IRAF-CL の外から IRAF タスクを利用する

2009.6　中島　康 (名古屋大 / 国立天文台)

0. はじめに

IRAF の CL スクリプトを使わなくても、他のスクリプト (シェル、perl, python) を使って IRAF のタスクを利用することができます。CL スクリプトでは届かなかったカユいところに手が届いたり、CL のバグ (?) を回避できたり、他のスクリプトに精通している人には便利であったりします。シェルから IRAF タスクを呼ぶ shiraf(シャイラフ)、CPAN に置いてある perl module を使って perl から IRAF タスクを呼ぶ方法、python から IRAF タスクを使う pyraf、これら三種類についての基本を述べます。

1. shiraf

IRAF のホームページ http://iraf.nao.ac.jp/iraf/web/new_stuff/cl_host.html を見ると、Host CL Scripting Capability なるお題でつらつらと UNIX-shell script として IRAF cl を動かす方法が述べられています。
ここでは、その方法について具体的にどうやればいいのかを述べたいと思います。

1.1. 環境変数設定

arch を .redhat .sunos .linux .macosx など使っているマシンに合わせる
csh の場合の例　`setenv arch .sunos`
bash の場合の例　`export arch=.sunos`

arch が .redhat, .linux, .linuxppc の場合、stacksize を unlimited にする必要がある。
csh の場合　`limit stacksize unlimited`
bash の場合　`ulimit -s unlimited`

これらの設定は shiraf を使うターミナルに直打ちしてもいいですし、.cshrc や .bashrc で定義しておいてもよいでしょう。

mkiraf して login.cl を作っておく必要はありません。

1.2. スクリプト作成と実行

いくつかスクリプトの例を見てみましょう

例 : shiraf_display (もちろん勝手にこう命名しただけです。他と混同しなければ何でも)

```
#!/iraf/irafbin/bin.sunos/cl.e -f                    (1)
reset stdimage = imt1024                             (2)
logver = "IRAF V2.14EXPORT November 2007"            (3)
images                                               (4)
tv                                                   (5)
{                                                    (6)
  printf("display %s\n",args) | cl()                 (7)
  logout                                             (8)
}                                                    (9)
```

このスクリプトを path の通っているところに置いて実行。`chmod +x` もお忘れなく。

実行例 :

```
% shiraf_display hoge.fits 1
% shiraf_display hoge.fits 1 zs- zr- z1=100 z2=1200
```

(1) スクリプトの最初にこのおまじないを唱えましょう。マシンの arch や IRAF のインストール方法によって、cl.e のある場所が変わります。自分のマシンに応じて path を変えてください。

(2) デフォルトの imt512 から変えたい場合にはこのように設定してやります

(3) このあと使う images パッケージを使うためには logvar を定義する必要があります。使うタスク、パッケージに応じて IRAF の環境変数を定義してやります。

(4)-(5) display タスクを使うために images と imutil をオープンします。

(6) と (9) コマンド部分は { } でくくります。

(7) このようにして cl にコマンドを食わせます。 args はこのスクリプトに与える引数です。上の実行例では `hoge.fits` とか `hoge.fits 1 zs- zr- z1=100 z2=1200` がここに入ります。

(8) IRAF-CL から logout してやります。

例 : shiraf_imstat

```
#!/iraf/irafbin/bin.sunos/cl.e -f
task    $echo  = "$foreign"                          (1)
logver = "IRAF V2.14EXPORT November 2007"
images
imutil
{
  echo "imstat"                                      (2)
  printf("imstat %s\n",args) | cl()
  logout
}
```

実行例 :

```
%  shiraf_imstat  hoge.fits
imstat
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
            hoge.fits     78400     407.7     403.9        0.     4554.

% shiraf_imstat  hoge.fits fields=midpt,mean,stddev
imstat
#      MIDPT      MEAN    STDDEV
       359.4     407.7     403.9
```

(1) IRAF 外のコマンドをここで設定します。このスクリプトでは echo を使うのでこのように設定しました。他に追加したい場合は `task  $echo $cp $awk  = "$foreign"` といったように並べればよいです。
(2) 別になくてもよいのですが、imstat してますよ〜って画面に出してやってます。

