IRAF講習会テキスト (16-18, Feb, 2011)
シェルスクリプトからIRAFを使う

東北大学大学院理学研究科天文学専攻
板 由房

1 このテキストについて

1.1 想定読者

このテキストは、以下のような人を想定して書きました。

- UNIX がある程度使える。
- 学部の授業で何らかのプログラミング言語を習い、少し使える。
- IRAF を何度かは使った事がある。
- CL スクリプトを書いた事が無い、または初心者。
- シェルスクリプト (bash) を書いた事が無い、または初心者。

CL スクリプト、シェルスクリプトをマスターしている人にとっては、物足りない内容だと思いますし、僕が教えられる事はもはや何もありませんので、更なる高みを目指してどんどん自習してください。

1.2 参考にした Web ページ、文章等

このテキストは、以下の Web ページ、文章等を適宜参考にしながら作成しました。

- 第 1 回 IRAF 講習会の大薮 進喜氏のテキスト
  http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo/2009/IRAF2009-1/iraf2009-1_oyabu.pdf
- 第 2 回 IRAF 講習会の吉田 道利氏のテキスト
  http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo/2009/IRAF2009-2/iraf2009-2_yoshida.pdf
- An Introductory User ’s Guide to IRAF Scrips
  http://iraf.noao.edu/iraf/ftp/iraf/docs/script.pdf
- IRAF CL Script Tips & Tricks
  http://iraf.noao.edu/iraf/ftp/iraf/docs/script_intro.pdf
- Host CL Scripting Capability
  http://iraf.noao.edu/iraf/web/new_stuff/cl_host.html
- A Beginner’s Guide to Using IRAF
  http://www.astro.pomona.edu/webdocuments/iraf/beg/beg-toc.html
- シェルスクリプト入門
  http://www.k4.dion.ne.jp/〜mms/unix/shellscript/index.html

1.3 このテキストの中身

1. CL スクリプトの書き方
2. シェルスクリプト (bash) の書き方
3. シェルスクリプト内での IRAF の使い方

IRAF(Imaging Reduction and Analysis Facility) はその名の通り、多彩な機能を持った解析&分析タスクをまとめた「Facility」ですので、このテキストで全てのタスクについて触れる事はできません。このテキストで扱っているのは IRAF のごく一部のタスクである事に注意してください。

1.4 サンプルデータ、サンプルスクリプトについて

1.4.1 サンプルデータ

サンプルデータは、第 2 回 IRAF 講習会で吉田 道利氏が用意されたイメージデータを使います。img というディレクトリの中に、

- 銀河 (M81) の画像 : MT819[1-8].fits
- ダーク画像 : MT869?.fits, MT870?.fits, MT871?.fits
- フラット画像 : skyflat.fits

が入っています。

1.4.2 サンプルスクリプト

このテキスト中で紹介&説明している以下のスクリプトをサンプルとして配布します。

- iterstat.cl : STSDAS パッケージに入っている、iterstat というタスクの CL スクリプト (ソースコード)。
- sample0[1-5].sh : この講習で使う、サンプルシェルスクリプト。まあ動くとは思いますが、いかんせん板オリジナルな物なので不備があったら教えて下さい。

2 はじめに

2.1 IRAF を使ってデータ解析を行う事のメリット、デメリットについて

- ○ 無料。これが IRAF を使う最大のメリット。
- ○ 多くの先達によって開発&改良が重ねられただけあって、優れたタスクが多く存在する。それをそのまま使える。
- △ メモリーを上手に使えないので長大なソフトの作成には向かない。× ではなく △ な理由は、僕自身は 1 つの CL スクリプトに色々な機能を詰め込みすぎず、1 スクリプト 1 機能ぐらいの短さが良いと思っており、さほど問題ではないと思っているからです。機能毎に分かれていた方がソースコードも読みやすいし。
- × デバッカ、プロファイラが無い。「超常現象」に遭遇する事もある。ただし、超常現象はよくよく調べると人為的なミスやコードの不備が原因で起こる事がほとんどで、IRAF に悪態をつく前に本当に IRAF のせいなのかもう一度良く考えるべきだと思う。「よく考える」時にデバッガが無いのが悲しい所だけど、自分で工夫してバグ取りする。

