echo El archivo se coloco en la carpeta de rechazados
                else
                        Mover "$grupo/arribos/$archivo" $grupo/recibidos/ invonio
                        echo El archivo se coloco en la carpeta de recibidos
                fi
        else
                    Mover "$grupo/arribos/$archivo" $grupo/rechazados/ invonio
                    echo El formato del nombre del archivo es invalido
        fi

        # Si hay una señal de SIGTERM, interrumpo el loop
        if [ $terminar -eq 1 ]; then break; fi


    fi
  done

  AUXILIARNOMBRE=`ls -A "$grupo/recibidos/"`
  if [ !  -z "$AUXILIARNOMBRE" ];
  then
     # Ejecutar invreci
     # Validar que no este corriendo ya el invreci siempre q haya archivos en recibidos
     # e invreci no este corriendo.

     if [ $PRIMERA -ne 0  ]
     then
             ./invreci&
             PIDINVRECI=$!
             # Si se ejecuto correctamente, muestro el PID de invreci
             echo El PID del comando invreci es: $PIDINVRECI
             PRIMERA=0
     else
             # grep devuelve 0 si encontro las lineas, y 1 en otro caso.
             ps -eo pid | grep "$PIDINVRECI" -q
             if [ $? -ne 0 ]
             then

                     ./invreci&
                     PIDINVRECI=$!
                     # Si se ejecuto correctamente, muestro el PID de invreci
                     echo
                     echo Se ejecuta el comando invreci. Su PID es : $PIDINVRECI
                     echo
             else
                    echo Error: ya se esta ejecutando el comando invreci en este momento.
             fi
     fi


  fi



 sleep $TIEMPO

done

# Elimino el archivo lock oculto
desbloquearProceso "$0"

echo ============================================================
echo "Fin del proceso invonio N°.:"\<$$\>
```

- ***Nombre del comando: invreci***

Archivos de Input:	Los remitos procesados por invonio, que se encuentran en la carpeta $grupo/recibidos. Las ordenes de compra global, que se encuentran en la carpeta $grupo/oc.
Archivos de Output:	Los remitos aceptados se colocan en la carpeta $grupo/aceptados con la extensión aproc. Los remitos rechazados se los ubica en la carpeta $grupo/rechazados. Los remitos que no se encuentran duplicados pasan a $grupo/yarecibidos.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplos de invocación:

```
user@Fiuba:~$ . /invreci
Inicio de Invreci: Archivos a procesar : <5>

================================================
Archivo a procesar: <00000001.20100101>

---------------------------------------
Se valida el siguiente registro :
000000;0000000001;2;10;12345.6;20100101;Algun_lugar;Desconocido;00000000001;Aseguradora
El registro de orden de compra es: 000000;20100101;00000000001;ABIERTA;LuCaS;01/01/2010 15:20:00
Se acepta el registro
---------------------------------------
Se valida el siguiente registro :
000000;0000000002;3;7;98452.12;20100101;Algun_lugar;Desconocido;00000000001;Aseguradora
El registro de orden de compra es: 000000;20100101;00000000001;ABIERTA;LuCaS;01/01/2010 15:20:00

Se acepta el registro
---------------------------------------
Se valida el siguiente registro :
000000;0000000002;1;2;01236.21;20100101;Algun_lugar;Desconocido;00000000001;Aseguradora
El registro de orden de compra es: 000000;20100101;00000000001;ABIERTA;LuCaS;01/01/2010 15:20:00
Se acepta el registro
---------------------------------------


Resumen del procesamiento del remito:

         -Remito Aceptado:<00000001.20100101>
         -Cantidad de Registros Leidos: 3
         -Cantidad de Registros Aceptados: 3
         -Cantidad Registros Rechazados por Orden de Compra Cerrada: 0
         -Cantidad De Registros Rechazados por Orden de Compra Inexistente: 0
         -Cantidad de Registros Rechazados por Otros Motivos: 0
```

- Si no hay archivos para procesar

```
user@Fiuba:~$ . /invreci

Inicio de Invreci: Archivos a procesar : <0>

No existen archivos para procesar, se suspende la ejecucion de invreci
```

- Si ya se estaba ejecutando el comando invreci

user@Fiuba:~$. /invreci

```
Ya existe un proceso invreci corriendo, se termina la ejecucion.
```

Listado del código del comando:

```
#!/bin/bash

cantidadRegistrosLeidos=0
cantidadRegistrosAceptados=0
cantidadRegistrosRechazadosPorOCompCerrada=0
cantidadRegistrosRechazadosPorOCompInexistente=0
cantidadRegistrosRechazadosPorOtrosMotivos=0

# Inicializar el log grabando inicio de invreci y la cantidad de archivos a procesar

#Si NO esta inicializado el ambiente
if [ -z $SISTEMA_INICIALIZADO ]
then
    echo Error: El ambiente no fue inicializado, no se continua con la ejecucion de invreci.
    Glog invreci "El ambiente no fue inicializado, no se continua con la ejecucion de invreci " W
    exit 1
fi

# Verifica que no exista un proceso invreci corriendo
bloquearProceso "$0"
if [ $? -ne 0 ]
then
    echo Ya existe un proceso invreci corriendo, se termina la ejecucion.
    Glog invreci "Ya existe un proceso invreci corriendo, se termina la ejecucion." W
    exit 1
fi

rechazarRegistro(){

    echo $1 >> "$grupo"/rechazados/$2.rech

}

#Devuelve 0 si el registro es valido.
#        1 si es rechazado por orden de compra cerrada
#        2 si es rechazado por orden de compra inexistente
#        3 si es rechazado por numero de CUIT invalido
validarRegistro(){

    local registroAValidar=$1
    local nombreArchivo=$2
    local ordendecompra=` echo $registroAValidar | cut -d ';' -f 1`
    local cuitProveedor=` echo $registroAValidar | cut -d ';' -f 9`

    echo Se valida el siguiente registro : "$registroAValidar"
    Glog invreci "$registroAValidar" i
    Glog invreci "Se procesa la orden de compra # $ordendecompra" i

    local archivosAProcesar2=`ls "$grupo"/oc/ -1 | wc -l`
    if [ $archivosAProcesar2 -eq 0 ]
    then
       Glog invreci "No existen archivos de orden de compra para procesar, se suspende la ejecucion
de invreci" e
       echo "No existen archivos de orden de compra para procesar, se suspende la ejecucion de
invreci"
       exit 1
```

```
    fi

    #Se busca la ultima orden de compra global
    ultimoArchivo=0
    for archivoOrdenCompra in "$grupo/oc/ocgob."*
    do
       actual=${archivoOrdenCompra##*.}
       if [ $ultimoArchivo -lt $actual ]
       then
           ultimoArchivo=$actual
       fi
    done

    #Validacion del formato del registro
    local registroOrdenDeCompra=`grep -h "^$ordendecompra" "$grupo/oc/ocgob.$ultimoArchivo"` #Esto
devuelve una lista o nada.

    if [ "$registroOrdenDeCompra" != "" ]
    then
       echo "El registro de orden de compra es: $registroOrdenDeCompra"
       if [  "`echo $registroOrdenDeCompra | cut -d ';' -f 4`" = "CERRADA" ]
       then
         echo La orden de compra $ordendecompra esta cerrada, se rechaza el registro
         Glog invreci "La orden de compra $ordendecompra esta cerrada, se rechaza el registro" e

         rechazarRegistro "$registroAValidar" "$nombreArchivo"
         local cantidadRegistrosRechazadosPorOCompCerrada=$
(( $cantidadRegistrosRechazadosPorOCompCerrada + 1))
         return 1
       elif [ "$cuitProveedor" != "`echo $registroOrdenDeCompra | cut -d ';' -f 3`" ]
       then
           echo "El CUIT "`echo $registroOrdenDeCompra | cut -d ';' -f 3`" es invalido, se rechaza
el registro (se esperaba $cuitProveedor)"
           Glog invreci "El CUIT es invalido, se rechaza el registro" e
           rechazarRegistro "$registroAValidar" "$nombreArchivo"
           cantidadRegistrosRechazadosPorOtrosMotivos=$
(($cantidadRegistrosRechazadosPorOtrosMotivos + 1))
           return 3
       else
         #Validacion correcta
           echo "Se acepta el registro"
           Glog invreci "Se acepta el registro" e
           return 0
       fi
     fi

    echo "La orden de compra $ordendecompra no existe, se rechaza el registro"
    Glog invreci "La orden de compra #$ordendecompra no existe, se rechaza el registro"     e
    rechazarRegistro "$registroAValidar" "$nombreArchivo"
    cantidadRegistrosRechazadosPorOCompInexistente=$
(($cantidadRegistrosRechazadosPorOCompInexistente +1))
    return 2
}


#Comienzo :
#Guarda en el log "Inicio de Invreci: <cantidad de archivos a procesar>

archivosAProcesar=`ls "$grupo"/recibidos/ -1 | wc -l`
Glog invreci "Inicio de Invreci: Archivos a procesar : <$archivosAProcesar> " i
echo
echo "Inicio de Invreci: Archivos a procesar : <$archivosAProcesar> "
echo

if [ $archivosAProcesar -eq 0 ]
then
```

```
    Glog invreci "No existen archivos para procesar, se suspende la ejecucion de invreci" e
    echo "No existen archivos para procesar, se suspende la ejecucion de invreci"
    exit 1
fi

for archivoRecibido in "$grupo/recibidos/"*
do
    archivo=`basename "$archivoRecibido"`

    Glog invreci "Archivo a procesar: <$archivo>" i
    Glog invreci "-------------------------------" i
    echo "================================================"
    echo "Archivo a procesar: <$archivo>"
    echo



    if [ -e "$grupo"/yarecibidos/$archivo ]
    then

       Glog invreci "Remito Rechazado Por Duplicado: <$archivo>" w
       echo "Remito Rechazado Por Duplicado: <$archivo>"
       Mover $archivoRecibido "$grupo"/rechazados/ invreci

    else
       cantidadRegistrosLeidos=0
       cantidadRegistrosAceptados=0
       cantidadRegistrosRechazadosPorOCompCerrada=0
       cantidadRegistrosRechazadosPorOCompInexistente=0
       cantidadRegistrosRechazadosPorOtrosMotivos=0
       cantidadRegistrosLeidos=$cantidadRegistrosRechazadosPorOtrosMotivos

       #Rechaza un registro porque el formato es invalido
         RechazadosFormato=`grep -v "^[0-9]\{6\};.\{10\};[0-9]*;[0-9]*;[0-9]*\.\?[0-9]*;[0-9]\
{8\};.\{11\};.\{11\};.\{11\};.\{11\}$" $archivoRecibido`

       if [ $? -eq 0 ] && [ ! -z $RechazadosFormato ]
       then
            echo "Los registros rechazados por formato inválido son :"
            echo $RechazadosFormato
          echo $RechazadosFormato >> "$grupo"/rechazados/${archivo}.rech
            cantidadRegistrosRechazadosPorOtrosMotivos=`wc -l "$grupo"/rechazados/${archivo}.rech |
cut -d ' ' -f 1`
       fi

       for registro in `grep '^[0-9]\{6\};.\{10\};[0-9]*;[0-9]*;[0-9]*\.\?[0-9]*;[0-9]\{8\};.\
{11\};.\{11\};.\{11\};.\{11\}$' $archivoRecibido | sed 's/ /_/g'`
       do

           cantidadRegistrosLeidos=$(($cantidadRegistrosLeidos + 1))
           echo "----------------------------------------"
           validarRegistro "$registro" "$archivo"
           Glog invreci "-------------------------------" i

           if [ "$?" -eq 0 ]
           then
              numeroremito=${archivo%%.*}
              fecha=${archivo##*.}
              numeroordencompra=` echo $registro | cut -d ';' -f 1`
              cantidadRegistrosAceptados=$(($cantidadRegistrosAceptados + 1))

              registro=`echo $registro | sed "s/^[^;]*;/${fecha};/"`

              echo "$registro" >> "$grupo"/aceptados/${numeroremito}.${numeroordencompra}.aproc

           fi
```

```
      done
      Mover $archivoRecibido "$grupo"/yarecibidos/ invreci
   fi

   if [ $cantidadRegistrosLeidos -eq 0 ]
   then
      Glog invreci " -Remito Rechazado Por Archivo Vacio: <$archivo> " i
      echo "-Remito Rechazado Por Archivo Vacio:<$archivo> "
   else
      Glog invreci "-Remito Aceptado:<$archivo> " i
      Glog invreci "-Cantidad de Registros Leidos: $cantidadRegistrosLeidos " i
      Glog invreci "-Cantidad de Registros Aceptados: $cantidadRegistrosAceptados " i
      Glog invreci "-Cantidad Registros Rechazados por Orden de Compra Cerrada:
$cantidadRegistrosRechazadosPorOCompCerrada " i
      Glog invreci "-Cantidad De Registros Rechazados por Orden de Compra Inexistente:
$cantidadRegistrosRechazadosPorOCompInexistente " i
      Glog invreci "-Cantidad de Registros Rechazados por Otros Motivos:
$cantidadRegistrosRechazadosPorOtrosMotivos " i
      echo "--------------------------------------


