Programación de Scripts en IRAF (II)

Marzo de 2006

Ricardo Gil-Hutton

Más allá de los comandos usuales que se utilizan en un script para hacer tareas sencillas se pueden resolver situaciones y problemas más complejos utilizando otros comandos y técnicas. A continuación se describirán algunas de estas situaciones y se propondrá una posible manera de solución.

Recordemos primero la estructura básica de un script en IRAF:

```
procedure xxxxx (arg1, arg2, arg3, ...)

declaracion de parametros posicionales
declaracion de parametros ocultos
declaracion de punteros

begin
  declaracion de variables locales

  comandos
  ...
  ...
end
```

donde:

- *argN* son los parámetros posicionales requeridos.
- un *puntero a lista* **DEBE** ser declarado como parámetro, pero no como argumento.
- **TODAS** las variables locales deben declararse en el script.
- los paquetes a utilizar por el script **DEBEN** estar cargados previamente.

- toda línea del script que se inicie con '#' se considera un comentario.

Es usual guardar el script en un archivo con el mismo nombre del procedimiento y con extensión '*.cl* '. Para ejecutarlo se lo debe cargar primero con la tarea *task*.

1. **¿Cómo controlar si un paquete o tarea está cargada?**

   Una de las condiciones para ejecutar un script es tener todos los paquetes o tareas necesarias cargadas antes de su ejecución. Lamentablemente, no es posible cargar un paquete desde dentro de un script por lo cual el único recurso que queda es efectuar una verificación y en caso de ser necesario salir con un mensaje de error. Para eso se utilizan las tareas *DEFPAC* o *DEFTASK* que devuelven *yes* o *no* dependiendo si el paquete o tarea se encuentra disponible o no, respectivamente. Por ejemplo, si se quiere verificar que el paquete *IMCOORDS* está cargado se puede utilizar:

```
  procedure xxxxx (arg1, arg2, arg3, ...)

  declaracion de parametros posicionales
  declaracion de parametros ocultos
  declaracion de punteros

  begin
    declaracion de variables locales

# verifica que esten cargados los paquetes necesarios
#
    if(!defpac('imcoords')){
      print("El paquete IMCOORDS debe estar cargado")
      bye
    }
    ...
    ...
  end
```

2. **¿Cómo comprobar si un archivo ya existe?**

Muchas veces es necesario sacar información desde alguna tarea de IRAF a un archivo y es importante verificar si ese archivo ya existe dado que se corre el peligro de sobrescribirlo. Para eso se puede utilizar la tarea *ACCESS*:

```
  procedure xxxxx (...,arch1,...)
  ...
  string arch1    {prompt="archivo de salida"}

  begin

# si el archivo ya existe saca un mensaje de error
#
    if(access(arch1)){
      print("El archivo "//arch1//" ya existe")
      bye
    }
    ...
    ...
  end
```

3. **¿Cómo utilizar la información que la tarea saca a pantalla?**

   Muchas tareas sacan a pantala información que puede ser o no útil para el script. Esa información puede re-dirigirse hacia un archivo para guardarla y usarla posteriormente. Por ejemplo, un ejemplo clásico es la tarea *IMSTATISTICS* que muestra en pantalla parámetros estadísticos de una región de la imagen cuya salida a pantalla es usualmente algo parecido a:

```
ecl> imstat gal
# IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
    gal    262144     108.3     131.3        -1.    19936.
ecl>
```

   Entonces, suponiendo que necesitamos en el script el valor medio y la desviación standard de una cierta imagen, podemos guardar estos datos en un archivo para utilizarlos después mediante el siguiente procedimiento:

```
 procedure xxxxx (imagen, ...)

# declaracion de parametros posicionales y ocultos
#
 string imagen   {prompt="Imagen a procesar"}
 ...

# declaracion de punteros
#
 struct *arch

 begin

# declaracion de variables locales
#
  string imin,tfile
  int dum
  real vmed,sigma

  imin=imagen
  tfile=mktemp("temp")

# obtengo los parametros estadisticos de la imagen
#
  imstat(imin,fields="mean,stddev",format-, > tfile)

# asigno al archivo un puntero
#
  arch=tfile

# leo el archivo
#
  dum=fscan(arch,vmed,sigma)

# libero el puntero
```

```
#
  arch=""

# borro el archivo
#
  delete(tfile,ver-)

  ...
  ...
 end
```