その他、参考になるスクリプトが下記の URL にあります。
http://www.eso.org/lasilla/instruments/wfi/inst/zeropoints/zpmap/shiraf.tar

2. Astro-IRAF-CL perl module

perl module の Astro-IRAF-CL をインストールすることで、perl スクリプトの中から IRAF のコマンドを実行することができます。ここではその module のインストール方法、および簡単なスクリプトの例を示します。

2.1 Astro-IRAF-CL のインストール

http://search.cpan.org/~sjquinney/Astro-IRAF-CL-0.2.0/
の [Download] から tar ファイルをおとしてきます。その tar ファイルを展開して、展開したディレクトリ Astro-IRAF-CL-0.2.0 の中に移動します。

IRAF2.14 以降を使う場合には、この段階でファイルの修正を行います。Astro-IRAF-CL は IRAF2.14 以降の ecl には対応していません。そこで、IRAF2.14 以降を使う場合には展開したディレクトリにある CL.pm の 127 行目の spawn('cl') のところを spawn('cl -old') に書き換えます。そうしてから、

```
% perl Makefile.PL
% make
% make test
% sudo make install
```

を行います。

なお、perl module の Expect-1.15 (or later), IO:Tty があらかじめインストールされていない場合には warning が出ます。そのときにはまず
http://search.cpan.org/~rgiersig/IO-Tty-1.07/
から IO-Tty-1.07 あるいは IO-Tty-1.08 をダウンロード。tar を展開したディレクトリで

```
% perl Makefile.PL
% make
% make test
% sudo make install
```

を実行します。
次に
http://search.cpan.org/~rgiersig/Expect-1.15/
から Expect-1.15 もしくは Expect-1.21 をダウンロードして、同様に展開したディレクトリで

```
% perl Makefile.PL
% make
% make test
% sudo make install
```

を実行します。
という手順をふんで IO-Tty と Expect の module をインストールしてから、改めて Astro-IRAF-CL のインストールを行いましょう。

2.2 スクリプトの作成

まずは具体例を見ましょう。
imstat を使う例

```
#!/usr/bin/perl                                                    (1)

use Astro::IRAF::CL;                                               (2)

my $iraf=Astro::IRAF::CL->new();                                   (3)
$file = "hoge2.fits";                                              (4)

my $output1=$iraf->imstat("$file fields=image,midpt,stddev         (5)
lower=INDEF upper=INDEF nclip=3 lsigma=3. usigma=3.
binwidth=0.1 format=no cache=yes");

print "$output1 \n";                                               (6)
```

(1) perl スクリプトお決まりのじゅもん
(2) Astro-IRAF-CL のパッケージを使うことを宣言します
(3) これでパッケージ立ち上げ立ち上げです。$iraf の前の my は必須。(local 変数にしておく)
(4) perl の変数は全て$からはじまります。
(5) $iraf-> の後に IRAF のタスクと (' パラメータ ') を書きます。 imstat の出力が変数

$output1 に代入される。
(6) $output1 の中身を見てみましょう。

[実行結果の例]

```
% imstat.pl
hoge2.fits  1090.388  47.20227
```

スクリプト作成の作法

```
my $iraf=Astro::IRAF::CL->new();
```

としてやることで IRAF のセッションをはじめます。perl に堪能な人なら、「これは perl のオブジェクト指向プログラミングであり、新しいオブジェクトを作成している」というほうがすっきりするかもしれません。なお、使用するにあたってオブジェクト指向を理解している必要はありません。

この new() の中で、必要であれば各種初期設定をすることが可能です。

```
my $iraf=Astro::IRAF::CL->new(iraf_start=>"/hoge/hoge/",
                    debug=>1,
                    work_dir=>"/wrk/washi/",
                    display_startup=>1,
                    log=> *FH,
                    set=>{imtype=>"fits", stdimage=>imt1600},
                    package=>['digiphot','apphot']
                    );
```

iraf_start ： login.cl がどこにあるかを指定します。デフォルトは指定なしです。
指定なしの場合、
$HOME　<- 環境変数の HOME
$HOME/iraf
/home/ $username/iraf　　$usename は環境変数の USER もしくは
/home/ $username　　whoami の出力
の順で login.cl をさがします。
見つかった login.cl に記述されている設定を読み込む。