Resumen del procesamiento del remito:

       -Remito Aceptado:<$archivo>
       -Cantidad de Registros Leidos: $cantidadRegistrosLeidos
         -Cantidad de Registros Aceptados: $cantidadRegistrosAceptados
         -Cantidad Registros Rechazados por Orden de Compra Cerrada:
$cantidadRegistrosRechazadosPorOCompCerrada
         -Cantidad De Registros Rechazados por Orden de Compra Inexistente:
$cantidadRegistrosRechazadosPorOCompInexistente
         -Cantidad de Registros Rechazados por Otros Motivos:
$cantidadRegistrosRechazadosPorOtrosMotivos
      "
   fi
done

Glog invreci "Fin de Invreci" i

# Se elimine el archivo de lock
desbloquearProceso "$0"
```

- ***Nombre del comando: remioc***

Archivos de Input:	Los remitos aceptados que se encuentran en la carpeta $grupo/yarecibidos con la extensión aproc. Las ordenes de compra global y detalle, que se encuentran en la carpeta $grupo/oc.
Archivos de Output:	Luego del procesamiento a los remito se les cambia la extensión a proc. y se los coloca en la carpeta $grupo/aceptados. Al finalizar el proceso de apareo, se generara una nueva versión del archivo de detalle de orden de compra actualizado, que se guarda en la carpeta $grupo/oc. El archivo de detalle de órdenes de compra ocdet posee un número de versión a modo de extensión de archivo. Si todos los registros de detalle de la orden de compra quedaron en estado CERRADO, se cierra también la orden de compra. Se genera una nueva versión del archivo global de orden de compra, que se almacena en la carpeta $grupo/oc. El archivo global de orden de compra ocgob posee un número de versión a modo de extensión de archivo. El archivo original no debe ser removido, permanece a modo de histórico.
Parámetros:	El ingreso de estos parámetros no tiene un orden; pudiéndose ingresar solo uno de ellos, como ambos. 1. **Parámetro 1 (opcional):** Orden / órdenes de compra se quieren conciliar contra el universo de los remitos aceptados a procesar. 2. **Parámetro 2 (opcional):** Remito / remitos aceptados se quieren conciliar.
Opciones:	Ninguna.

Ejemplos de invocación:

```
user@Fiuba:~$ ./remioc
Este comando escribe en el log, se muestra la salida que se vuelca en el mismo.
2010/05/08 22:48:37 <I> Inicio de remioc:
2010/05/08 22:48:37 <I> Remito disponible: 00000001.000000.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000002.000001.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000003.000001.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000004.123456.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000002.000001.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000003.000001.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000004.123460.aproc
2010/05/08 22:48:37 <I> Remito disponible: 00000005.123461.aproc
2010/05/08 22:48:37 <I> Conciliación de la orden de compra 000000 000001 123456 000001 123460
123461.
2010/05/08 22:48:42 <I> Remito elegido: 00000005.123461.aproc
2010/05/08 22:48:42 <I> Renombrando aceptados/00000005.123461.aproc a
aceptados/00000005.123461.proc
```

- Invocación con parámetros

```
user@Fiuba:~$ ./remioc [parametros]
Nota: Parametros es un valor numerico de 6 a 8 caracteres y pueden ser de 0 a N parametros.
user@Fiuba:~$ ./remioc 000000 000001
```

```
Este comando escribe en el log, se muestra la salida que se vuelca en el mismo.
2010/05/10 10:31:59 <I> Inicio de remioc: 000000 000001
2010/05/10 10:31:59 <I> Remito disponible: 00000001.000000.aproc
2010/05/10 10:31:59 <I> Remito disponible: 00000002.000001.aproc
2010/05/10 10:31:59 <I> Remito disponible: 00000003.000001.aproc
2010/05/10 10:31:59 <I> Conciliación de la orden de compra 000000 000001.
2010/05/10 10:33:24 <I> Remito elegido: 00000001.000000.aproc
2010/05/10 10:33:24 <I> Renombrando aceptados/00000001.000000.aproc a
aceptados/00000001.000000.proc
```

- Invocación con parámetros inválidos

```
user@Fiuba:~$ ./remioc 00000 aaa000
Error: '00000', no identifica una orden de compra o remito.
Este comando escribe en el log, se muestra la salida que se vuelca en el mismo.
2010/05/10 10:36:32 <I> Inicio de remioc: 00000 aaa000
2010/05/10 10:36:32 <E> Error: 00000, no identifica una orden de compra o remito.
```

Listado del código del comando:

```
#!/usr/bin/perl

# Salidas:  0 - OK
#           1 - Sistema no inicializado
#           2 - Ya existe otro proceso remioc
#           3 - Existe un proceso invreci corriendo
#           4 - No se encuentra el archivo de ordenes de compra global
#           5 - Error de parámetros. Se espera un numero de orden de compra (6 caracteres) o de
remito (8 caracteres).
#           6 - Todas las ordenes de compra a procesar estan cerradas.
#           7 - No existen remitos disponibles para las ordenes de compra especificadas.
#           8 - No se eligió ningún remito.
#           9 - No se encontró el archivo de descripción de ordenes de compra.
#           10 - No se especifico ningun remito u orden de compra.
#           11 - No se pudo abrir el archivo de detalle. Las operaciones no se terminaron.
#           12 - No se pudo crear un nuevo archivo de detalle. Las operaciones no se terminaron.
#           13 - No se pudo abrir el archivo global de orden de compra. Las operaciones no se
terminaron.
#           14 - No se pudo crear un nuevo archivo global de orden de compra. Las operaciones no
se terminaron.


use strict;

$0="remioc";

my $SISTEMA_INICIALIZADO = $ENV{'SISTEMA_INICIALIZADO'};
my $grupo = $ENV{'grupo'};

if(!$SISTEMA_INICIALIZADO){
    print "El sistema no esta inicializado.\n";
    exit 1;
}


sub Glog{
    my $mensaje=shift(@_);
    my $tipo=shift(@_);
    $mensaje =~ s/ /\\ /g;
    `/bin/bash -c "Glog $0 $mensaje $tipo"`;
}


`/bin/bash -c "bloquearProceso $0"`;
if( $? != 0 ){
    print "Error: Ya existe un proceso remioc corriendo.\n";
```

```
    &Glog("Ya existe un proceso remioc corriendo.", "SE");
    exit 2;
}

`/bin/bash -c "estaCorriendo invreci"`;
if( $? == 0 ){
    `/bin/bash -c "desbloquearProceso $0"`;
    print "Error: Ya existe un proceso invreci corriendo.\n";
    &Glog("Ya existe un proceso invreci corriendo.", "SE");
    exit 3;
}

#busca todos los remitos que tienen el numero de orden de compra
#especificado
sub buscarRemitos{
    # $1 -> numero de OC
    my $NUMERO = shift(@_);

    opendir(DIR, "$grupo/aceptados/");
    my @RDISPONIBLES = grep(/.*\.$NUMERO\.aproc$/,readdir(DIR));
    closedir(DIR);

    foreach (@RDISPONIBLES) {
       &Glog("Remito disponible: $_", "I");
    }

    return @RDISPONIBLES;
}

#Da al usuario la posibilidad de elegir de entre todos los remitos,
#cuáles quiere procesar
sub elegirRemitos(){

    return () if ($#_+1 == 0);

    my $FILAS;
    my %ARCHIVOS;
    foreach (@_){
       (my $CODIGO) = $_;
       if($CODIGO){
           $FILAS .= "$CODIGO - dummy ";
           $ARCHIVOS{$CODIGO} = $_;
       }
    }

    my $ELEGIDOS=`dialog  --checklist \"Lista de remitos\"  24 50 12 $FILAS --stdout 2>/dev/null`;

    return () if $? != 0;

    (my @TEMPORAL) = $ELEGIDOS =~ /"([^"]*)"/g;
    my @RELEGIDOS;

    foreach (@TEMPORAL){
       push(@RELEGIDOS, $ARCHIVOS{$_});
    }

    foreach (@RELEGIDOS) {

       &Glog("Remito elegido: $_", "I");
    }
    return @RELEGIDOS
}

#Busca el ultimo archivo de ordenes de compra
sub buscarUltimaOC(){
    #$_[0] -> ocgob o ocdet
```

```
    my $INICIAL=01;
    my $NUMERO=0;
    my $otrogrupo = $grupo;
    $otrogrupo =~ s/ /\\ /g;
    my @CANDIDATOS = <$otrogrupo/oc/$_[0].*>;

    foreach (@CANDIDATOS){

       ($NUMERO) = $_ =~ /\.([0-9]*)$/;

       if($NUMERO >= $INICIAL){
           $INICIAL=$NUMERO;
       }
    }

    return $NUMERO;
}

#Procesa la orden de compra con los remitos elegidos.
sub procesarOrden{

    my $OCDET = shift(@_);
    my (@NUMEROORDEN) = @{$_[0]};
    my (@REMITOS) = @{$_[1]};


    my %ORDENES;
    @ORDENES{@NUMEROORDEN} = ();


    # extraigo la información de todos los productos de los remitos
    my %PRODUCTOS = ();
    for my $origen (@REMITOS){

       (my $destino) = $origen =~ /^(.*)\.aproc$/;
       $destino .= ".proc";
       &Glog("Renombrando aceptados/$origen a aceptados/$destino", "I");
       if (! rename "$grupo/aceptados/$origen", "$grupo/aceptados/$destino"){
           &Glog("No se puede renombrar aceptados/$origen a aceptados/$destino", "SE");
           next; #salteo el remito (no lo proceso)
       }

       if(open(my $archivo, '<', "$grupo/aceptados/$destino")){
           while(<$archivo>){
              my $CODPROD = $_;
              ($CODPROD) = $CODPROD =~ /^[^;]*;([^;]*);/;
              my $CANTIDAD = $_;
              ($CANTIDAD) = $CANTIDAD =~ /^[^;]*;[^;]*;([^;]*);/;

              #para el hash, uso la clave: NumeroOrdenDeCompra+CodigoProducto
              (my $CLAVE) = $destino =~ /^[^.]*\.([^.]*)\..*$/;
              $CLAVE .= $CODPROD;
              $PRODUCTOS{$CLAVE} += $CANTIDAD;

           }

           close $archivo;

       }
       else{
           print "Error: No se puede abrir $grupo/aceptados/$destino, se saltea.\n";
           &Glog("No se puede abrir $grupo/aceptados/$destino, se saltea.", "SE");
       }

    }
```

```
    #ahora, genero un nuevo OCDET y voy procesando los productos
    my %DEBOCERRAR;
    @DEBOCERRAR{@NUMEROORDEN} = ();

    (my $OCDET2) = $OCDET =~ /^(.*)\..*$/;
    (my $NUMERO) = $OCDET =~ /^.*\.(.*)$/;