Primero se guarda la imagen de entrada (en la variable *imagen*) en una variable local (*imin*). Luego se utiliza la tarea *MKTEMP* para crear un archivo temporal con nombre único para ser utilizado en la tarea *IMSTATISTICS* para obtener el valor medio y la desviación standard. El parámetro oculto *format* permite o no sacar el encabezado indicando que es cada valor. Como queremos sólo los valores le decimos a la tarea que no lo imprima. La salida es redirigida al archivo temporal *tfile*. Por último se asigna el archivo al puntero y se lee con *FSCAN* los valores que interesan que quedan disponibles para su uso en las variables *vmed* y *sigma*. Finalmente, se libera el puntero y se borra el archivo temporal.

Es importante recordar que cuando se quiere redirigir información de la salida de una tarea a un archivo la instrucción de redirección debe ir siempre **AL FINAL** de la lista de parámetros de la tarea.

Por otro lado, cuando la información que muestra la salida de la tarea no es útil se puede redirigir a un archivo nulo para que no sea mostrada por pantalla:

```
  ...
  display (imag,1, > "dev$null")
  ...
```

4. **¿No hay otra forma de leer la información que la tarea saca a pantalla?**

Si, existen varias formas de leer la información que la tarea saca a pantalla. Si bien el método anterior es útil porque permite aprender rápidamente como leer y escribir archivos desde el script la forma más rápida de leer la infromación es utilizando la tarea *SCAN*. Esta tarea lee desde el STDIN por lo cual podemos hacer un pipe entre la tarea que saca la información y *SCAN* para asignar los valores a ciertas variables. Utilizando el ejemplo anterior:

```
 procedure xxxxx (imagen, ...)

# declaracion de parametros posicionales y ocultos
#
 string imagen   {prompt="Imagen a procesar"}
 ...

 begin

# declaracion de variables locales
#
  string imin
  int dum
  real vmed,sigma

  imin=imagen

# obtengo los parametros estadisticos de la imagen y los leo
#
  imstat(imin,fields="mean,stddev",format-, > tfile)|scan(vmed,si

  ...
  ...
 end
```

5. **¿Cómo expander una lista de imágenes?**

   IRAF permite trabajar con listas de imágenes para efectuar procesamien-

tos en serie y muchas veces sería de gran utilidad que un script tenga esta posibilidad. Las listas se pueden indicar directamente con los comodines "*" o "?" en los nombres (por ejemplo, "imagen-*.fits" o "imagen-0??.fits") o incluyendo los nombres en un archivo. Para identificar que un archivo es una lista el nombre del archivo debe ser precedido por el caracter "@", entonces se puede intentar detectar este identificador utilizando las tareas de manejo de strings que tiene IRAF (*SUBSTR*, *STRLEN*, etc.) pero sería necesario discriminar entre archivos de listas o listas con comodines. Por suerte existe un procedimiento más rápido utilizando la tarea *SECTIONS*:

```
 procedure xxxxx (imagen, ...)

# declaracion de parametros posicionales y ocultos
#
  string imagen   {prompt="Lista de imagenes a procesar"}
 ...

# declaracion de punteros
#
  struct *lista

  begin

# declaracion de variables locales
#
   string img,imin,tfile

   imin=imagen
   tfile=mktemp("temp")

# Expande la lista de imagenes
#
   sections (imin,option="fullname",>tfile)
   lista=tfile
```

```
# procesa imagen por imagen

   while (fscan (lista,img) != EOF){
    display(img, 1)
    ...
    ...
   }

   ...
   ...

# libero el puntero y borro el archivo
#
   lista=""
   delete(tfile,ver-)
   ...
   ...
  end
```

El script toma el nombre de la lista que se encuentra en la variable *imagen* y se la asigna a la variable *imin*. Luego crea un archivo temporario con la tarea *MKTEMP* para guardar las imagenes de la lista y con la tarea *SECTIONS* expande la lista y redirige la salida al archivo temporario. Este archivo se lee linea a linea para identificar las imagenes a procesar.

