

データをコピーしよう

- 各人のワークディレクトリを作る
 - `mkdir /adc/data/tekito`
 - `cd /adc/data/tekito`
- 実習用データ、プログラムをコピーする
 - `cp /data/yoshida/yoshida.tar.gz .`
 - `gunzip -c yoshida.tar.gz | tar xvf -`

※講習会担当者 注釈

上記の /data/yoshida 以下のファイル類
(iraf2010-1_yoshida.tar.gz; 上記の yoshida.tar.gz に対応)
は当WEBページ上からダウンロードできます。

IRAF講習会 CLスクリプトの作成

吉田道利

(HASC/Hiroshima University)

想定受講者その1

- IRAFは使える、もしくは、使ったことがある
- CLスクリプトは書いたことがない、もしくは、書いたことがあるが書き方を忘れてしまった
- 前回の講習会には出ていない

想定受講者その2

- バリバリのCL使いである
- 前講習会に出席し、その内容を完璧に理解しており、よりアドバンストな内容を期待している
- そういう方々は、用意したサンプルプログラムを即座に理解されるであろうから、これからのわたくしのお話を聞かずとも、ご自分でそれらのプログラムの発展的改造を行うことで実習可能と思われるので、ぜひそのような自習を行っていただきたい。

想定範囲外の受講者

- IRAFをいじったこともなければ、clが何なのか、さっぱり分からない
- そういう方々向けのお話を用意していません。とにかくサンプルプログラムで雰囲気をつかみ、後で自習してください。

参考資料（第一回資料改訂版）

- An Introductory User's Guide to IRAF Scripts
 - <http://iraf.noao.edu/iraf/ftp/iraf/docs/script.pdf>
 - 古い（V2.8用）が今でも有効
- IRAF CL Script Tips & Tricks
 - http://iraf.noao.edu/iraf/ftp/iraf/docs/script_intro.pdf
 - 比較的新しい（2003年）。プレゼンファイル。例が豊富
- Host CL Scripting Capability
 - http://iraf.noao.edu/iraf/web/new_stuff/cl_host.html
 - CLを外から利用する方法
- help language

今回の実習の道具立て

- `img/*.fits`
 - 実習に使用する画像
- `ffimg/*.fits`
 - フラットフィールド処理した画像(自習用)
- `calib/*.fits`
 - ダークとフラットフィールド画像(自習用)
- `myex*.cl`
 - 実習に使うサンプルプログラム
- `printit.cl`
 - 実習に使うサンプルプログラム
- `yoshida.cl`
 - パッケージ化プログラム。余裕があれば実習で教える。
- その他のファイル
 - 基本的に自習用。余裕があれば実習で触れる。

実習予定

1. 簡易CLスクリプト
2. CLスクリプトの基本構成 printit.cl
3. CLスクリプト特有の機能 printit.cl
 - 3.1 prompting
 - 3.2 mktempコマンド
 - 3.3 list directed parameters
 - 3.4 ワイルドカードとsectionsコマンド
4. 実習
 - 4.1 いかにして画像ヘッダを読むか myex1.cl
 - 4.2 いかにして画像を仕分けるか myex2.cl
 - 4.3 オーバースキャン領域を引いてみよう myex3.cl
 - 4.4 標準入力から文字列を読み取ろう myex4.cl
 - 4.5 スクリプトの中から外部プログラムを呼んでみよう myex5.cl
 - 4.6 画像ブラウザから座標を読み取ろう myex6.cl
 - 4.7 画像ブラウザを使った画像足し合わせ myex7.cl
5. より進んだ実習
 - 5.1 いかにしてパッケージを作るか yoshida.cl
 - 5.2 CLスクリプトを外部から呼ぶ方法 cldisp

0. Magic Words

- unlearn
 - CLでパラメータキャッシュをクリアする。task登録したCLスクリプトを書いている途中にパラメータを変えたら、必ずunlearnせよ。
- flprcache
 - プロセスの使ったキャッシュを掃除する。
- 動作が変な時はclを再起動

1. 簡易CLスクリプト

- パラメータなし型CLスクリプト

- IRAFコマンドを並べただけのものが一番簡便

```
ecl> vi myex0.cl
```

← myex0.clを作る

```
print("imcopy")  
imcopy img/MT8191.fits test.fits
```

```
ecl> task $myex0=myex0.cl
```

task登録のときにコマンド名に"\$"をつける

←コマンドにパラメータがないから
パラメータ付きのスクリプトをタ
スク登録するときは\$をつけない

mkscript

- 簡易CLスクリプトを生成するツール

```
ecl> mkscript
```

```
Script file name (scr.cl): tmp.cl
```

```
Task name of command to be added to script: imcopy
```