3 最低限知っておくべき IRAF のお作法

3.1 login.cl はありますか？

ホームディレクトリには login.cl があるが、ワーキングディレクトリには login.cl が無い、そんな状況で、ワーキングディレクトリで IRAF を立ち上げると、Warning: no login.cl found in login directory と怒られる。それでも見た目では IRAF は立ち上がるが、通常ロードされるパッケージがロードされないので、いつもと違う挙動をする。この場合は一旦 IRAF をログアウトし、ホームディレクトリの login.cl をワーキングディレクトリにコピーしてからワーキングディレクトリで IRAF を立ち上げるか、ホームディレクトリで IRAF を立ち上げて IRAF の中で change directory をするようにして下さい。

CL スクリプトが理解できるようになった後、一度 login.cl の中身を見て、中に書かれている事が全部理解できるか自分を試してみると良いです。どのようなパッケージが IRAF 起動時に知らない間にロードされているか等がわかりますか？

3.2 キーワードをもとに、どんな IRAF のタスクがあるかを調べたい時

- ref キーワード
- 使用例：ref subtract

こうすると、ソフトの 1 行紹介文に subtract というキーワードが入っている IRAF タスクを表示させる事ができる。他のキーワードでも試してみて下さい。Google のように賢くはないので、IRAF のキーワードサーチはサーチする側が頭を使う必要があります。例えば、ref calculation としても何もひっかかりませんが、ref calculate とすると、沢山タスクがひっかかります。もし何もひっかからなくても、めげずに色々変えてやってみてください。

3.3 タスクのパラメーターを編集したい時

- epar タスク名
- 使用例：epar imexam

パラメーター編集後は、:q(コロンの後に、アルファベットのキュー) で編集画面を抜ける事ができる。パラメーターを編集し、そのまま動かしたい場合は、:go(コロンの後に、ゴー) で、編集したパラメータでタスクを動かす事ができる。

3.4 タスクのヘルプが見たい時

- help タスク名
- 使用例：help imexam

3.5 動作が何か変な時

まずはエラーメッセージを良く読み、解決できそうだったら解決してください。エラーメッセージは決して親切ではなく、的外れな事を言ってくる場合もあります。その場合、以下を試してください。

- unlearn
  パラメータキャッシュをクリアする。task 登録した CL スクリプトにバグを発見したり、アップグレードしたり等で、スクリプトで使用する引数を変えたい場合がある。その場合は必ず unlearn しなければならない。また、タスクのパラメーターをいじくっては見たが、なんだかデフォルト値に戻したくなった、という場合も unlearn するとパラメーターの値がデフォルト値に戻る。
- flprcache
  プロセスの使ったキャッシュを掃除する。

それでもダメな場合は、

- ctrl + c (タスクの強制終了) & logout & IRAF 再起動

をしてください。

3.6 IRAF タスクの使い方

IRAF タスクの使い方には 2 種類ある。

1. command mode : taskname arg1 arg2 .... argN par1=val1 par2=val2 .... parN=valN
2. compute mode : taskname (arg1, ..., argN, par1=val1, par2=val2, ..., parN=valN)

command/compute mode の違いは、括弧の有無、カンマの有無、と、compute mode では引数が文字列の場合は“”でくくる必要がある所。微妙な差なので間違いやすく、注意が必要。

3.7 その他、知っていると便利な IRAF のお作法

- ! マークを先頭につけることで、IRAF の中から全ての UNIX コマンドを使う事ができる。自分で作ったコンパイル済みのバイナリもしかり (例えば C で書いたソフト等)。この機能は非常に便利で、CL スクリプトの中でも同様に使う事ができる。これは覚えておくべき。
  使用例：! head -n 10 file
- help language
  IRAF の中でこうやると、CL スクリプトの中で使える命令一覧を見る事ができる。
- Builtin Commands and Functions がある。例えば
  `http://www.astro.pomona.edu/webdocuments/iraf/beg/beg-apA.html` を参照。