6. **¿Cómo leer y utilizar el cursor de imagen?**

   Ya se vio con anterioridad que IRAF puede leer el cursor grfico *gcur* y el cursor de imagen *imcur* utilizando:

   ```
   cl> =gcur
   ```

   para el cursor gráfico o:

   ```
   cl> =imcur
   ```

para el cursor de imagen. En ambos casos aparece un cursor que espera a que se presione alguna tecla para devolver las coordenadas (X,Y) en la imagen o gráfico, un código del sistema de coordenadas utilizado, y la tecla presionada. Esto se puede utilizar de muchas maneras pero una posibilidad es identificar estrellas en una imagen para gerenar una lista de coordenadas (X,Y). Si bien hay muchas opciones, un script que haga esta tarea podría ser el siguiente:

```
 procedure xxxxx (imagen,lista, ...)

# declaracion de parametros posicionales y ocultos
#
 string imagen   {prompt="Imagen a procesar"}
 string lista   {prompt="Archivo con las coordenadas"}
 ...

 begin

# declaracion de variables locales
#
  struct linea
  string imin,tfile,letra
  int dum,lee
  real cx,cy

  imin=imagen
  tfile=lista

# primero despliego la imagen
#
  display(imin,1,> "dev$null")

# asigno la salida del cursor de imagen
#
  linea=imcur
```

```
# lee por pirmera vez el cursor
#
  lee=fscan(linea,cx,cy,dum,letra)

# ve si la variable letra es Q o q
#
  while((letra != "q") && (letra != "Q")){

# guarda coord. XY
#
    print(cx,cy, >>tfile)

# vuelve a asignar el cursor de imagen y a leer
#
    linea=imcur
    lee=fscan(linea,cx,cy,dum,sn)
  }

  ...
  ...
 end
```

Lo que el script hace es asignar el cursor de imagen a la variable *linea* que es una variable tipo *struct*. No es posible usar una variable *string* porque se truncaría la salida en el primer espacio que lea. Luego de asignado el cursor se queda esperando hasta que se presione una tecla y se genere la salida la cual es leida en las variables *cx*, *cy*, *dum*, y *letra*. A partir de este momento el programa entra en un bucle en donde se guardan las coordenadas en el archivo y se lee nuevamente el cursos hasta que se presione la tecla Q o q.

7. **¿Cómo ejecutar un programa de FORTRAN o C desde un script de IRAF?**

   Muchas veces se utiliza un script de IRAF para obtener datos de imagenes

y crear un archivo de inicio para un programa externo de reducción o análisis. Si bien es posible usar el script de esta forma y ejecutar luego el programa desde UNIX para obtener los resultados buscados, sería mucha más util tener un unico script que cree el archivo de inicio, ejecute el procedimiento externo y utilice los resultados obtenidos para continuar el procesamiento. Por suerte existe una forma de hacer esto.

Cando desde IRAF queremos ejecutar alguna instrucción de UNIX anteponemos a la instrucción el carater "!":

```
cl> !cp arch1 arch2
```

esto copia el archivo *arch1* con el nombre *arch2* utilizando una instrucción de UNIX y sin salir de IRAF. Es posible hacer exactamente eso desde un script mediante:

```
 procedure xxxxx (arch1,arch2,...)

# declaracion de parametros posicionales y ocultos
#
 string arch1   {prompt="Archivo de origen"}
 string arch2   {prompt="Archivo de destino"}
 ...

 begin

# declaracion de variables locales
#
  ...
  ...
  print("!cp "//arch1//" "//arch2) | cl

  ...
  ...

 end
```

Lo que aqui se hace es utilizar la tarea *PRINT* para escribir en la terminal la instrucción que me permite ejecutar comandos UNIX desde IRAF (con el caracter "!" delante) pero en vez de sacarlo a pantalla se redirige a una nueva instancia del lenguaje de comandos *CL* por lo cual es ejecutado. De este modo es posible desde dentro de un script de IRAF ejecutar programas externos.