    $NUMERO++;
    $NUMERO = sprintf "%.2i",($NUMERO);
    $OCDET2 .= ".$NUMERO";

    my ($archivo,$archivo2);

    if (! open $archivo, '<', $OCDET){
       print "Error: No se pudo abrir el archivo de detalle. Las operaciones no se terminaron.\n";
       &Glog("No se pudo abrir el archivo de detalle. Las operaciones no se terminaron.", "SE");
       return 11;
    }
    if (! open $archivo2, '>', $OCDET2){
       print "Error: No se pudo crear un nuevo archivo de detalle. Las operaciones no se
terminaron.\n";
       &Glog("No se pudo crear un nuevo archivo de detalle. Las operaciones no se terminaron.",
"SE");
       return 12;
    }

    while(my $linea = <$archivo>){

       (my $CODIGOORDEN) = $linea =~ "^([^;]*);";

       if( exists $ORDENES{$CODIGOORDEN} ){
           #la linea es parte de la orden de compra que me interesa (alguna)
           (my $CODPROD, my $REMANENTE) = $linea =~ "^[^;]*;[^;]*;([^;]*);[^;]*;([^;]*);";

           my $CLAVE = $CODIGOORDEN;
           $CLAVE .= $CODPROD;

           my $ESTADO = "ABIERTO";
           if($REMANENTE <= $PRODUCTOS{$CLAVE}){
              $PRODUCTOS{$CLAVE} -= $REMANENTE;
              $REMANENTE=0;
              $ESTADO = "CERRADO";
           }
           else{
              $REMANENTE -= $PRODUCTOS{$CLAVE};
              $PRODUCTOS{$CLAVE} = 0;
              delete $DEBOCERRAR{$CODIGOORDEN};
           }
           my $usuario=`/bin/bash -c getUsuario`;
           my $fecha=`/bin/bash -c getFechaYHora`;

           $linea =~ s/^([^;]*;[^;]*;[^;]*;[^;]*);[^;]*;[^;]*;.*$/$1;$REMANENTE;$ESTADO;$usuario;
$fecha/;

           print $archivo2 $linea;
       }
       else{
           #la linea no me interesa, la guardo sin modificarla
           print $archivo2 $linea;
       }
    }

    close $archivo;
    close $archivo2;

    mkdir "$grupo/sobrantes/";
```

```
    my $abierto = open my $archivoSobrante, ">>", "$grupo/sobrantes/sobrante.sob";
    for my $clave (keys %PRODUCTOS){
       if($PRODUCTOS{$clave} > 0){
            (my $CODIGOORDEN) = $clave =~ /^(......)/;
            (my $CODIGOPROD) = $clave =~ /(..........)$/;
           print "Error: sobraron $PRODUCTOS{$clave} unidades del producto $CODIGOPROD, en la orden
de compra $CODIGOORDEN\n";
           &Glog("Sobraron $PRODUCTOS{$clave} unidades del producto $CODIGOPROD, cuando se
intentaba conciliar la orden de compra $CODIGOORDEN", "E");
           if($abierto){
              my $usuario=`/bin/bash -c getUsuario`;
              my $fecha=`/bin/bash -c getFechaYHora`;
              print $archivoSobrante "$CODIGOORDEN;$CODIGOPROD;$PRODUCTOS{$clave};$usuario;
$fecha\n";
           }
       }
    }

    #Me fijo si tengo que cerrar alguna orden de compra en el global
    my @claves = keys %DEBOCERRAR;
    if( @claves > 0 ){

       my $NUMERO = &buscarUltimaOC("ocgob");
       my $OCGOB = "$grupo/oc/ocgob.$NUMERO";
       $NUMERO++;
       $NUMERO = sprintf "%.2i",($NUMERO);
       my $OCGOB2 = "$grupo/oc/ocgob.$NUMERO";

       my ($archivo,$archivo2);

       if (!open $archivo, '<', $OCGOB){
           print "Error: No se pudo abrir el archivo global de orden de compra. Las operaciones no
se terminaron.\n";
           &Glog("No se pudo abrir el archivo global de orden de compra. Las operaciones no se
terminaron.", "SE");
           return 13;
       }

       if (!open $archivo2, '>', $OCGOB2){
           print "Error: No se pudo crear un nuevo archivo global de orden de compra. Las
operaciones no se terminaron.\n";
           &Glog("No se pudo crear un nuevo global de orden de compra. Las operaciones no se
terminaron.", "SE");
           return 14;
       }


       while(my $linea = <$archivo>){
           (my $CODIGOORDEN) = $linea =~ "^([^;]*);";

           if( exists $DEBOCERRAR{$CODIGOORDEN}){
              #me interesa esta linea. La cierro

              my $usuario=`/bin/bash -c getUsuario`;
              my $fecha=`/bin/bash -c getFechaYHora`;

              $linea =~ s/^([^;]*;[^;]*;[^;]*);[^;]*;.*$/$1;CERRADA;$usuario;$fecha/;
              print $archivo2 $linea;
           }
           else{
              #no me interesa la linea, la dejo como está.
              print $archivo2 $linea;
           }

       }
```

```
       close $archivo;
       close $archivo2;
    }
}

sub main{

#Inicializar el log
    &Glog("Inicio de $0: @ARGV", "I");

    my $PARAMETRO;
    my $NUMERO;
    my @OC;
    my $OCDET;
    my $OCGOB;
    my $ULTIMO;
    my @REMITOS;
    my $FLAGREMITOS=0;

    $ULTIMO=&buscarUltimaOC("ocgob");
    if (! $ULTIMO){
       print "Error: No existe el archivo global de ordenes de compra $ULTIMO.\n";
       &Glog("No existe el archivo global de ordenes de compra.", "E");
       return 4;
    }
    $NUMERO=$PARAMETRO;

#Por cada parametro, miro si tiene 6 u 8 caracteres. Si tiene 6, me
#guardo el numero como una orden de compra a procesar. Si tiene 8, lo
#tomo como un numero de remito y busco si existe. Si existe, me guardo
#el numero de remito y el numero de orden de compra asociada.

    foreach $PARAMETRO (@ARGV){

       if(length($PARAMETRO) == 6){
           push (@OC, $PARAMETRO);
       }
       elsif(length($PARAMETRO) == 8){

           opendir my $DIRECTORIO, "$grupo/aceptados/";


           my @archivos = readdir $DIRECTORIO;
           closedir $DIRECTORIO;
           @archivos = grep(/$PARAMETRO\..*\.aproc/, @archivos);

           my $archivo;

           foreach $archivo (@archivos){
              if( -e "$grupo/aceptados/$archivo" ){
                   ($NUMERO) = $archivo =~ /$PARAMETRO\.([0-9]{6})\.aproc$/;
                   push (@OC, $NUMERO);
                   $FLAGREMITOS=1;
                   push(@REMITOS , "$PARAMETRO.$NUMERO.aproc");

              }
              else{
                   print "Error: no existe el remito $NUMERO (archivo $archivo).\n";
                   &Glog("Error: no existe el remito $NUMERO (archivo $archivo).","E");
              }
           }

       }
       else{
           print "Error: '$PARAMETRO', no identifica una orden de compra o remito.\n";
           &Glog("Error: '$PARAMETRO', no identifica una orden de compra o remito.","E");
```

```
            return 5;
        }
    }

    if( $#ARGV == -1 ){
        #busco todos los remitos

        opendir my $directorio,"$grupo/aceptados/";

        my @archivos = readdir($directorio);
        closedir($directorio);

        foreach (@archivos){

            next if !( $_ =~ /[0-9]{8}\.[0-9]{6}\.aproc/);

            ($NUMERO) = $_ =~ /[0-9]{8}\.([0-9]{6})\.aproc$/;
            push (@OC, $NUMERO);
            push(@REMITOS , "$_");
        }

        if(@REMITOS){
            print "Se procesarán los siguientes remitos: @REMITOS\n";
        }

    }

    $OCGOB="$grupo/oc/ocgob.$ULTIMO";

#Si tenemos alguna orden de compra
    if( @OC ){
        #Verifico que esté abierta

        my @aux=@OC;
        @OC = ();
        #verifico que las ordenes de compra elegidas esten abiertas.
        foreach (@aux){
            `grep "^$_;[0-9]\\{8\\};[0-9]\\{11\\};ABIERTA;.*" "$OCGOB"`;
            if( $? == 0 ){
                push(@OC, $_);
            }
            else{
                print "Error: La orden de compra $_, no está abierta.\n";
                &Glog("La orden de compra $_, no está abierta.", "E");
            }
        }

        if(!@OC){
            print "Todas las ordenes de compra especificadas estan cerradas.\n";
            &Glog("Todas las ordenes de compra especificadas estan cerradas.", "I");
            return 6;
        }

        if(! $FLAGREMITOS){ #Si la invocacion del programa no incluyó remitos
            foreach (@OC){
                #Busco los remitos que se corresponden a cada orden de compra
                my @RDISPONIBLES = &buscarRemitos($_);
                if( ! @RDISPONIBLES){
                    print "No hay remitos disponibles para la orden de compra $NUMERO\n";
                    &Glog("No hay remitos disponibles para la orden de compra $NUMERO.", "I");
                }
                else{
                    push(@REMITOS, @RDISPONIBLES);
                }
            }
        }
```

```
      my %auxiliar;
      @auxiliar{@REMITOS} = ();

      @REMITOS = keys %auxiliar;

      if( ! @REMITOS ) {
          print "No hay remitos disponibles para las ordenes de compra solicitadas (@OC).\n";
          &Glog("No hay remitos disponibles para las ordenes de compra solicitadas @OC.", "I");
          return 7;
      }

      &Glog("Conciliación de la orden de compra @OC.", "I");

      if(! $FLAGREMITOS){ #Si la invocacion del programa no incluyó remitos
          #Dejo al usuario elegir los remitos a procesar
          @REMITOS = &elegirRemitos(@REMITOS);
      }


      if( ! @REMITOS){
          print "No se eligió ningun remito.\n";
          &Glog("No se eligió ningun remito.","W");
          return 8;
      }

      #Busco el ultimo archivo de descripción de ordenes de compra
      $ULTIMO = &buscarUltimaOC("ocdet");
      my $OCDET = "$grupo/oc/ocdet.$ULTIMO";

      if( ! $OCDET ){
          print "Error: No se encontro el archivo de descripción de ordenes de compra.";
          &Glog("No se encontro el archivo de descripción de ordenes de compra.", "SE");
          return 9;
      }

      #Procesa cada detalle de orden de compra utilizando los
      #remitos elegidos
      &procesarOrden($OCDET, \@OC, \@REMITOS);
   }
   else{
      print "Error: No se especifico nungun remito/orden de compra.\n";
      &Glog("No se especifico nungun remito/orden de compra.", "E");
      return 10;
   }
}


my $retorno = &main;

`/bin/bash -c "desbloquearProceso $0"`;

exit $retorno;
```

- ***Nombre del comando: occtrl***

Archivos de Input:	Las ordenes de compra global y detalles, que se encuentran en la carpeta $grupo/oc.
Archivos de Output:	Los archivos con el resultado del comando; puede ser un archivo específico o salida standard.
Parámetros:	1. **Parámetro 1 (opcional):** El parámetro permite definir el formato de salida. 2. **Parámetro 2 (opcional):** El parámetro permite definir que ordenes de compra se quieren controlar (todas, por rango, específica)
Opciones:	1. **Parámetro 1 (opcional):** • **-f :** Se ingresa el nombre del archivo en el que se desea guardar el informe. • **-std :** El informe se mostrará por salida standard. • **-b :** Habilita ambas salidas. **2. Parámetro 2 (opcional):** • **-all:** Es el parámetro por defecto. El rango mínimo es 0 y el rango máximo es 999999. • **-range:** Se ingresa un mínimo y un máximo para determinar un rango. • **-single:** Se ingresa el número de una orden de compra a buscar.

Ejemplos de invocación:

- Invocación sin parámetros, se utilizan los parámetros por defecto(salida estandar y todas las ordenes de compra)