4 CL スクリプトの作成

4.1 CL スクリプトで使える変数の型

- int : 整数型 32bit
- real : 実数型 double 相当で、指数は E で表す。例えば 3.2E8 等。
- bool : 判定型 yes または no
- string : 文字列型
- file : ファイル型
- struct : 特殊な文字列型
  fscan などで文字列を読み込む時に、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。ファイル名等は string、file、struct のどれでもかまわず、相互に代入できる。空白文字がある文字列を扱いたい場合は struct を使用する。

4.2 リダイレクトの仕方

- > file : spool output in a file
- < file : read input from a file (rather than the terminal)
- >> file : append the output to a file
- > & file : spool both error and regular output in a file
- >> & file : append both error and regular output to a file
- 使用例 1 : type *.cl > hoge.txt
- 使用例 2 : type (”*.cl”, > ”hoge.txt”)

4.3 簡易 CL スクリプトを生成するタスク=mkscript

IRAF には mkscript というタスクがあり、これを使う事で簡易 CL スクリプトを作る事ができる。使い方は、mkscript と入力して、あとは聞かれる質問に答えていくだけ。以下に imarith の例を示す。

```
ecl> mkscript
Script file name (script.cl): tekitou.cl
Task name of command to be added to script: imarith
```

(画面が変わり、以下のようなパラメータ入力画面になる)

```
                                  I R A F
                       Image Reduction and Analysis Facility
PACKAGE = imutil
   TASK = imarith

operand1=               a.fits  Operand image or numerical constant
op      =                    +  Operator
operand2=               b.fits  Operand image or numerical constant
result  =               c.fits  Resultant image
(title  =                     ) Title for resultant image
```

```
(divzero=                    0.) Replacement value for division by zero
(hparams=                      ) List of header parameters
(pixtype=                      ) Pixel type for resultant image
(calctyp=                      ) Calculation data type
(verbose=                   no) Print operations?
(noact  =                   no) Print operations without performing them?
(mode   =                   ql)
```

パラメーターを編集後、:q で抜ける

```
imarith ("a.fits",
"+", "b.fits", "c.fits", title="", divzero=0., hparams="", pixtype="",
calctype="", verbose=no, noact=no)
Is the command ok? (yes):  yes
Add another command? (yes):  no
```

他のコマンドを続けてスクリプトにしたい場合は yes と答える。
そうすると、コマンドは何かと聞かれ、パラメーター編集画面になる。

```
imarith ("a.fits",
"+", "b.fits", "c.fits", title="", divzero=0., hparams="", pixtype="",
calctype="", verbose=no, noact=no)

Is the script ok? (yes): yes
Submit the script as a background job? (yes): no
```

今作ったスクリプトを走らせたい場合は yes と答える。
no と答えた場合は、tekitou.cl が作られ、mkscript が終了する。

tekitou.cl を表示してみると中身はこんな感じ。

```
ecl> !cat tekitou.cl
imarith ("a.fits",
"+", "b.fits", "c.fits", title="", divzero=0., hparams="", pixtype="",
calctype="", verbose=no, noact=no)
```

僕はこの mkscript を CL スクリプトそのものを作る用途には使っていない。僕は CL スクリプトを作る際に mkscript で一旦テンプレートを作り、これをコピー&ペーストする事で省入力をするために mkscript を使っている。mkscript を使わないとそのタスクにどのようなパラメータがあるかを一旦調べ、パラメーター名を自分で手打ちしなくてはならないので面倒。CL スクリプトを作る際には使用するタスクの数だけ tekitou.cl を何度も作りかえ、自分のスクリプトにコピー&ペーストを繰り返すのが良いだろう。