このあと、imcopyのepar画面が出る。

適当に編集して、:wqで抜ける。

```
Is the command ok? (yes): yes
```

```
Add another command? (yes): no
```

```
Is the script ok? (yes): yes
```

```
Submit the script as a background job? (yes): yes
```

ターミナルスクリプト

- CLのコマンドプロンプトから直接打つ。
- 例) FITSヘッダを読む

```
ecl> string ss1, ss2
```

```
ecl> imgets img/MT8191.fits title
```

```
ecl> ss1=imgets.value
```

```
ecl> ss2=substr(ss1, 2, 3)
```

```
ecl> =ss2
```

CLのデータ型

- int 整数型 32ビット
- real 実数型 指数部はEであらわす。例) 3.2E+8
- bool 判定型 yesかno
- string 文字列型
- file ファイル型
 - 実際は文字列型と同じ。ファイルアクセス可能かどうかなどの判定が入る?
- struct 特殊な文字列型
 - fscanなどで読むときに、stringだと空白文字で切られてしまう。structは空白も込みで文字列として認識する。古いバージョンでは長さ64バイトの制限があったが、今は無くなっているようだ
- gcur, imcur カーソルパラメータ型
 - グラフィック画面や画像ブラウザからカーソルパラメータを読む
- ファイル名などはstring, file, structのどれで読んでもたいした違いはない。相互に代入できる。ただし、空白文字のあるような文字列を扱いたい場合には、structを使用せよ。

2. CLスクリプトの基本構成

- パラメータ有り型CLスクリプトの構造

1. procedure宣言
2. 明示パラメータ(スクリプト引数)宣言
3. (隠れパラメータ宣言)
4. (list directed parameters宣言)
5. begin
6. スクリプトの中身
7. end

- タスクの登録法

```
ecl> task scr=scr.cl
```

task登録のとき"\$"は無し

←パラメータ(明示and/or隠れ)
がスクリプト中にあるから

printit.cl

```
procedure printit (file_name)
```

← 始まりはprocedure

```
string file_name
```

← スクリプト引数の宣言

```
struct *flist
```

← list directed parameterの宣言

```
begin
```

← スクリプトの中身はbeginからendまで

```
    struct line
```

```
    flist = file_name
```

```
    while( fscan( flist, line ) != EOF )
```

```
        print(line)
```

```
end
```

← スクリプトの中身はbeginからendまで

printit.clを実行してみよう

```
ecl> task printit=printit.cl
```

```
ecl> printit printit.cl
```


パラメータの読み取り

- `scan (p1, p2, ...)`
 - 標準入力から読み取って内部変数に格納
 - `ecl> string ss1`
 - `ecl> =scan(ss1)`
 - 何か文字列を打ち込む
 - `ecl> =ss1`
- `fscan(pp, p1, p2, ...)`
 - 内部変数ppから読み取って別の内部変数に格納
 - `ecl> string ss2`
 - `ecl> =fscan(ss1, ss2)`
 - `ecl> =ss2`

scanの便利な使い方

- IRAFコマンドの出力を内部変数に格納する

```
ecl> real x1, x2
```

```
ecl> imstat img/MT8191.fits field="min,max"  
format- | scan( x1, x2 )
```

```
ecl> =x1
```

```
ecl> =x2
```

stringとstructの違い

```
ecl> string moto = "I am fine"  
ecl> string ss1  
ecl> struct st1  
ecl> =fscan( moto, ss1 )  
ecl> =ss1  
ecl> =fscan( moto, st1 )  
ecl> =st1
```

printit.cl

```
procedure printit (file_name)
```

← 始まりはprocedure

```
string file_name
```

← スクリプト引数の宣言

```
struct *flist
```

← list directed parameterの宣言

```
begin
```

← スクリプトの中身はbeginからendまで

```
    struct line
```

```
    flist = file_name
```

```
    while( fscan( flist, line ) != EOF )
```

```
        print(line)
```

```
end
```

← スクリプトの中身はbeginからendまで

3. CLスクリプト特有の機能

- 覚えておくべき機能
 1. prompting
 2. mktempコマンド
 3. list directed parameters (LDP)
 4. ワイルドカードの取り扱いとsectionsコマンド

3.1 prompting

- ユーザーからの入力をうながすプロンプトは、パラメータの宣言のところで行う。
- 例) myex1.cl

```
procedure myex1( imlist )  
string imlist {prompt = "Input images"} ← プロンプト  
struct *flist  
begin  
    string infile, tmpfile  
    infile = imlist ← 内部変数に代入したときに  
                    プロンプトが出る  
    .....  
end
```