```
user@Fiuba:~$ ./occtrl
 Estado de la orden     : ABIERTA
 Orden de compra numero : 000000
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100101
 Pendiente              : 6
 Proveedor              : 00000000001

 Estado de la orden     : ABIERTA
 Orden de compra numero : 000001
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100102
 Pendiente              : 5
 Proveedor              : 00000000002

 Estado de la orden     : ABIERTA
 Orden de compra numero : 000002
```

```
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100205
 Pendiente              : 8
 Proveedor              : 00000000003

 Estado de la orden     : ABIERTA
 Orden de compra numero : 123456
 Grado de cumplimiento  : 42.86
 Fecha de OC            : 20100210
 Pendiente              : 8
 Proveedor              : 30324567820

 Estado de la orden     : ABIERTA
 Orden de compra numero : 123460
 Grado de cumplimiento  : 25.00
 Fecha de OC            : 20100210
 Pendiente              : 6
 Proveedor              : 30324567891

 Estado de la orden     : ABIERTA
 Orden de compra numero : 123461
 Grado de cumplimiento  : 22.73
 Fecha de OC            : 20100210
 Pendiente              : 34
 Proveedor              : 30324567891
```

- Invocación con parámetros

```
user@Fiuba:~$ ./occtrl -f salida -all
user@Fiuba:~$ cat salida
 Orden de compra numero : 000000
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100101
 Pendiente              : 6
 Proveedor              : 00000000001
 Orden de compra numero : 000001
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100102
 Pendiente              : 5
 Proveedor              : 00000000002
 Orden de compra numero : 000002
 Grado de cumplimiento  : 0.00
 Fecha de OC            : 20100205
 Pendiente              : 8
 Proveedor              : 00000000003
 Orden de compra numero : 123456
 Grado de cumplimiento  : 42.86
 Fecha de OC            : 20100210
 Pendiente              : 8
 Proveedor              : 30324567820
 Orden de compra numero : 123460
 Grado de cumplimiento  : 25.00
 Fecha de OC            : 20100210
 Pendiente              : 6
 Proveedor              : 30324567891
 Orden de compra numero : 123461
 Grado de cumplimiento  : 22.73
 Fecha de OC            : 20100210
 Pendiente              : 34
 Proveedor              : 30324567891

user@Fiuba:~$ ./occtrl -b salida -single 123461
 Estado de la orden     : ABIERTA
 Orden de compra numero : 123461
```

```
 Grado de cumplimiento  : 22.73
 Fecha de OC            : 20100210
 Pendiente              : 34
 Proveedor              : 30324567891
user@Fiuba:~$ cat salida1
Orden de compra numero : 123461
 Grado de cumplimiento  : 22.73
 Fecha de OC            : 20100210
 Pendiente              : 34
 Proveedor              : 30324567891

user@Fiuba:~$ ./occtrl -std -range 123456 123461
Estado de la orden     : ABIERTA
 Orden de compra numero : 123456
 Grado de cumplimiento  : 42.86
 Fecha de OC            : 20100210
 Pendiente              : 8
 Proveedor              : 30324567820

 Estado de la orden     : ABIERTA
 Orden de compra numero : 123460
 Grado de cumplimiento  : 25.00
 Fecha de OC            : 20100210
 Pendiente              : 6
 Proveedor              : 30324567891

 Estado de la orden     : ABIERTA
 Orden de compra numero : 123461
 Grado de cumplimiento  : 22.73
 Fecha de OC            : 20100210
 Pendiente              : 34
 Proveedor              : 30324567891
```

Listado del código del comando:

```
#!/usr/bin/perl

# Salidas:  0 - OK
#           1 - Sistema no inicializado
#           2 - Ya se encuentra corriendo otra instancia de occtrl
#           3 - No se encontró el archivo de ordenes de compra global
#           4 - No se encontró el archivo de detalles de ordenes de compra
#          5 - Se encuentra corriendo remioc

use strict;
use Switch;

$0 = "occtrl";

my $SISTEMA_INICIALIZADO = $ENV{'SISTEMA_INICIALIZADO'};
my $grupo = $ENV{'grupo'};

if(!$SISTEMA_INICIALIZADO){
    print "El sistema no esta inicializado.\n";
    exit 1;
}

sub salir{
    #=============================================================
    #Desbloquear proceso.
    `/bin/bash -c "desbloquearProceso occtrl"`;
    #=============================================================

    exit $_;
}
```

```
#Busca el ultimo archivo de ordenes de compra
sub buscarUltimaOC(){
    #$_[0] -> ocgob o ocdet
    my $INICIAL=01;
    my $NUMERO=0;
    my $otrogrupo = $grupo;
    $otrogrupo =~ s/ /\\ /g;
    my @CANDIDATOS = <$otrogrupo/oc/$_[0].*>;

    foreach (@CANDIDATOS){

       ($NUMERO) = $_ =~ /\.([0-9]*)$/;

       if($NUMERO >= $INICIAL){
           $INICIAL=$NUMERO;
       }
    }

    return $NUMERO;
}

#============================================================
#Si se encuentra corriendo el proceso remioc suspender la ejecucion

`/bin/bash -c "estaCorriendo remioc"`;

if ( $? == 0 ){
    print "Se esta ejecutando el comando remioc en este momento, se suspende la ejecucion.\n";
    exit 5;
}


#============================================================
#Si el proceso ya se encuentra corriendo suspender la ejecucion

`/bin/bash -c "estaCorriendo occtrl"`;

if ( $? == 0 ){
    print "Ya se encuentra corriendo un proceso occtrl.\n";
    exit 2;
}

`/bin/bash -c "bloquearProceso occtrl"`;
#============================================================

my $NUMEROARCHIVO = &buscarUltimaOC("ocgob");
my $TOTAL = 0;
my $REMANENTE = 0;

open (ARCHIVO,'<',"$grupo/oc/ocgob.$NUMEROARCHIVO") or &salir(3);



#Formato programa -salida -all
#        programa -salida -range mrangominimo rangomaximo
#       programa -salida -single numero

#Parseo de parametros
my $ARGC = @ARGV;
my $RANGOMINIMO = 0;
my $RANGOMAXIMO = 999999;
my $ARCH = 0;
my $STD = 0;
my $PROXIMOARG;
if ($ARGC != 0){
```

```
    switch ($ARGV[0]) {
       case "-f" { open (ARCHIVOSALIDA , '>>', "$ARGV[1]"); $ARCH = 1; $PROXIMOARG = 2 ;}
       case -std { $STD = 1; $PROXIMOARG = 1;}
       case "-b" { $STD = 1; $ARCH = 1 ; open (ARCHIVOSALIDA , '>>', "$ARGV[1]") ; $PROXIMOARG =
2;}
       else  { print "El parametro correspondiente a la salida es invalido, se procesa con el
parametro por defecto.\n";
               $STD = 1; $PROXIMOARG = 0;}
    }

    switch ($ARGV[$PROXIMOARG]){
       case -all { $RANGOMINIMO = 0; $RANGOMAXIMO = 999999;}
       case -range { $RANGOMINIMO = $ARGV[$PROXIMOARG+1]; $RANGOMAXIMO = $ARGV[$PROXIMOARG+2];}
       case -single { $RANGOMINIMO = $ARGV[$PROXIMOARG+1]; $RANGOMAXIMO = $ARGV[$PROXIMOARG+1];}
       else { print "El parametro correspondiente al rango de ordenes de compra es invalido, se
procesa con el parametro por defecto.\n";
              $RANGOMINIMO = 0; $RANGOMAXIMO = 999999;}
    }

}
else{
    $STD=1;
}


while (my $LINEA = <ARCHIVO>){
    (my $ESTADO) = $LINEA =~ "^[^;]*;[^;]*;[^;]*;([^;]*);";
    (my $NUMEROOC) = $LINEA =~ "^([^;]*);";
    (my $FECHAOC) = $LINEA =~ "^[^;]*;([^;]*);";
    (my $CUIT) = $LINEA =~ "^[^;]*;[^;]*;([^;]*);";
    (my $FECHAGRABACION) = $LINEA =~ "^[^;]*;[^;]*;[^;]*;[^;]*;[^;]*;([^;]*)";
    chomp $FECHAGRABACION;
    if ( $NUMEROOC >= $RANGOMINIMO && $NUMEROOC <= $RANGOMAXIMO ) {
       $TOTAL=0;
       $REMANENTE=0;
       my $CUMPLIMIENTO=100;
       my $PENDIENTE=0;
       if ($ESTADO eq "ABIERTA") {
           #Obtener las sumas totales de las cantidades de la orden de compra detallada.
           $NUMEROARCHIVO = &buscarUltimaOC("ocdet");
           open(ARCHIVO2, '<',"$grupo/oc/ocdet.$NUMEROARCHIVO") or &salir(4);
           while (my $LINEA2 = <ARCHIVO2>){
              (my $NUMEROOCDETALLE) = $LINEA2 =~ "^([^;]*);";
              if ( $NUMEROOC == $NUMEROOCDETALLE ) {
                  (my $auxiliar) = $LINEA2 =~ "^[^;]*;[^;]*;[^;]*;([^;]*);";
                  $TOTAL+=$auxiliar;
                  ($auxiliar) = $LINEA2 =~ "^[^;]*;[^;]*;[^;]*;[^;]*;([^;]*);";
                  $REMANENTE += $auxiliar;
              }
              if ( $TOTAL != 0){
                  $CUMPLIMIENTO = 100 - (($REMANENTE) / $TOTAL) * 100;
              }
              $PENDIENTE = $REMANENTE;

           }
           close ARCHIVO2;
       }

       if ( $STD ) {
           printf ( "\n" );
           printf ( " Estado de la orden     : %s \n",$ESTADO);
           printf ( " Orden de compra numero : %s \n",$NUMEROOC);
           printf ( " Grado de cumplimiento  : %.2f \n",$CUMPLIMIENTO);
           printf ( " Fecha de OC            : %s \n",$FECHAOC);
           if ($ESTADO eq "CERRADA"){
```

```
            printf ( " Fecha de cierre        : %s \n",$FECHAGRABACION);
          }else { printf (" Pendiente             : %i \n",$PENDIENTE);}
          printf ( " Proveedor             : %s \n",$CUIT);

      }
      if ( $ARCH ) {
          printf (ARCHIVOSALIDA " Orden de compra numero : %s \n",$NUMEROOC);
          printf (ARCHIVOSALIDA " Grado de cumplimiento  : %.2f \n",$CUMPLIMIENTO);
          printf (ARCHIVOSALIDA " Fecha de OC            : %s \n",$FECHAOC);
          if ($ESTADO eq "CERRADA"){
             printf (ARCHIVOSALIDA " Fecha de cierre        : %s \n",$FECHAGRABACION);
          }else { printf (ARCHIVOSALIDA " Pendiente             : %i \n",$PENDIENTE);}
          printf (ARCHIVOSALIDA " Proveedor              : %s \n",$CUIT);
      }


    }

}
close ARCHIVO;
close ARCHIVOSALIDA if ($ARCH);

&salir(0);
```

- ***Nombre del comando: modulo_mover***

Archivos de Input:	El archivo definido en el parámetro 1.
Archivos de Output:	El archivo definido en el parámetro 2. El archivo de Log del comando que la invoca (si corresponde).
Parámetros:	1. **Parámetro 1 (obligatorio):** Origen. 2. **Parámetro 2 (obligatorio):** Destino. 3. **Parámetro 3 (obligatorio):** El nombre del comando que la invoca.
Opciones:	Ninguna.

Ejemplos de invocación:

- Invocación sin parámetros:

```
user@Fiuba:~$ Mover
Error: el comando necesita al menos 2 (dos) parametros

Invocacion con parámetros válidos: (Mueve el archivo archivo al directorio directorio)

user@Fiuba:~$ Mover archivo directorio/
```

- Invocación con archivo inexistente:

```
user@Fiuba:~$ Mover archivo directorio/
Error: El archivo archivo no existe.
```

- Invocación con tercer parámetro (opcional):