4.4 CL スクリプトの基本構成

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 parameter 宣言
   LDP は IRAF 独特の機能で、ファイルの中身を順次読み、EOF が来ると勝手に Close する。パラメータの名前の先頭に * をつける事でこれは LDP であると明示できる。ややこしいので、次サブセクションのサンプルスクリプトをみて理解してください。

5. begin
6. スクリプトの中身
7. end

4.5 サンプル CL スクリプト

STSDAS という IRAF 上で動くソフトウエアパッケージがある。その中の HST のデータを解析するパッケージに iterstat.cl というタスクが含まれている。これはその名の通り imstat を飛び値を除きながら iterative にかけるタスクである。言うまでもないが、HST のデータだけでなくどんなデータにでも使える。このタスクのソースコードは公開されており、そのソースコードの中に CL スクリプトの全てが詰まっている。このソースコードをサンプルとし、まねて書く事によって、ほぼどんな機能を持った CL スクリプトでも書けるだろう。//から始まる行が、僕が適宜書き込んだ所です。

```
procedure iterstat(image)

// 始まりは、 procedure
// iterstat は procedure の名前。
// 名前だけでどういう意図のソフトかわかるようにつけるのが良い。
// 明示パラメータ ( スクリプト引数 ) は、 image

# Script to find image statistics excluding deviant pixels
# 4 August 1992 by John Ward
# Minor modifications 4 August 1992 MD
# Various subsequent variations.
# Latest revision: 18 Aug 1993 MD
# Modifications for public release: Howard Bushouse, 13 April 1999

// このように、コメント文は頭に#をつける事で実現できる。
// スクリプトの先頭には、このスクリプトが何を意図して作った物か、
// 改訂があったらその日付等を書いておくべき。

string  image   {prompt="Input image(s)"} // 明示パラメータの宣言

// 以下は、隠しパラメーター
real    nsigrej {5.,min=0.,prompt="Number of sigmas for limits"}
// 中括弧の中に注目。
// デフォルト値を 5 に設定し、
// 取り得る値の最小値を 0 に設定している。 min=0
// 最小値の他に、更に最大値を設定したければ、min=0 に続けて、
// , ( カンマ ) の後に例えば max=50 のように書く。
// prompt には、パラメータの説明を " " で囲って書く。
int     maxiter {10,min=1,prompt="Maximum number of iterations"}
bool    print   {yes,prompt="Print final results?"}
bool    verbose {yes,prompt="Show results of iterations?"}
real    lower   {INDEF,prompt="Initial lower limit for data range"}
real    upper   {INDEF,prompt="Initial upper limit for data range"}
int     npix    {prompt="Returned value of npix"}
real    mean    {prompt="Returned value of mean"}
real    sigma   {prompt="Returned value of sigma"}
```

```
real    median  {prompt="Returned value of median"}
real    valmode {prompt="Returned value of mode"}
#               Must be "valmode" to avoid conflict w/ omnipresent task
#               parameter "mode"
# Yoshifusa Ita added : query(ql) / learn(al) parameters
string  mode="al"
struct  *inimglist  // list directed parameter の宣言

begin // スクリプトの始まりは begin

        string  imglist         # equals image
        string  infile          # temporary list for files
        string  img             # image name from fscan
        real    mn              # mean from imstat
        real    sig             # stddev from imstat
        real    med             # midpt from imstat
        real    mod             # mode from imstat
        real    ll              # lower limit for imstat
        real    ul              # upper limit for imstat
        int     nx, npx         # number of pixels used
        int     m               # dummy for countdown

        # Get query parameter
        imglist = image

        # Load necessary packages
        if (!defpac("images")) images
        if (!defpac("imutil")) imutil

        # Expand file lists into temporary files.
        infile =  mktemp("tmp$iterstat")
        sections (imglist,option="fullname",>infile)
        inimglist = infile

        # Loop through images
        while (fscan(inimglist,img) != EOF) {

                # Compute image statistics
                imstat(img,fields="mean,stddev,npix,midpt,mode",lower=lower,
                        upper=upper,format-) | scan(mn,sig,npx,med,mod)
                        // このように、scan はパイプでつなぐと前のコマンドの出力結果も読める。
                if (verbose)
                    print (img,": mean=",mn," rms=",sig," npix=",npx,
                            " median=",med," mode=",mod)

                # Loop over rejection cycles
                m = 1
                while (m <= maxiter)  {

                        ll = mn - (nsigrej*sig)
```

```
                    ul = mn + (nsigrej*sig)
                    if (lower != INDEF && ll < lower) ll = lower
                    if (upper != INDEF && ul > upper) ul = upper
                    imstat(img,fields="mean,stddev,npix,midpt,mode",lower=ll,
                        upper=ul,format-) | scan(mn,sig,nx,med,mod)

                    if (nx == npx)
                        break

                    if (verbose)
                        print (img,": mean=",mn," rms=",sig," npix=",nx,
                            " median=",med," mode=",mod)

                    npx = nx
                    m = m + 1
                }

                if (print && !verbose)
                    print (img,": mean=",mn," rms=",sig," npix=",nx," median=",
                        med," mode=",mod)

                # Store results in output parameters
                npix = npx
                mean = mn
                sigma = sig
                median = med
                valmode = mod

        } # End of loop over images

        # Clean up temp files
        delete (infile, verify-, >& "dev$null")
        inimglist = ""

end
```