3.2 mktemp

- 一時使用ファイル (temporary file) を作るコマンド
- ファイル名を自動生成する
- スクリプト中で大変便利
- 例:

```
ecl> tmpfile = mktemp( "ppp." )
```

→ ppp.xxxx (xxxxは数字) というファイル名が自動生成されてtmpfileにアサインされる。

mktempしてみよう

```
ecl> string tmpfile
```

```
ecl> tmpfile = mktemp( "ppp." )
```

```
ecl> =tmpfile
```

```
ecl> print("hello", > tmpfile)
```

```
ecl> ls
```

```
ecl> del (tmpfile)
```

```
ecl> imcopy ("img/MT8191.fits", tmpfile)
```

```
ecl> ls
```

```
ecl> imdel (tmpfile)
```


3.3 list directed parameters(LDP)

- テキストファイルの中身を、改行で区切られた文字列の順序リストとして格納してくれる
- CLスクリプト中での宣言： 必ずbeginの前で、
`struct *ppp` あるいは `string *ppp`
- テキストファイルをLDPに格納するやり方
`ppp = file_name`
- CLスクリプト中でファイルの中身を順序読み出しするときは、必ずこれを使う。

LDPを使ってみよう

ターミナルスクリプトで試す。

```
ecl> struct *tlist
```

```
ecl> tlist = "table1"
```

```
ecl> while (fscan (tlist, s1) != EOF) {
```

```
>>> print(s1)
```

```
>>> }
```

3.4 ワイルドカードとsectionsコマンド

- IRAFでのワイルドカード
 - *、? などのふつうのUNIXワイルドカードと、@付きファイルリスト
- IRAFワイルドカードはsectionsコマンドで展開して標準出力に書き出す
 - sections(“*.fits”, option=“full”)
 - sections(“@list”, option=“root”)

sectionsを使ってみよう

```
ecl> sections( "table*", option="full" )
```

```
ecl> sections("@table1", option="full", >  
"contents" )
```

LDPとsectionsを組み合わせてワイルドカードを展開してファイルを読む

```
struct *flist
```

```
string infile, junk
```

```
tmpfile = mktemp("ppp.")
```

```
sections( infile, option="full", > tmpfile)
```

```
flist = tmpfile
```

```
while( fscan( flist, junk ) != EOF ) { ... }
```


infileに格納されたワイルドカードを展開して、tmpfileにファイル名リストとして入れる



tmpfileをLDPに格納



fscanでLDPから一つずつファイル名をjunkに読み込んでいく



4. 実習

4.1 いかにして画像ヘッダを読むか

myex1.cl

```
ecl> less myex1.cl
```

```
ecl> task myex1=myex1.cl
```

```
ecl> myex1 img/*.fits
```

```
tmpfile = mktemp("tmp$gh_tmp.")  
sections(infile, option="fullname", > tmpfile)  
flist = tmpfile
```

`sections`でワイルドカードを展開して、`mktemp`で作った一時ファイルに格納。それを`flist`というLDPに入れる。

```
while( fcan( flist, inname ) != EOF ) {  
    imgets( inname, 'title' )  
    obj = imgets.value  
}
```

`imgets`で読んだパラメータは、`imgets.value`で参照できる。

課題1

- FITS画像リストを入力して、FITSファイル名、天体名、露出時間、画像の平均値(mean)、画像のモード(mode)を出力するスクリプトを作れ。

出力例

```
img/MT8191.fits M81 60 1256. 1100.
```

```
img/MT8192.fits M81 60 1316. 1189.
```

```
....
```


露出時間を読んで、それを実数の 内部変数に格納する

```
ecl> string s1
```

```
ecl> real x
```

imagesでは一回に一つしか読めない

```
ecl> imgets( "img/MT8191.fits", "EXPTIME" )
```

```
ecl> s1 = imgets.value
```

← imgetsでは読んだ変数は文字列

```
ecl> x = real( s1 )
```

← 文字列を実数に変換

```
ecl> =x
```

imstatで画像の統計量を読むやり方は、「scanの便利な使い方」を参照

4.2 いかにして画像を仕分けるか myex2.cl

```
ecl> task myex2=myex2.cl
```

```
ecl> myex2 img/*.fits
```

```
hselect(param,"DATA-TYP,i_title,FILTER",yes) ¥  
| scan( datatyp, title, wavelen )
```

hselectで三つのヘッダ情報を読んで、その出力をscanがパイプから読み取って、datatyp, title, wavelen変数に格納

```
outfiles = datatyp//". "//title//". "//wavelen  
print( param, >> outfiles )
```