```
user@Fiuba:~$ Mover archivo log/ remioc
Crea entradas en el log de remioc:
user@Fiuba:~$ cat $grupo/comandos/log/remioc.log
2010/05/10 17:16:09 <I> Se mueve el archivo archivo a log/archivo
2010/05/10 17:16:11 <I> Se mueve el archivo archivo a log/dup/archivo.000
2010/05/10 17:16:14 <I> Se mueve el archivo archivo a log/dup/archivo.001
```

Listado del código del comando:

```
#! /bin/bash

# Salidas:  0 - OK
#           1 - Error de parámetros
#           2 - Archivo inexistente
#           3 - Directorio inexistente
#           4 - Sistema no inicializado

if [ -z "$SISTEMA_INICIALIZADO" ]
then
    echo "Error, no esta inicializado el sistema."
    return 4
fi

Mover(){
```

```
local DUP="dup"
local ESCRIBIR_LOG=""

#Primero verificamos que la cantidad de parametros sea como minimo 2 (dos)
if [ $# -le 1 ]
then
   echo "Error: el comando necesita al menos 2 (dos) parametros"
   Glog "Mover" "Error: el comando necesita al menos 2 (dos) parametros" "E"
   return 1
fi

if [ $# -ge 3 ]
then
   case "$3" in
       "invreci") ESCRIBIR_LOG=1 ;;
       "remioc")  ESCRIBIR_LOG=1 ;;
   esac
fi

#verificamos que exista el primer archivo
if [ ! -e "$1" ]
then
   echo "Error: El archivo $1 no existe."
   Glog "Mover" "Error: El archivo $1 no existe." "E"

   if [ ! -z $ESCRIBIR_LOG ]
   then
       Glog "$3" "Error: El archivo $1 no existe." "E"
   fi

   return 2
fi

#Verificamos que origen y destino no sean iguales
if [ "$1" = "$2" ]
then
   Glog "Mover" "Origen y destino son iguales" "I"

   if [ ! -z $ESCRIBIR_LOG ]
   then
       Glog "$3" "Origen y destino son iguales" "I"
   fi

   return 0
fi

#idem, pero para el caso que el segundo parámetro sea un directorio
if [ -d "$2" ]
then
   local directorio=`dirname "$1"`
   if [ "$directorio" = "." ]
   then
       directorio=`pwd`
   fi

   if [ "$directorio" = "$2" ]
   then
       Glog "Mover" "Origen y destino son iguales" "I"

       if [ ! -z $ESCRIBIR_LOG ]
       then
          Glog "$3" "Origen y destino son iguales" "I"
       fi
       return 0
   fi
fi
```

```
directorio="${2%/*}"
if [ ! -d  "$directorio" ]
then
   echo "Error: El directorio $2 no existe."
   Glog "Mover" "Error: El directorio $2 no existe." "E"
   if [ ! -z $ESCRIBIR_LOG ]
   then
       Glog "$3" "Error: El directorio $2 no existe." "E"
   fi
   return 3
fi

local archivo="${2##*/}"

if [ "$archivo" = "" ]
then
   archivo=`basename "$1"`
fi

if [ -e "$directorio/$archivo" ]
then
   #si existe, es un archivo duplicado
   if [ ! -d "$directorio/$DUP" ]
   then
       #si no existe el directorio de duplicados, lo creo
       rm -f "$directorio/$DUP" > /dev/null 2> /dev/null
       mkdir "$directorio/$DUP"
   fi

   local numero=0
   local i=0
   for i in "$directorio/$DUP/$archivo".*
   do

       local actual="${i##*.}"
       if [ "$actual" != "*" ]
       then
          if [ $((10#$actual)) -ge $((10#$numero)) ]
          then
              numero=$(( 10#$actual + 1 ))
          fi
       fi
   done
   numero=`printf "%.3i" $numero`
   mv "$1" "$directorio/$DUP/$archivo.$numero"

   Glog "Mover" "Se mueve el archivo $1" "I"
    Glog "Mover" "a $directorio/$DUP/$archivo.$numero" "I"
   if [ ! -z $ESCRIBIR_LOG ]
   then
       Glog "$3" "Se mueve el archivo $1" "I"
        Glog "$3" "a $directorio/$DUP/$archivo.$numero" "I"
   fi

else
   mv "$1" "$directorio/$archivo"
   Glog "Mover" "Se mueve el archivo $1" "I"
    Glog "Mover" "a $directorio/$archivo" "I"
   if [ ! -z $ESCRIBIR_LOG ]
   then
       Glog "$3" "Se mueve el archivo $1" "I"
        Glog "$3" "a $directorio/$archivo" "I"
   fi
fi
```

```
    return 0
}

export -f Mover
```

- ***Nombre del comando: modulo_glog***

Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	1. **Parámetro 1 (obligatorio):** El nombre del comando que la invoca. 2. **Parámetro 2 (obligatorio):** El mensaje a guardar en el log. 3. **Parámetro 3 (obligatorio):** El tipo de mensaje.
Opciones:	1. **Parámetro 3 (obligatorio):** • I = INFORMATIVO: En el Archivo de Log se registran eventualmente mensajes informativos sobre el curso de ejecución del comando. Ej: Inicio de Ejecución • W = WARNING (de alerta): En el Archivo de Log se registran eventualmente mensajes de alerta sobre el curso de ejecución del comando. Ej: Archivo ya procesado • E = ERROR: En el Archivo de Log se registran SIEMPRE mensajes de error Ej: Archivo Inexistente. • SE = ERROR SEVERO: En el Archivo de Log se registran SIEMPRE mensajes severos de error Ej: Comando Inexistente.

Ejemplos de invocación:

```
user@Fiuba:~$ Glog $0 "Mensaje a guardar en el log" I
user@Fiuba:~$ cat bash.log
2010/05/10 11:21:30 <I> Mensaje a guardar en el log

user@Fiuba:~$ Glog $0 "Mensaje a guardar en el log Warning" W
user@Fiuba:~$ cat bash.log
2010/05/10 11:22:58 <W> Mensaje a guardar en el log Warning

user@Fiuba:~$ Glog $0 "Mensaje a guardar en el log Error" E
user@Fiuba:~$ cat bash.log
2010/05/10 11:24:09 <E> Mensaje a guardar en el log Error

user@Fiuba:~$ Glog $0 "Mensaje a guardar en el log Several Error" SE
user@Fiuba:~$ cat bash.log
2010/05/10 11:24:51 <SE> Mensaje a guardar en el log Several Error
```

Listado del código del comando:

```
#!/bin/bash

# Salidas:  0 - OK
```

```
#           1 - Error de parámetros
#           2 - Sistema no inicializado


if [ -z "$SISTEMA_INICIALIZADO" ]
then
    echo "Error, no esta inicializado el sistema."
    return 2
fi

Glog(){

    if [ $# -ne 3 ]
    then
       #Se requieren exactamente 3 parametros
       return 1
    fi

    local TIPO=""

    case $3 in
       [Ii]) TIPO="I";;
       [Ww]) TIPO="W";;
       [Ee]) TIPO="E";;
       [Ss][Ee]) TIPO="SE";;
       *) TIPO="I";;
    esac


    local DIRECTORIO="$grupo/comandos/log/"
    local NOMBRE_LOG="$DIRECTORIO/`basename $1`.log"

    mkdir -p "$DIRECTORIO"

    local FECHA=`getFechaYHora`

    local LINEA="$FECHA <$TIPO> $2"
    LINEA=`echo $LINEA | cut -c1-132` #limito la linea a 132 caracteres
    echo  $LINEA >> "$NOMBRE_LOG"

    return 0
}

export -f Glog
```

Comandos Auxiliares

- ***Nombre del comando: bloquearProceso***

Justificación de su uso:	Este comando se utiliza para permitir ejecutar un comando sólo a la vez y no permitir ejecutar el mismo comando en simultáneo, para ello el mismo genera un archivo de lock.
Archivos de Input:	Ninguno.
Archivos de Output:	Si es la primera vez que se lo ejecuta, es decir, si este comando no se encontraba ejecutándose en ese momento, genera el archivo de lock. Éste archivo conforma su nombre de la siguiente manera: ".lock_comandoqueloinvoca_corriendo" . Como puede observarse se trata de un archivo oculto.
Parámetros:	Nombre del comando que lo invoca.
Opciones:	Ninguna.

Ejemplo de invocación:	`bloquearProceso "$0"`
	`if [ $? -ne 0 ]`
	`then`
	`  echo Ya existe un proceso $0 corriendo, se termina la ejecucion.`
	`fi`
	`-Donde $0 es el comando que lo invoca.`

Listado del código:	`bloquearProceso(){` `  if [ $# -ne 1 ]` `  then` `  return 1` `  fi` `  mkdir -p "$grupo/locks"` `  local nombre=`basename "$1"` ` local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"` ` if [-e "$ARCHIVO_LOCK"]` ` then` ` return 2` ` fi`  ` # Creo un archivo lock oculto con el numero de pid del proceso` ` echo PID=$$ > "$ARCHIVO_LOCK"`