debug　： デフォルトは 0. debug=> 1 にすると実行されたコマンドの内容が出力される。
例

```
% imstat.pl
CL: imstat hoge2.fits fields=image,midpt,stddev lower=INDEF
upper=INDEF nclip=3 lsigma=3. usigma=3. binwidth=0.1
format=no cache=yes
hoge2.fits  1090.388  47.20227
```

work_dir ： コマンドを実行するディレクトリ。デフォルトはカレント

display_startup : =>1 にすると IRAF 起動時の画面 (コレ↓) を表示。

```
   NOAO/IRAFNET PC-IRAF Revision 2.14 Fri Nov 30 15:27:05 MST 2007
   This is the RELEASED version of IRAF V2.14 supporting PC systems.


Welcome to IRAF.  To list the available commands, type ? or ??.  To get
detailed information about a command, type `help <command>'.  To run  a
command  or  load  a  package,  type  its name.    Type  `bye' to exit a
package, or `logout' to get out  of the CL.    Type `news' to find  out
what is new in the version of the system you are using.

Visit http://iraf.net if you have questions or to report problems.

The following commands or packages are currently defined:


                  IIIII    RRRR     AAA    FFFFF
                    I      R   R   A   A   F
                    I      RRRR    AAAAA   FFF
                    I      R  R    A   A   F
                  IIIII    R   R   A   A   F

    apropos     fitsutil.   lists.      plot.       stsdas.
    color.      gemini.     mscred.     proto.      system.
    ctio.       gmisc.      nmisc.      salt.       tables.
    dataio.     images.     noao.       softools.   utilities.
    dbms.       language.   obsolete.   stlocal.    xdimsum.
```

普通はいらないのでデフォルトは 0

log　　: ここでファイルハンドルを指定すると、指定したファイルにログが残される。
デフォルトでは STDERR

set　　: デフォルトでは login.cl に記述されている変数設定をそのまま利用。

package　: login.cl で呼ばれていない package を開くときに使う。
ここで package を開かなくてもあとで load_package() を使って開くこともできる。SUN のマシンではなぜかこれらの機能が使えない。login.cl で呼び出しておきましょう。

コマンドの実行

二通りあります。

上記の例のように$iraf-> コマンド (" パラメータ ") とする方法。

もうひとつは

$iraf->exec(command=> " コマンド パラメータ ") とする方法。

例：`$iraf->exec(command=> "imstat $file fields=image,midpt,stddev lower=INDEF upper=INDEF nclip=3 lsigma=3. usigma=3. binwidth=0.1 format=no cache=yes");`

後者の方法ではコマンドの実行だけでなくエラーハンドリングなどができます。

```
$iraf->exec(command => ' print "hello world"  ',
        timeout => 10,
        timeout_handler => \&timeout_sub,
        death_handler => \&death_sub,
        error_handler => \&error_sub);
```

TIMEOUT　handling

タスクが固まってしまった場合、何秒かでタイムアウトにして終了させる。

timeout=> でその秒数を指定してやる。

timeout_handler=> timeout したときに指定した subroutine を実行

```
my $timed_out = 0;
$iraf->exec(command => 'print "hello"',
        timeout => 2,
        timeout_handler => sub {print $command . " timed out\n";
                                $timed_out = 1});
```

Death handling

CL 自身が死んだとき (まああありますよね)、それでもなんとか対応します。

```
$iraf->cd($workdir);
$iraf->exec(command => 'print "hello"',
       death_handler => sub {$iraf->restart(); $iraf->cd($workdir)});
```

このようにしておけば、死んでも restart をして作業ディレクトリに戻ります。
いっぱいジョブを流したいときにはこのような設定をしておくとよいかも。

CL error handling

CL からエラーが出たときになにか対応できます。

```
my $error = 0;
$iraf->exec(command => 'print "hello"',
       error_handler => sub {print "error occurred\n"; $error = 1});
```

2.3 その他

Warning handling

未対応。

Astro-IRAF-CL.LOCK

プログラムを Ctrl+C とかで途中で殺したときに、login.cl のあるディレクトリにこのファイルが残る。これが残ったままだと、再開できない。手動で消しましょう。