“//”は文字列の連結

4.3 オーバースキャン領域を引いて みよう myex3.cl

```
ecl> task myex3=myex3.cl
```

```
ecl> myex3 img/*.fits
```

```
ecl> epar myex3
```

```
procedure myex3( imlist )  
string imlist {prompt = "Input images"}  
  
string ovs="5:1020,1030:1070"
```

ovsはスクリプトの引数になってないので、隠しパラメータ。

```
sections( "@//tmpfile1//"/".O", option="full", > tmpfile2 )  
tmpfile1に格納されたファイル名(拡張子を除く)に".O"を付加して、それを  
tmpfile2に格納している。
```

4.4 標準入力から文字列を読み取ろう

myex4.cl

```
ecl> task myex4=myex4.cl
```

```
ecl> myex4
```

```
bool sw
while( scan( sw ) == 0 ) {
    print( sw )
}
```

scan()用いてbool型の変数swに標準入力からy(yes)かn(no)を読み込む。

課題2

- 複数の画像を次々に表示するスクリプトを作れ。
- myex4.clで使用した方法によって、画像を表示するタイミングをコントロールせよ。

4.5 スクリプトの中から外部プログラム を呼んでみよう myex5.cl

```
ecl> !gcc hello.c -o ./hello
```

```
ecl> !gcc mktable.c -o ./mktable
```

```
ecl> task myex5=myex5.cl
```

```
ecl> myex5
```

```
task $hello = ("$"//osfn(".$")// "hello")
```

外部プログラムはパラメータ無しCLスクリプトと同じようにしてタスク登録できる。ホストの環境変数を参照するには、osfnを使う。

4.6 画像ブラウザから座標を読み取ろう myex6.cl

```
ecl> !ds9 &
```

```
ecl> task myex6=myex6.cl
```

```
ecl> myex6 ffimg/MT8191.F.fits pos.txt
```

```
while( fscan( imcur, xx, yy, wcs, command ) != EOF ) {  
    key = substr( command, 1, 1 )  
    if( key == "q" )  
        break  
    else {  
        printf( "Clicked position is %.1f %.1f¥n", xx, yy )  
        print( xx, yy, >> oput )  
    }  
}
```

課題3

- 画像ブラウザを使って、矩形領域の総カウント値を求めるスクリプトを作れ。
- 模範解答は、`/data/yoshida/sphotoex.cl`

4.7 画像ブラウザを使った画像足し 合わせ myex7.cl

```
ecl> task myex7=myex7.cl
```

```
ecl> myex7 ffimg/*.fits ffimg/MT8191.F pos.txt M81.fits
```

```
tmpfile2 = mktemp( "tmp$shimg." )  
tmpfile3 = mktemp( "tmp$shimg." )  
<中略>  
print( mktemp( "shimg" ), >> tmpfile2 )  
<中略>  
imalign( inim, refim, refp, "@"//tmpfile2, shifts=tmpfile3, mode=mode )  
imcombine( "@"//tmpfile2, outim, combine="average" reject="none",  
           mode=mode )
```

mktempで一時ファイル名を作って、tmpfile2にリストとして格納。それをimalignに渡して、位置合わせした画像群を作る。それらを最後にimcombineで合成。

課題4

- myex7.clで、最後のimcombineに与える combineとrejectのパラメーターを(myex7.clの中ではそれぞれ“average”, “none”で決め打ち)、それぞれスクリプトの隠しパラメーターとして指定できるように改造せよ
- 模範解答は、/data/yoshida/imgcombex.cl

5. より進んだ実習

5.1 いかにしてパッケージを作るか

yoshida.cl

- コードを参照して、自分の名前を付けたパッケージを作れ

5.2 CLスクリプトを外部から呼ぶ方法

cldisp

- コードを参照し、自分で中身を変更(いろんなIRAFタスクに変更・追加してみる)して動作を確認し、使い方を学べ。
- コード実行の前に、
 - setenv arch .sunos

5.3 データ解析パイプライン

- サンプルデータ (imgディレクトリ) を用いて、
 1. オーバースキャンの自動差引
 2. ダークデータの自動生成
 3. 天体画像からダークの差引
 4. フラットフィールド (calibディレクトリのskyflat.fitsを用いて良い)
 5. 画像の重ね合わせ (myex7.clを参考に、インタラクティブにやっても良い)

までを行うパイプラインを作れ。