```
    return 0
}
```

- ***Nombre del comando: desbloquearProceso***

Justificación de su uso:	Este comando se utiliza para desbloquear un proceso bloqueado mediante el comando bloquearProceso. Para ello elimina el archivo de lock generado por éste último.
Archivos de Input:	Archivo de lock del proceso invocante (si existe).
Archivos de Output:	Ninguno.
Parámetros:	Nombre del comando que lo invoca.
Opciones:	Ninguna.

Ejemplo de invocación:	`# Elimino el archivo lock oculto`
	`desbloquearProceso "$0"`
	`-Donde $0 es el nombre del comando que lo invoca.`

Listado del código:	`desbloquearProceso(){` `  if [ $# -ne 1 ]` `  then` `  return 1` `  fi` `  local nombre=`basename "$1"` ` local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"`  ` rm -rf "$ARCHIVO_LOCK" > /dev/null` ` return $?` `}`

- ***Nombre del comando: estaCorriendo***

Justificación de su uso:	Éste comando se utiliza para verificar si el proceso ya se encuentra corriendo en ese momento. Se lo utiliza en conjunto con los comandos bloquearProceso y desbloquearProceso.
Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Nombre del comando que lo invoca.
Opciones:	Ninguna.
Ejemplo de invocación:	En perl: `/bin/bash -c "estaCorriendo invreci"`; Este comando chequea si esta corriendo el comando invreci en ese momento.

Listado del código:	estaCorriendo(){
	if [$# -ne 1]
	then
	return 1
	fi
	
	local nombre=`basename "$1"`
	local ARCHIVO_LOCK="$grupo/locks/.lock_${nombre}_corriendo"
	
	if [-e "$ARCHIVO_LOCK"]
	then
	return 0
	fi
	return 2
	}

- ***Nombre del comando: getFechayHora***

Justificación de su uso:	Se lo utiliza para generalizar la obtención de la fecha y hora que es necesaria para grabar los registros durante la ejecución de los comandos.
Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplo de invocación:	En perl:
	`my $fecha=`/bin/bash -c getFechaYHora`;`
	`print "fecha: $fecha\n";`

Listado del código:	`getFechaYHora(){`
	`  echo -n `date +"%Y/%m/%d %H:%M:%S"``
	`  return $?`
	`}`

- ***Nombre del comando: getUsuario***

Justificación de su uso:	Se utiliza para generalizar la obtención del usuario necesario para grabar los registros durante la ejecución de los comandos.
Archivos de Input:	Ninguno.
Archivos de Output:	Ninguno.
Parámetros:	Ninguno.
Opciones:	Ninguna.

Ejemplo de invocación:	`En perl:`
	`my $usuario=`/bin/bash -c getUsuario`;`
	`print "Usuario: $usuario\n";`

Listado del código:	`getUsuario(){`
	`  echo -n `id -un``
	`  return $?`
	`}`

Archivos Auxiliares

- ***Nombre del archivo: .lock_invonio_corriendo***

Estructura:	En su interior se guarda el PID del proceso invonio que se ejecutó.
Justificación de uso:	Es un archivo de lock que se utiliza para saber si ya se encuentra en ejecución el comando invonio en ese momento, para ello se valida la existencia de dicho archivo y al finalizar el proceso se lo elimina para permitir que se pueda correr nuevamente. En especial el archivo .lock_invonio_corriendo utiliza el PID para luego desde la función ./stopinvonio puede realizarse un SIGTERM sobre el proceso invonio que se inicializó anteriormente.

- ***Nombre del archivo: .lock_invreci_corriendo***

Estructura:	En su interior se guarda el PID del proceso invreci que se ejecutó.
Justificación de uso:	Es un archivo de lock que se utiliza para saber si ya se encuentra en ejecución el comando invreci en ese momento, para ello se valida la existencia de dicho archivo y al finalizar el proceso se lo elimina para permitir que se pueda correr nuevamente.

- ***Nombre del archivo: .lock_remioc_corriendo***

Estructura:	En su interior se guarda el PID del proceso remioc que se ejecutó.
Justificación de uso:	Es un archivo de lock que se utiliza para saber si ya se encuentra en ejecución el comando remioc en ese momento, para ello se valida la existencia de dicho archivo y al finalizar el proceso se lo elimina para permitir que se pueda correr nuevamente.

- ***Nombre del archivo: sobrante.sob***

Estructura:	**Formato de registros** **Campo** / **Descripción** **Número de orden de compra** — 6 caracteres **Código de producto a entregar** — 10 caracteres **Cantidad sobrante** — Numérico, mayor que cero **Usuario de grabación** — N caracteres. Usuario que graba el registro **Fecha y hora de grabación** — N caracteres. Fecha y hora de grabación de registro
Justificación de uso:	Se utiliza para resguardar los registros que no tienen un lugar donde registrarse para de este modo evitar perder la información correspondiente a los mismos. Cuando sobran productos, se guardan los datos en un archivo /$grupo/sobrantes/sobrante.sob/.

Apéndice A

Tema	Grupo	Ayudante		Evaluación Grupal del TP
			Nota	
			Fecha	
			Firma	

Integrantes					
Padrón	**Apellido**	**Nombre**	**Asistencia a Entrega**	**Asistencia a Revisión**	**Evaluación Individual Final**

Evaluación Grupal de la Entrega

	Aprobada	Se Debe Rehacer	No Entrega	Observaciones
Presentación				
Hipótesis Planteadas				
Diagrama de Procesos				
Comandos: invini				
invonio				
invreci				
remioc				
occtrl				
glog, mover, startinvonio, stopinvonio				
Comandos Auxiliares				
Archivos Auxiliares				
Problemas Relevantes				
Hoja de Ruta				
Funcionamiento General				

Condiciones de Resolución y Corrección

Introducción

1. Se deberá tener en cuenta para la resolución TODAS las condiciones que se enuncian.
2. Se deben respetar los formatos de archivos especificados
3. Se debe respetar la estructura de directorios planteada
4. Cada comando deberá ser resuelto utilizando el lenguaje indicado.
5. El día de vencimiento del TP, cada ayudante convocará a los integrantes de un grupo e iniciará la corrección mediante una entrevista grupal. Es imprescindible la presencia de todos los integrantes del grupo el día de la corrección. El objetivo de esto es comprender la dinámica de trabajo del equipo y los roles desempeñados por cada uno.
6. Posterior a la entrega se podrá acordar entre el Ayudante y el Grupo el intercambio de correspondencia a través de la cuenta so7508@gmail.com. Por este medio el ayudante podrá solicitar a los alumnos correcciones, mejoras, nuevos datos, etc. Cabe aclarar que todos los mensajes a esta casilla deberán tener como asunto "Grupoxx (xx es el número asignado al grupo) para poder redireccionar el mail al ayudante correspondiente.
7. Dentro de los ítems a chequear el ayudante evaluará **aspectos formales** (como ser la forma de presentación de la carpeta), **aspectos funcionales** (como ser que el TP funcione) y **aspectos operativos**: que el TP resuelva el problema planteado.

Documentación a Presentar

Se deberá hacer entrega de:

- Una Carpeta de Trabajo Práctico con el contenido solicitado mas adelante. En el pie de cada hoja se debe incluir: Número de Grupo y Tema (en el margen izquierdo) y Número de Hoja (en el margen derecho). TODAS las hojas deberán estar numeradas y enganchadas a una carpeta. Las hojas sueltas no se considerarán como parte de la misma.
- Un archivo con el formato y contenido solicitado mas adelante

Contenido de la Carpeta

1. **Carátula**: La entregada en este mismo documento con los datos completos en 2 COPIAS.
2. **Índice del Contenido de la Carpeta**.
3. **Hipótesis y Aclaraciones:** todas las hipótesis que han sido consideradas en la resolución del TP y cualquier otra aclaración que se considere necesaria.
4. **Problemas relevantes:** Enumeración de los problemas relevantes que se les hayan presentado durante el desarrollo y prueba de los comandos y como los solucionaron.
5. **Archivo README: Instructivo de Instalación**
6. **Diagrama de Procesos:** Diagrama de proceso a alto nivel indicando los comandos las entradas y las salidas.
7. **Hoja de Ruta para la Corrección:** Se deberá indicar claramente la secuencia de pasos que se deben dar para realizar una prueba integral de todos y cada uno de los comandos solicitados. Indicar claramente cuales son las precondiciones que se deben cumplir para el correcto funcionamiento del comando (que variables deben estar definidas, como definirlas y con que valores, que archivos deben borrarse, moverse, o crear, etc.), Cuales son los archivos de input y cuales los de output.
8. **Comandos Solicitados**

   a. **Nombre del Comando**
   b. **Archivos de Input**
   c. **Archivos de Output**
   d. **Parámetros,** si corresponde
   e. **Opciones**, si corresponde
   f. **Ejemplos de invocación** con distintos parámetros y valores, indicando en que oportunidad es usado y cuál es el resultado previsto.
   g. **Listado del código** del comando: éste deberá incluir comentarios útiles para la comprensión del código.

9. **Comandos Auxiliares**
   a. **Nombre del Comando**
   b. **Justificación de su uso**
   c. **Archivos de Input**
   d. **Archivos de Output**
   e. **Parámetros,** si corresponde
   f. **Opciones**, si corresponde
   g. **Ejemplos de invocación** con distintos parámetros y valores, indicando en que oportunidad es usado y cuál es el resultado previsto.
   h. **Listado del código** del comando: éste deberá incluir comentarios útiles para la comprensión del código.
10. **Archivos Auxiliares**
   a. **Archivos auxiliares:** Por cada archivo auxiliar que se utilice indicar:
      i. Nombre del Archivo y Estructura
      ii. Justificación de su uso
11. **Apéndice A**
   a. El presente documento.

Formato y contenido del Archivo

El archivo de Trabajo Práctico deberá ser enviado a la cuenta so7508@gmail.com dentro de las 24hs posteriores a la corrección. Deberá ser un único archivo instalable en formato ".tgz" con todos los archivos y directorios empaquetados en un archivo "tar" y luego comprimido con "gzip".

MUY IMPORTANTE: cuando se efectúa el "tar" se debe usar la opción de "directorio relativo" para evitar problemas cuando se proceda a la instalación en la facultad para su corrección.

Observaciones a tener en cuenta en la instalación

1. Generar un script para la instalación
2. Los usuarios disponibles son solo tres: prueba01(pw test01), prueba02 (pw test02), prueba03 (pw test03). Debido a esto cada grupo deberá crear un subdirectorio llamado grupoxx (donde xx es el nro de grupo)
3. El instalable deberá contener:
   3.1. Los scripts desarrollados (según se detallan mas adelante)
   3.2. Juego de datos de ejemplo y de prueba lo suficientemente variado como para contemplar todas las variantes posibles. Los archivos de prueba deben respetar las estructuras y directorios indicados en el enunciado ya que es uso y costumbre intercambiar datos entre los distintos grupos para enriquecer los

testeos.

3.3. Un archivo README con las instrucciones de instalación y cualquier otra indicación u observación que se quiera incorporar para que el ayudante pueda realizar los testeos. A modo de ejemplo podemos enumerar los siguientes pasos:

- Insertar el dispositivo de almacenamiento con el contenido del tp (pen drive, cd, etc)
- Crear en el directorio corriente un subdirectorio grupoxx
- Copiar el archivo *.tgz en ese directorio
- Descomprimir el *.tgz de manera de generar un *.tar
- Extraer los archivos del tar.
- …

3.4. Cualquier otro archivo que el grupo considere necesario (ej: resultado de las pruebas por Uds. Efectuadas, logs, etc.)

4. Los comandos deben almacenarse en $grupo/comandos

4.1. Los datos de prueba deben almacenarse en $grupo/prueba

5. Los archivos de Ordenes de Compra deben almacenarse en $grupo/oc

6. Todo el camino (path) desde la raiz hasta el directorio de trabajo por Uds elegido lo llamaremos “$grupo” para simplificar el enunciado.

Observaciones a tener en cuenta en el desarrollo

1. Código de Retorno:

1.1. Toda invocación desde un comando a otro debe devolver un código de retorno cero (0) si fue exitoso o distinto de cero si tuvo errores. Siempre al finalizar el comando se debe indicar si finalizó correctamente o con errores.

2. Pasos Sugeridos:

2.1. En la descripción de los comandos se sugieren los pasos de ejecución. Estos deben considerarse indicativos, ya que el programador puede alterar este orden según su conveniencia siempre que no afecte el resultado final esperado. Se dan a los efectos de ordenar la explicación. Si Ud. considera conveniente modificar algún paso, tanto sea en el orden de ejecución o en la forma de resolverlo, no hay inconveniente, siempre que lo aclare debidamente en las hipótesis.

3. Errores y Mensajes:

3.1. Cuando se describen los comandos se hace referencia a errores y mensajes informativos, estas referencias son sólo una guía para el desarrollador, pueden existir errores no descriptos en el enunciado que deban ser manejados por el programador.

3.2. Siempre se debe explicar el error con un mensaje amigable.