2.4 コメント

shiraf よりは多機能で、perl 使いにとってはプログラムが組みやすい。だが、バージョンバージョンが 0.2.0 のまま止まっている。2002 年 8 月以降に更新がない。pyraf がでてきたからだろうか。Waning handing については「将来的に対応する」と著者が記述しているがたぶん対応されないだろう。

2.5 講習会の Intel-Solaris マシンで Perl-IRAF を使うときの注意

Intel-Solaris に共通な問題であるのかどうかまだはっきりしませんが、Astro-IRAF-CL の Perl module と (少なくとも) 今回の実習用のマシンは相性があまりよくありません。でも使えない事はなく、少々工夫をしてやればよいです。というか、工夫が必要です。

(1) package コマンドで IRAF のパッケージを開く事ができない。
login.cl であらかじめ開いておきましょう。

(2) 実行結果を変数として返す事ができない。
テキストファイルに書き出しておいて、それを読み込むようにする。
imstat を使う例

```
#!/usr/bin/perl

use Astro::IRAF::CL;

my $iraf=Astro::IRAF::CL->new();
$file = "hoge2.fits";

my  $output1=$iraf->imstat("$file  fields=midpt,stddev
lower=INDEF  upper=INDEF  nclip=3  lsigma=3.  usigma=3.
binwidth=0.1 format=no cache=yes > output.txt");

open(IN, "output.txt");
$output1=<IN>;
close(IN);

print "$output1 \n";

unlink "output.txt";
```

太字になっているところが「工夫をした」場所です。

(3) 他にも講習会中に、人によっては (マシンによっては？)imstat のパラメータで fields=image を使うとうまく動かないなどの不具合が生じました。後日、私が再現しようとしてもその不具合を再現できなかったので、何が問題なのかは謎です。
ともかく、Intel-Solaris とは相性が悪いようなのでご注意を。

2.6 Perl-IRAF 演習

(1) サンプルの撮像データ (どれか 1 枚でよい) のバックグラウンドレベルのメジアンを測り、バックグラウンドレベルのメジアンをゼロにするスクリプトを作成しましょう。

(2) 7 枚の撮像データのそれぞれのバックグラウンドレベルのメジアンを測り、バックグラウンドレベルの平均値と標準偏差を画面に表示させましょう。また、7 枚全ての撮像データのバックグラウンドレベルをその平均値に合わせましょう。

準備 : 下のようにファイルを読み込みます。配列の使い方もついでに。やってみましょう。

```
#!/usr/bin/perl

@array=();
open(IN, "file.list");
while($line=<IN>){
   chomp $line;
   print "$line \n";
   push @array, $line;
}
close(IN);

for($i=0; $i<=$#array; $i++){
  print "$array[$i] \n";
}
```

(3) サンプルの撮像データ (どれか一枚でよい) について、DAOFIND で 20 シグマ以上の星を検出して、APPHOT で測光しましょう。結果は Instrumental でかまいません。

7 枚のサンプルの撮像データは sample ディレクトリにあります。
上記 (1)-(3) の回答例のスクリプトは sample-code ディレクトリにあります。
(Intel-Solaris 用への書き換えはしていません。)

3. PyRAF

ここではPyRAFのごく基礎を紹介します。こんなものも世の中にあるんですよという程度に。
本来ならPyRAFだけで講習会がひとつ組めそうなくらいに奥が深いです。
http://stsdas.stsci.edu/stsci_python_epydoc/

3.1 なぜPyRAF?

HSTのソフトウエアグループによって開発されました。彼らの主張は以下。

(1) IRAFではデバッグやエラーハンドルが難しい。詳細なエラーレポートが出ないので、どこにバグがあるのかが非常にわかりにくい。
(2) CLスクリプトなんていうのは光赤外天文屋しか使わない。もっと広く使われている言語を使えないか？
(3) IRAF以外のソフトウエアと統合しやすくしたい

Pythonが少なくとも1998年の時点では最善の選択であった。
　無料のオープンソースである。(比較 -> IDL : 商用で高価)
　ユーザーおよび開発者のコミュニティが深くて広い。これからも成長しそう。
　汎用である。
　軽い。
　比較的習得しやすい言語である。(比較 -> Perl)
　などなど
(Rubyも選択肢としてあり得た。Libraryの貧弱さで負けた。)