4.6 CL スクリプトを IRAF タスクとして登録するには？

作った CL スクリプトは以下のようにして IRAF に登録する事で使えるようになる。ただし、タスクに明示&隠しパラメータがあるか無いかで、ビミョーに登録の仕方が異なる。

4.6.1 パラメータありの場合

上で見た iterstat には、image という明示パラメータがあった。こういう場合は、以下のようにする。

```
ecl> task iterstat=/フルパス/iterstat.cl
```

この方法だと、IRAF をログアウトすると iterstat はタスク登録から抹消されてしまう。IRAF を立ち上げる度に毎回タスク登録をする作業が面倒な場合は、〜/login.cl の適当な箇所に以下の一行を書き込むと良い。

```
task iterstat=/フルパス/iterstat.cl
```

ただし、ワーキングディレクトリがホームディレクトリと異なる場合は、ホームディレクトリの login.cl をワーキングディレクトリにコピーするのを忘れると、IRAF 起動時に iterstat が task 登録されず、使えない。自分が非常に忘れっぽい人間で、こんなの嫌だ、どんな時も登録して欲しいという場合は、ルートになって/iraf/iraf/unix/hlib/extern.pkg に、以下の一行を書き込むと良い。

```
task iterstat=/フルパス/iterstat.cl
```

これは、IRAF に別のパッケージ (STSDAS 等) をインストールする手順とまったく同じです。
　一旦登録が終わると、あとは普通の IRAF のタスク同じように使用できます。epar iterstat 等も効きます。やってみて下さい。

4.6.2 パラメータなしの場合

パラメータが無いタスク (例えば IRAF コマンドを並べただけの物等) を登録したい場合は、以下のようにコマンド名の頭に$をつける。あとはパラメータ有りの場合と一緒。例えば、

```
ecl> !cat simple.cl
print ("imcopy")
imcopy a.fits b.fits
```

のようなシンプルなタスクの場合は、

```
task $simple=/フルパス/simple.cl
```

のように$をつける。この微妙な違いが、IRAF タスクのパラメータ有り無しを区別している。

5　シェルスクリプト (bash,sh) の作成

このテキストで単にシェルスクリプトと言った場合は、bash または sh で書かれたスクリプトを指す事にします。bash と sh の違いは何か？という問いの答えは適宜 Google 等で調べてみて下さい。僕は単純に sh に出来て bash に出来ない事は無いというような理解で良いと思っています。tcsh や csh でスクリプトを書いてはいけないのか？という疑問も当然あると思いますが、僕は bash か sh で書く事をお勧めします。これも理由は Google 等で調べて見て下さい。