3.3. Es conveniente para el seguimiento y corrección de los comandos que a medida que se avanza en la ejecución del mismo se vayan mostrando por consola mensajes descriptivos indicando el punto de avance y resultados intermedios.

4. Archivos de Log, Función de escritura de log

4.1. Todos los comandos indican si deben o no un archivo de log llamado: $grupo/comandos/log/<nombre del comando>.log

4.2. El archivo de log no existe, se debe crear. Si existe se le deben agregar los registros

4.3. En el Log se debe registrar toda la interacción entre el comando y el operador: ej: todos los mensajes que se muestran por la consola del operador (3.3) y los mensajes de los errores que se produzcan (3.2), las

opciones o parámetros que ingresa el operador.

4.4. La escritura en el archivo de Log debe ser homogénea para todos los comandos que generan registros de log, por lo tanto debe estar centralizada a través de una función **Glog.**

4.5. Esta función **Glog** debe contemplar el uso de los siguientes parámetros: nombre del comando y mensaje a grabar

4.6. Debe permitir ser invocada desde la línea de comando

4.7. Cuando graba una linea de log siempre se debe indicar: fecha y hora, tipo de mensaje y mensaje.

4.8. Existen cuatro tipos de mensajes:

- I = INFORMATIVO: En el Archivo de Log se registran eventualmente mensajes informativos sobre el curso de ejecución del comando. Ej: Inicio de Ejecución
- W = WARNING (de alerta): En el Archivo de Log se registran eventualmente mensajes de alerta sobre el curso de ejecución del comando. Ej: Archivo ya procesado
- E = ERROR: En el Archivo de Log se registran SIEMPRE mensajes de error Ej: Archivo Inexistente.
- SE = ERROR SEVERO: En el Archivo de Log se registran SIEMPRE mensajes severos de error Ej: Comando Inexistente.

5. Movimiento de Archivos:

5.1. En líneas generales no se borra ningún archivo, excepto los archivos temporales por Uds, definidos. Debido a esto muchos comandos mueven archivos de un directorio a otro.

5.2. El movimiento de archivos debe ser homogéneo para todos los comandos, por lo tanto debe estar centralizado a través del comando **Mover**

Enunciado

Una empresa necesita automatizar el control de cumplimiento de las órdenes de compra. Para ello instrumenta un proceso de cinco pasos a saber:

Primer Paso

El proceso se inicia con la Orden de Compra. Los datos de la orden de compra se almacenan en dos archivos: El archivo global de órdenes de compra y el archivo de detalle de Órdenes de compra.

Ambos archivos están disponibles para su lectura y eventual actualización.

A los efectos exclusivos de este tp solo validaremos la existencia de los archivos de órdenes de compra, no desarrollaremos su carga. Esta actividad debe estar reflejada en el comando ***invini.***

Segundo Paso

Luego tenemos el proceso de recepción remitos. Cada proveedor nos hace llegar en forma anticipada información de la entrega a través de los remitos. Para ello nos envía un archivo por cada remito. Contar en forma anticipada con esta información permite organizar la descarga de la mercadería.

A los efectos de este tp organizaremos la recepción en dos actividades:

La primera de ellas será detectar el arribo de los archivos a un directorio específico y la verificación de su nombre a través de un demonio. Esta actividad debe estar reflejada en el comando ***invonio.***

La segunda de ellas será la validación interna de cada archivo. Esta actividad debe estar reflejada en el comando

invreci.

Tercer Paso

No se recibe mercadería que no haya sido informada previamente, por lo tanto en el proceso de entrega/recepción de la mercadería se debe controlar esta situación.

Esta etapa del proceso es manual, por lo tanto a los efectos exclusivos de este tp pasaremos al siguiente paso asumiendo que todas las mercaderías entregadas se corresponden con los remitos aceptados.

Cuarto Paso

El proceso de conciliación de Orden de compra es un proceso que se alimenta de los remitos aceptados, verifica si la orden de compra ya esta completa y actualiza su estado.

Quinto Paso

Finalmente tenemos el proceso de control que permite ir viendo el grado de cumplimiento de las órdenes de compra.

Se pide:
Desarrollar los siguientes comandos:

1. Desarrollar un comando de Inicialización de Ambiente (**Invini**), en shell script
2. Desarrollar un comando de Detección de Arribo de Archivos (**Invonio**), en shell script
3. Desarrollar un comando de Validación de Remitos (**Invreci**), en shell script
4. Desarrollar un comando de Conciliación de Orden de Compra (**Remioc**), en shell script o en perl
5. Desarrollar un comando de Control (**Occtrl**), en perl
6. Desarrollar las funciones **glog, mover, startinvonio, stopinvonio**

Comandos a Desarrollar

Comando de Inicialización de Ambiente

Nombre del Comando

invini

Descripción

El propósito de este comando es preparar el entorno de ejecución del TP (ambiente). Es el primero en orden de ejecución y no graba en el archivo de Log. Debe efectuar el seteo inicial de las variables de ambiente, incluyendo la variable PATH, para permitir la correcta ejecución de los scripts por parte de usuarios que no son root.

En los comandos desarrollados, cuando se hace referencia a un archivo, se deberá usar la variable de ambiente $grupo, la cual toma valor en este script de inicialización. Su valor es = a todo el camino (path) desde la raiz hasta el directorio de trabajo por Uds. elegido.

Esto se pide para evitar el hardcode de directorios dentro de los comandos, dentro de ellos no debe escribirse /usr/prueba01/grupo22/arribos sino $grupo/arribos, por ejemplo.

Invini debe setear las variables de ambiente una sola vez por cada sesión de usuario. Si el ambiente ya fue inicializado, entonces mostrar un mensaje de advertencia con el contenido de las variables seteadas.

Invini debe verificar si la instalación está completa, si detecta algún problema en la instalación, debe explicar la situación y terminar su ejecución. Este control debe incluir la verificación de los archivos indispensables para ejecutar el sistema, como ser los archivos de órdenes de compra.

Luego del seteo de las variables de ambiente y de la verificación de las condiciones óptimas para la ejecución, se debe invocar al script invonio siempre que invonio no se esté ejecutando (verificar con ps).

- Si ejecuta correctamente debe mostrar por pantalla el siguiente mensaje:
  Inicialización de Ambiente Concluida
  Ambiente <mostrar contenido de las variables de ambiente seteadas>
  Demonio corriendo bajo el no.: <Process Id de **invonio>**
- Si da algún tipo de error
  Inicialización de Ambiente No fue exitosa. Error en <descripción del error>

Comando de Detección de Arribo de Archivos

Nombre del Comando

invonio

Descripción

El propósito de este comando es detectar la llegada de archivos al directorio $grupo/arribos. Es el segundo en orden de ejecución y no graba en el archivo de Log. Debe efectuar la validación del nombre del archivo y ponerlo a disposición del siguiente paso. Si el archivo no es válido, debe rechazarlos.

Este comando forma parte del proceso de recepción de remitos.

Los archivos válidos poseen nombres con este formato: <remito>.<fecha>

- Para verificar el remito usar los valores 00000000 a 99999999
- Para verificar la fecha, además de ser válida debe ser menor o igual a la fecha del día.

Este comando es un proceso del tipo "Demonio"

- Un demonio es un proceso que corre en forma permanente, en background.
- Suele armarse como un conjunto de sentencias dentro de un ciclo eterno.
- Se utiliza generalmente para:
  - monitorear otros procesos
  - ejecutar procesos rutinarios

Pasos sugeridos

1. Chequear la existencia de archivos en el directorio $grupo/arribos
2. Si existen archivos, por cada archivo que se detecta
   2.1. Verificar si el nombre del archivo es valido
   2.1.1. Si es válido **mover** el archivo a $grupo/recibidos empleando la función mover indicada mas abajo
   2.1.2. Si no es válido **mover** el archivo a $grupo/rechazados empleando la función mover indicada mas abajo
3. Chequear la existencia de archivos en el directorio $grupo/recibidos (ya sean del ciclo actual o de ciclos anteriores)
   3.1. Si existen archivos en $grupo/recibidos
   3.1.1. Invocar al Comando **invreci**
   Se debe invocar al comando **invreci** siempre que haya algún archivo en $grupo/ recibidos e **invreci** no se esté ejecutando.
   Si arranca correctamente se debe mostrar por pantalla el process id de **invreci**
   Si da algún tipo de error se debe mostrar por pantalla el mensaje explicativo
4. Dormir un tiempo x
5. Volver al punto 1

Queda a consideración de cada grupo el valor que se asigna a la variable de tiempo x, para la prueba del TP se solicitará modificar el valor usando vi.

Se deben generar dos comandos satélites llamados **stopinvonio** (para detener la ejecución del demonio) y

startinvonio (para iniciar la ejecución del demonio). No se puede arrancar el demonio si la inicialización de ambiente no fue realizada o si ya existe otro demonio corriendo.

Comando de Validación de Remitos

Nombre del Comando

invreci

Descripción

El propósito de este comando es validar los archivos de remito. Es el tercero en orden de ejecución y graba en el archivo de Log. Debe efectuar la validación del remito contra la orden compra. Si el remito es valido, debe formatearlo y dejarlo disponible para que el comando remioc lo pueda utilizar. Si el remito no es válido, debe rechazarlo.

Este comando forma parte del proceso de recepción de remitos.

Pasos sugeridos

Ver si existe otro invreci corriendo

Si se detecta otro en ejecución mostrar un mensaje explicativo y terminar la ejecución de éste.

Ver si se inicializó el ambiente

No se puede ejecutar este comando si la inicialización de ambiente no fue realizada. Si detecta algún problema mostrar un mensaje explicativo y terminar la ejecución.

Inicializar el Log

Grabar en log

- Inicio de Invreci: <cantidad de archivos a procesar>

Un Archivo

Los archivos de input se encuentran en $grupo/recibidos
Grabar en log

- Archivo a Procesar: <nombre del archivo>

Verificar que no sea un remito duplicado

Es posible detectar tempranamente si el remito vino duplicado analizando el directorio $grupo/yarecibidos. Si existe en este directorio un archivo de igual nombre, entonces se lo considerará duplicado. En este caso, mover el archivo a $grupo/rechazados empleando la función mover indicada mas abajo.
Grabar en log

- Remito Duplicado: <nombre del archivo>

Seguir en el paso 5.

Un Registro

Validar un Registro

Si en el registro se indica una orden de compra inexistente, rechazar el registro
Si en el registro se indica una orden de compra cerrada, rechazar el registro

Ambas validaciones se hacen contra el archivo global de órdenes de compra.
Si el registro no posee el formato adecuado, rechazar el registro.
Es fundamental validar que venga informado el código de producto a entregar.
También se debe validar que el CUIT del proveedor sea el mismo que el informado en la orden de compra global.
Pueden incluir otras validaciones que consideren adecuadas.

Si las validaciones dan ok, entonces el registro es aceptado, sino el registro es rechazado

Rechazar un Registro

Para rechazar un registro se lo debe grabar tal cual se lo recibió en un archivo de igual nombre pero con extensión rech en el directorio $grupo/rechazados.

Nota: el grupo deberá construir un conjunto de archivos con datos de prueba lo suficientemente variado como para contemplar todos los motivos de rechazo indicados. Documenten claramente que registro responde a que validación para facilitar el seguimiento y ahorrar tiempo en la corrección.

Aceptar un Registro

Para aceptar un registro se lo debe grabar con la estructura indicada mas abajo en un archivo de nombre <nro de remito>.<nro de orden de compra>.aproc en el directorio $grupo/aceptados.
(si en un remito vienen productos de distintas ordenes de compra, entonces un archivo de remito original puede generar mas de un archivo de remito-oc)

Campo	Descripción
Fecha de Remito	**8 Caracteres**
Código de Producto a Entregar	**10 Caracteres**
Cantidad a Entregar	**Numérico, mayor que cero**
Tiempo de Descarga en minutos	**Numérico, mayor que cero**
Volumen en m3	**Numérico, mayor que cero, con decimales**
Fecha de Entrega	**8 Caracteres, formato aaaammdd**
Lugar de Entrega	**11 Caracteres**
Transportista	**11 Caracteres**
CUIT del Proveedor	**11 Caracteres**
Compañía Aseguradora	**11 Caracteres**

Fin de Archivo

Si el archivo no vino duplicado, moverlo al directorio $grupo/yarecibidos empleando la función mover indicada mas abajo y grabar en el archivo de log:

- Remito Aceptado: <nombre del archivo>
- Cantidad de Registros de Leídos:
- Cantidad de Registros Aceptados:
- Cantidad de Registros Rechazados por Orden de Compra Cerrada
- Cantidad de Registros Rechazados por Orden de Compra Inexistente

- Cantidad de Registros Rechazados por Otros Motivos

Repetir el paso 4 hasta que se terminen todos los archivos.

Cerrar el Log

Grabar en log

- Fin de Invreci

Comando de conciliación de orden de compra

Nombre del Comando

remioc

Descripción

El propósito de este comando es actualizar el estado de la orden de compra. Es el cuarto en orden de ejecución y graba en el archivo de Log. Se invoca manualmente indicando:

- opción a) que orden / órdenes de compra se quieren conciliar contra el universo de los remitos aceptados a procesar
- opción b) que remito / remitos aceptados se quieren conciliar

Este comando forma parte del proceso de conciliación de Orden de compra

Pasos sugeridos

Ver si existe otro remioc corriendo

Si se detecta otro en ejecución mostrar un mensaje explicativo y terminar la ejecución de éste.
Ver si se inicializó el ambiente

No se puede ejecutar este comando si la inicialización de ambiente no fue realizada. Si detecta algún problema mostrar un mensaje explicativo y terminar la ejecución.
Ver si invreci está corriendo

Si se detecta que invreci está corriendo mostrar un mensaje explicativo y terminar la ejecución de remioc
Inicializar el Log

Grabar en log

- Inicio de remioc: <parámetros de invocación>

Si la opción es por orden de compra

Se debe verificar que la orden de compra pasada como parámetro sea una orden de compra existente y no esté cerrada. Ambas validaciones se hacen contra el archivo global de órdenes de compra.
Si se detecta algún inconveniente, mostrar y grabar en el log un mensaje explicativo y seguir con la siguiente orden de compra.
Sino Grabar en log

- Conciliación de la orden de compra <nro de oc>

Selección del remito

Encontrar todos los archivos de remitos aceptados a procesar que se relacionan con la orden de compra y listarlos para que el usuario indique cuales desea procesar.
Puede elegir todos, por rango o por número especifico.
Implementar esto con un menú amigable y mensajes de ayuda.
Pedir confirmación antes seguir adelante.
Si no confirma, grabar en el log un mensaje explicativo y seguir con la siguiente orden de compra.

Grabar en log

- El listado de remitos disponibles y el listado de remitos seleccionados para procesar.

Proceso

Para cada ítem de detalle de orden de compra en estado Abierto, calcular su remanente (Cantidad remanente nueva = cantidad remanente – cantidad a entregar)
Para ello aplicar los registros de remitos aceptados que coincidan con él. Hay coincidencia entre ambos archivos cuando:

Registro Ocdet	Registro de remito
Ocdet.Número de Orden de Compra	= Remito.Número de Orden de Compra
Ocdet.Código de Producto	= Remito.Código de Producto a Entregar
Ocdet.Estado del Ítem	= ABIERTO

Al comparar ambos registros puede ocurrir que:

a) cantidad remanente = cantidad a entregar

b) cantidad remanente < cantidad a entregar

c) cantidad remanente > cantidad a entregar

En los casos a y b, la cantidad remanente queda en cero y el estado del ítem pasa a cerrado.

El caso b) nos dice que se entregaron mas unidades que las remanentes. Si ocurre esto, se debe buscar otro ítem de detalle de oc coincidente para aplicar las unidades sobrantes del remito. Si aún asi no se logran aplicar todas las unidades entregadas, se debe generar un mensaje de advertencia y grabar en el log la descripción de esta situación con todo el detalle necesario para poder efectuar un seguimiento del caso.

El caso c) nos dice que aún falta efectuar entregas para ese ítem de detalle, por lo tanto la cantidad remanente será la cantidad remanente nueva. Esta es una situación habitual del proceso.

Fin del Remito

Luego del procesamiento del archivo de remito, cambiar el nombre del archivo de extensión aproc a extensión proc.

Seguir con el siguiente remito

Repetir el proceso 5.2 para todos los archivos de remito. Cuando no hay más archivos para procesar seguir en el siguiente paso

Fin del Proceso

Actualizar ocdet

Al finalizar el proceso de apareo, se debe generar una nueva versión del archivo de detalle de orden de compra actualizado.

El archivo de detalle de órdenes de compra ocdet posee un número de versión a modo de extensión de archivo. Este número debe generarse a partir de la secuencia 1 en adelante.

Se debe generar una nueva versión del archivo solo si se modifica algún registro.

El archivo original no debe ser removido, permanece a modo de histórico.

Campo	**Descripción**
Número de Orden de Compra	**Ídem registro original**

Nro. de Ítem	**Ídem registro original**
Código de Producto	**Ídem registro original**
Cantidad Total	**Ídem registro original**
Cantidad Remanente	**Según resultado del proceso**
Estado del Ítem	**ABIERTO para los registros con cantidad remanente > 0, CERRADO para los registros con cantidad remanente = 0**
Usuario de grabación	**Según corresponda**
Fecha y Hora de grabación	**Según corresponda**

Grabar en log

El nombre del nuevo archivo de detalle.

Actualizar ocgob

Si todos los registros de detalle de la orden de compra quedaron en estado CERRADO, entonces estamos en condiciones de cerrar también la orden de compra

Para ello se debe generar una nueva versión del archivo global de orden de compra.

El archivo global de orden de compra ocgob posee un número de versión a modo de extensión de archivo. Este número debe generarse a partir de la secuencia 1 en adelante.

Se debe generar una nueva versión del archivo solo si se modifica algún registro.

El archivo original no debe ser removido, permanece a modo de histórico.

Si la opción es por remito

Se debe verificar que el/los remitos pasados como parámetro tengan su archivo aproc correspondiente.
Si no hay archivos a procesar, mostrar y grabar en el log un mensaje explicativo y permitir volver a ingresar parámetros

Selección de orden de compra

Listar remito-oc a procesar para que el usuario indique cuales desea.
Puede elegir todos o por número especifico.
Implementar esto con un menú amigable y mensajes de ayuda.
Pedir confirmación antes seguir adelante.
Grabar en log la selección efectuada.

Proceso

Igual al descripto en 5.2

Repetir el proceso hasta terminar con los archivos seleccionados

Fin del Proceso

Igual al descripto en 5.4

Cerrar el Log

Grabar en log

- Fin de remioc

Comando de Control

Nombre del Comando

occtrl

Descripción

El propósito de este comando es efectuar el control del grado de cumplimiento de las órdenes de compra. No Graba en el archivo de Log.
Se invoca manualmente indicando:

- que ordenes de compra se quieren controlar (todas, por rango, especifica)
- que formato de salida se desea (pantalla, archivo, ambos)

Este comando forma parte del proceso de Control

Pasos sugeridos

Ver si remioc está corriendo

Si se detecta que remioc está corriendo mostrar un mensaje explicativo y terminar la ejecución de éste.
Ver si se inicializó el ambiente

No se puede ejecutar este comando si la inicialización de ambiente no fue realizada. Si detecta algún problema mostrar un mensaje explicativo y terminar la ejecución.
Para cada orden de compra que se quiere controlar

Si la orden de compra esta cerrada listar:

Orden de Compra Numero	Número de Orden de Compra
Grado de Cumplimiento	100%
Fecha de OC	Fecha de Orden de Compra
Fecha de Cierre	Fecha y Hora de grabación
Proveedor	CUIT del Proveedor

Si la orden de compra esta abierta, antes de listar se debe calcular el grado de cumplimiento.
Obtener la suma A = Sumatoria de todas las cantidades totales de los ítems de detalle de la oc
Obtener la suma B = Sumatoria de todas las cantidades remanentes de los ítems de detalle de la oc
Obtener el grado de cumplimiento = B/A expresándolo en porcentaje con hasta dos decimales.
Obtener el pendiente = A-B
Ejemplo: A=1000 B=100 => Grado = 10% Pendiente = 900

Orden de Compra Numero	Número de Orden de Compra
Grado de Cumplimiento	Según cálculo
Fecha de OC	Fecha de Orden de Compra
Pendiente	Según cálculo
Proveedor	CUIT del Proveedor

Si se indico salida por pantalla: Mostar el listado de forma ordenada y clara para su rápida interpretación
Si se indico salida por archivo: Grabar un archivo de salida en el directorio $grupo/oc sin borrar ni pisar ninguna salida previa.
Seguir con la siguiente oc

Repetir el paso 3

Nombre de la función: Mover

Input

- Archivo definido en el parámetro 1

Output

- Archivo definido en el parámetro 2
- Archivo de Log del comando que la invoca (si corresponde)

Opciones y Parámetros

- Parámetro 1 (obligatorio): origen
- Parámetro 2 (obligatorio): destino
- Parámetro 3 (opcional): comando que la invoca
- Si son necesarios mas, especificar por el desarrollador

Descripción

Esta función tiene por objeto mover un archivo de un directorio a otro contemplando la posibilidad de archivos duplicados.

Puede ser invocada desde la línea de comando o bien desde otro comando.

En caso de archivos duplicados se debe gestionar un numero de secuencia nnn.

Este número de secuencia puede ser centralizado (una única secuencia para todo el sistema) o descentralizado (diferentes secuencias).

Pasos Sugeridos

1. Antes de efectuar el movimiento, debe:
   1.1. Verificar si el origen y el destino son iguales, no hacer nada
   1.2. Verificar si es un archivo duplicado, es decir, si en el destino ya existe un archivo con ese mismo nombre. En este caso debe:
      1.2.1. Ver si existe dentro del directorio destino un subdirectorio /dup
         1.2.1.1. si no existe, crearlo
         1.2.1.2. Mover el archivo a ese subdirectorio con el siguiente nombre: <nombre del archivo original>.nnn dónde nnn es un número de secuencial que evita nombres duplicados.
   1.3. si no es un archivo duplicado, entonces hacer el move al destino indicado.
2. Si esta función es invocada por un comando que graba en el archivo de log, actualizarlo indicando el resultado de la operación
   2.1. Ejemplo: Si el move no tuvo complicaciones Grabar en el Log:
   <fecha y hora>-I-Mover exitoso desde: <parámetro 1> hacia <parámetro 2>
3. Debe devolver un código de retorno cero (0) si fue exitoso el movimiento o distinto de cero si tuvo errores.

Directorios y Estructuras

Directorios

Archivo	Directorio
Arribos	**$grupo/arribos**
Ordenes de Compra Global	**$grupo/oc/ocgob.nn**
Ordenes de Compra Detalle	**$grupo/oc/ocdet.nn**
Remitos Recibidos	**$grupo/recibidos/<remito>.<fecha>**
Remitos Rechazados	**$grupo/rechazados/<remito>.<fecha>**
Remitos Ya Recibidos	**$grupo/yarecibidos/<remito>.<fecha>**
Registros Rechazados	**$grupo/rechazados/<remito>.<fecha>.rech**
Remitos Aceptados	**$grupo/aceptados/<remito>.<oc>.aproc**
Remitos Procesados	**$grupo/aceptados/<remito>.<oc>.proc**
Comandos	**$grupo/comandos**
Archivos de Log	**$grupo/comandos/log/<comando>.log**

Algunas Estructuras de Archivos

Entre campo y campo el separador es el carácter ; (punto y coma).
En los campos indicados con N Caracteres, su longitud es indeterminada

Archivos de Log

Son archivos de texto donde cada registro es una línea del reporte. La longitud de la línea no debe superar los 132 caracteres.

Campo	Descripción
Texto	**Caracteres (como máximo 132)**

Archivo Global de Ordenes de compra ocgob

Campo	Descripción
Número de Orden de Compra	**6 Caracteres**
Fecha de Orden de Compra	**8 Caracteres, formato aaaammdd**
CUIT del Proveedor	**11 Caracteres**
Estado de la Orden de Compra	**Valores Posibles: ABIERTA-CERRADA**
Usuario de grabación	**N caracteres, usuario que graba el registro.**
Fecha y Hora de grabación	**N caracteres, fecha y hora de grabación del registro.**

Archivo de Detalle de Ordenes de compra ocdet

Campo	Descripción

Número de Orden de Compra	**6 Caracteres**
Nro. de Ítem	**N Caracteres, Nro de ítem dentro de la orden de compra**
Código de Producto	**10 Caracteres**
Cantidad Total	**Numérico, mayor que cero**
Cantidad Remanente	**Numérico, cualquier valor**
Estado del Ítem	**Valores Posibles: ABIERTO-CERRADO**
Usuario de grabación	**N caracteres, usuario que graba el registro.**
Fecha y Hora de grabación	**N caracteres, fecha y hora de grabación del registro.**

Archivo de Remitos <no. de remito>.<fecha de remito>

Campo	Descripción
Número de Orden de Compra	**6 Caracteres**
Código de Producto a Entregar	**10 Caracteres**
Cantidad a Entregar	**Numérico, mayor que cero**
Tiempo de Descarga en minutos	**Numérico, mayor que cero**
Volumen en m3	**Numérico, mayor que cero, con decimales**
Fecha de Entrega	**8 Caracteres, formato aaaammdd**
Lugar de Entrega	**11 Caracteres**
Transportista	**11 Caracteres**
CUIT del Proveedor	**11 Caracteres**
Compañía Aseguradora	**11 Caracteres**