3.2 PyRAF-- コマンドラインおよびGUIの使用

普通にCLターミナルで操作するようにも使えます。

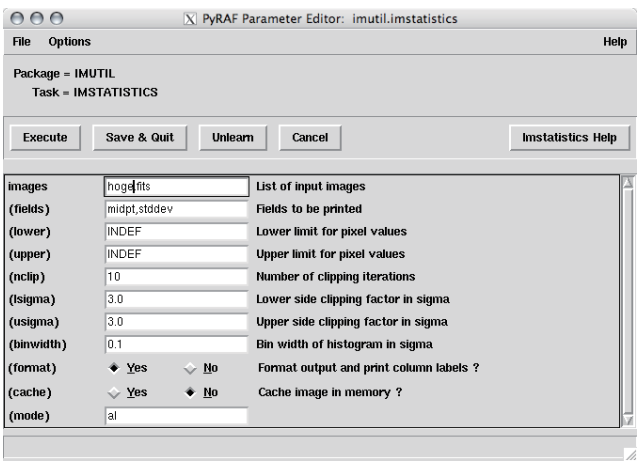
(1) pyrafを立ち上げます (プロンプトが --> になります) カレントかホームディレクトリのirafというディレクトリにlogin.clがあれば、ちゃんとlogin.clを読み込みます。

```
% pyraf
-->
```

(2) CLでやるようにコマンドうちます

```
--> imstat hoge.fits fields=midpt,stddev
#      MIDPT     STDDEV
       1090.      27.67
```

(3) epar imstat とやるとGUIが立ち上がります。 ここで実行もできます。



(4) こんな書き方もできる。

```
--> iraf.imstat('hoge.fits', fields="midpt,mean,stddev")
#      MIDPT       MEAN     STDDEV
       1090.      1093.      27.67
```

じつはこれが python の書き方。スクリプト作成のときにはこっちを使う。

(5) GUI が立ち上がるのが嫌だったら、パラメータ設定をこうやってもよい

```
--> iraf.imstat.fields="image,midpt,mode"
--> iraf.imstat('hoge.fits')
#                IMAGE      MIDPT       MODE
               hoge.fits      1090.      1084.
```

(6) 終了

```
--> .exit
```

3.3 PyRAF -- スクリプト

python の勉強もしなければいけないので、例だけ示します。

```
#! /usr/bin/env python

import sys
from pyraf import iraf

def run_imstat(input):
    iraf.images()
    for image in input:
        iraf.imstat(image)

if __name__ == "__main__":
    run_imstat(sys.argv[1:])
```

python のおまじない

おまじない
pyraf を呼びます

run_imstat のサブルーチン宣言
IRAF の image パッケージの立ち上げ
input に複数の FITS ファイル
iraf のタスクをパーツとして利用

main ルーチンがここに来る
python は tab でインデントしないと動かない

シェルのコマンドラインで

```
%  imstat.py hoge1.fits hoge2.fits
```

もしくは pyraf のコマンドラインで

```
--> import imstat_example
--> imstat_example.run_imstat(["hoge1.fits", "hoge2.fits"])
#      MIDPT    STDDEV
       1090.     27.67
#      MIDPT    STDDEV
       1061.     27.37
```

3.3 コメント

PyRAF と IRAF-CL の互換性は完全ではありません。PyRAF のホームページには
http://www.stsci.edu/resources/software_hardware/pyraf/what_is_pyraf
You can even run more than 90% of IRAF CL scripts!
と書かれているが残りの 10%が気になる。

- CL の GOTO 文はサポートされていない
- bye で package の unload するのは効かない
- バックグラウンドでの実行はできない
- Redirection of graphics streams (using >G) はサポートされない

などなどあるらしい

Known PyRAF Shortcomings に v1.1 beta の時点での欠点がまとめられている
http://www.stsci.edu/resources/software_hardware/pyraf/known_short
現状は v.1.8.0(2009 年 6 月更新) なので改善されている点も多いはず

次世代 IRAF のようなソフトウエアの作成には python が使われる可能性が非常に高いので、
python をやっといて損はないかもしれない。