シェルスクリプトは、複数の処理をまとめて行なうためのスクリプトです。なので、一番簡単なシェルスクリプトは、ただ単に実行したい UNIX コマンドをそのまま順番に書いたものです。このため、シェルスクリプトを作るには、まず有用な UNIX のコマンドと、その使い方を知っておく必要があります。

5.1　知っていると便利な UNIX のコマンドと使い方

- “：unix_command1 ‘unix_command2‘のように、‘と ‘で UNIX のコマンドを囲む事によって、その結果を前のコマンドに渡す事ができる。
  使用例：ds9 ‘ls -1 *.fits‘ とやると、ds9 *.fits と同じ。

- | : パイプは便利。俗に言うワンライナーと呼ばれている人達 (パイプでつなげて一行でプログラムを書く事を格好いいと思う変わった人達) はパイプを駆使している。
  使用例：du -sk * | sort -rn | head

- ワンライナー御用達のスクリプト言語、awk、perl と、UNIX の便利コマンド sed。

  使用例：`cat file_with_tabs.txt | perl -pe 's/\t/ /g'`
  使用例：`cat file1 file2 file3 | awk '{print $1}'`
  使用例：`cat file1 file2 file3 | sed -i -e "s/text to replace/final text/g"`

  sed は文字列の置換や削除等の他、機能満載でかなり便利。置換は tr、文字列操作は perl でもできるけれど、sed でやるのが格好いいと僕は思う。あと、意外に良く使うのが

  - sort : ソートする。
    ex. sort -n -k 1 -r file
  - paste : cat はファイルを縦に繋げる事ができるが、paste は横に繋げる事ができる。
    ex. paste -d ” ” file1 file2 > file3
  - uniq : 同じ内容の行を除いたり、同じ内容の行のみを表示させる事ができる。
    ex. cat file1 file2 | sort -k 1 | uniq
  - cut : ファイルの一部分を切り出す事ができる。
    `ex. zcat hip_main.dat.gz | cut -c  42-46  | perl -e 'while(<>){chomp;`
    `if($_ =〜 m/^\s*$/){print("99.99\n");}else{print("$_\n");}}' > Vmag.dat`
  - bc : 実数演算。
    echo ”52.1 * 12.3” | bc -l
  - head, tail : ファイルの先頭、末尾を表示する
  - join : ファイル 1 とファイル 2 に同じインデックスがある場合、それを頼りにファイル 1 と 2 を繋げる事ができる。

で、これらは絶対に使い方を覚えておくべき UNIX コマンドである。

5.2 シェルスクリプトの予約変数

シェルスクリプトでは、いくつかの変数は予約されており、特別な意味を持っている。これを特殊変数と呼ぶ。特殊変数と同じ名前の変数は使えないので注意が必要。また、これらの特殊変数は、参照専用で値を代入することはできない。以下に代表的な特殊変数を示す。

変数	意味
$$	シェル自身の PID(プロセス ID)
$!	シェルが最後に実行したバックグラウンドプロセスの PID
$?	最後に実行したコマンドの終了コード（戻り値）
$-	set コマンドを使って設定したフラグの一覧
$*	全引数リスト。”$*”のように”で囲んだ場合、”$1 $2 … $n” と全引数を一つにした形で展開される。
$@	全引数リスト。”$@”のように”で囲んだ場合、”$1” ”$2” … ”$n” のようにそれぞれの引数を個別にダブルクォートで囲んだ形で展開される。
$#	シェルに与えられた引数の個数
$0	シェル自身のファイル名
$1 ~ $n	シェルに与えられた引数の値。$1 は第 1 引数、$2 は第 2 引数…

5.3 シェルスクリプトの基本構成

```
#!/bin/bash
```

これをファイルの先頭に書くだけ。これは、この下に書かれているコマンドを/bin/bash で処理してください、というおまじないです。これに続けて、あとはコマンドを並べたり、制御文を書いたりしていけば良いです。一番簡単なシェルスクリプトのサンプルとして、前サブセクションで紹介した特殊変数の値を表示するシェルスクリプトを示します。
% cat sample01.sh

```
#!/bin/bash
#シェル変数の中身を表示してみる。
echo "\$$" $$
echo "\$!" $!
echo "\$?" $?
echo "\$-" $-
echo "\$*" $*
echo "\$@" $@
echo "\$#" $#
echo "\$0" $0
echo "\$1" $1
echo "\$2" $2
echo "\$3" $3
```

このように、値の参照には、echo を使う。シェルスクリプトの使い方は、

```
chmod 755 sample01.sh
```

のようにして、スクリプトを記述したファイルに実行属性をつけたあと、

```
./sample01.sh
```

とするか、bash で書いた物であれば、

```
bash ./sample01.sh
```

とする事で、スクリプトを動かす事ができる。

```
bash ./sample01.sh abc d e
```

等と、引数を与えて、まずは使ってみて下さい。

5.4 シェルスクリプト (bash,sh) の制御構文

5.4.1 if

```
if [ 条件 1 ]
then
    処理 1
elif [ 条件 2 ]
then
    処理 2
else
    処理 3
fi
```

条件の前後に半角スペースは必須。if [条件] を if test 条件と書く事もできる。if と次の while, until 等で使う条件の所は、次のような比較ができる。

数値 (整数) 比較　数値の比較の仕方は以下。数値は整数しか扱えない。実数の大小を比較したい場合は別の手を用いる必要がある。それは後の IRAF をシェルスクリプトの中から使う所で示す。

演算子	意味
数値 1 -eq 数値 2	数値 1 と数値 2 が等しい場合に真
数値 1 -ne 数値 2	数値 1 と数値 2 が等しくない場合に真
数値 1 -gt 数値 2	数値 1 が数値 2 より大きい場合に真
数値 1 -lt 数値 2	数値 1 が数値 2 より小さい場合に真
数値 1 -ge 数値 2	数値 1 が数値 2 より大きいか等しい場合に真
数値 1 -le 数値 2	数値 1 が数値 2 より小さいか等しい場合に真

文字列　文字列の比較の仕方は以下

演算子	意味
文字列 1 = 文字列 2	2 つの文字列が等しければ真
文字列 1 != 文字列 2	2 つの文字列が等しくなければ真

ファイル、ディレクトリチェック　ファイル、ディレクトリチェックの仕方は以下

演算子	意味
-d ディレクトリ名	ディレクトリなら真
-f ファイル名	通常ファイルなら真
-L ファイル名	シンボリックリンクなら真
-r ファイル名	読み取り可能ファイルなら真
-w ファイル名	書き込み可能ファイルなら真
-x ファイル名	実行可能ファイルなら真
-s ファイル名	サイズが 0 より大きければ真
ファイル 1 -nt ファイル 2	ファイル 1 がファイル 2 より新しければ真
ファイル 1 -ot ファイル 2	ファイル 1 がファイル 2 より古ければ真

論理結合　論理結合の仕方は以下

演算子	意味
! 条件	条件が偽であれば真
条件 1 -a 条件 2	条件 1 が真、かつ、条件 2 が真であれば真
条件 1 -o 条件 2	条件 1 が真、または、条件 2 が真であれば真

5.4.2 while

```
while [ 条件 ]
do
    処理
done
```

例 1:

```
#!/bin/bash
i=0
while [ $i -ne 10 ]
do
    i=‘expr $i + 1‘
    echo "${i}"
done
```

expr は四則演算 (+, –, ＼*, /) と剰余 (%) の計算ができるが、整数しか取り扱えない。かけ算は * と書きたくなるが、これは UNIX のワイルドカードと同じになってしまうので、それとの混同を避けるためバックスラッシュをつける必要がある。実数の計算をしたい場合は bc というコマンドを使う。
例 2:

```
#!/bin/bash
while read line
do
    echo ${line}
done < file
```

ファイルを読む場合はこうすると良い。

5.4.3 until

```
until [ 条件 ]
do
    処理
done
```

条件が偽である間、処理を繰り返し実行する。途中で中断してループを抜けたい場合は break、ループの先頭に戻りたい場合は continue を使う。

5.4.4 case

```
case 変数 in
      パターン 1) 処理;;
      パターン 2) 処理;;
      パターン 3 | パターン 4) 処理;;
      *) 処理;;
esac
```

;; は case 文を抜けるという意味。*) はいずれのパターンにもあてはまらない場合。
例

```
case "$var" in
  a* ) echo "a で始まる文字列" ;;
  ?b* ) echo "2 文字目が b の文字列" ;;
  [A-Z]* ) echo "大文字で始まる文字列" ;;
  [!xX]* ) echo "先頭が x ではない文字列" ;;
  * ) echo "上記のいずれでもない文字列" ;;
esac
```

5.4.5 for

```
for 変数 in 引数 1 引数 2 …
do
    処理
done
```

例 1:

```
#!/bin/bash
for filename in ‘ls -1 *.fits‘
do
    echo ${filename}
done
```

例 2:

```
#!/bin/bash
for char in A B C D E F G H I J K
do
    echo ${char}
done
```

例 3:

```
#!/bin/bash
for i in ‘seq 10‘
do
    echo ${i}
done
```

5.5 オプションの付け方

例えば、ls コマンドに-l とつける事によって出力が変化するように、オプションをつける事でスクリプトの動作を変えたい場合があります。その場合は getopts という組み込み関数を使って引数を調べます。
% cat sample02.sh

```
#!/bin/bash

help(){
    printf "Example: ./sample02.sh -a -b 10 -c 20\n"  1>&2
    printf "Options: -h : show this help\n"  1>&2
    printf "Options: -a : set variable1=1\n"  1>&2
    printf "Options: -b [argument]: assign the value to variable2\n"  1>&2
    printf "Options: -c [argument]: assign the value to variable3\n"  1>&2
}

#初期値の代入
variable1=0
variable2=0
variable3=0
while getopts "hab:c:" opt
do
    case ${opt} in
        h)help;
          exit 0;;
        a)variable1=1;;
        b)variable2=${OPTARG};;
        c)variable3=${OPTARG};;
    esac
done
echo "variable1=${variable1}"
echo "variable2=${variable2}"
echo "variable3=${variable3}"
```

注意すべきは、h と a というオプションは引数をとらないが、b と c は引数をとるようにしてある所です。引数をとるオプションには、getopts の所で:(コロン) がついている事に注意して下さい。変数$OPTARG はオプションに続く引数 (文字でも良い) が格納さる getopts 固有の変数で、名前を変える事はできません。sample02.sh を動かして遊んでみて下さい。ちなみに、シェルスクリプトで関数が使える事もこれでわかりますね？help() が関数になっています。
bash ./sample02.sh
bash ./sample02.sh -h
bash ./sample02.sh -a
bash ./sample02.sh -a -b 50 -c 100
等とやって、それぞれ出力が違う事を確かめて下さい。

5.6 標準出力は 1 番、標準エラー出力は 2 番

上記のオプションの例で、1> &2ってなんだ？と思った人は注意深くて偉いです。スクリプトやプログラムが何かを標準出力したり、エラーを出したりする場合、
command 1>file1 2>file2
とすると、標準出力を file1 に書き出し、標準エラー出力を file2 に書き出します。標準出力と標準エラー出力を両方まとめて出力するのが 1> &2 で
command > file > 1> &2
のように書きます。> と & の間にスペースは必要ないです (あるように見えますが)。

5.7 関数に引数を渡して、更に関数からの戻り値を使う場合

```
#!/bin/bash

funcion(){
  if [ $1 -eq 1 ]; then
    return 1
  else
    return 0
  fi
}

funcion 1

echo $?
```

こんな感じで引数を渡して、戻り値を参照する事もできる。

5.8 キーボードからの入力を受け付けたいとき

まあ、使わないかもしれませんが、こういう事もできるという事で、キーボードからの入力を受け付ける例を示します。このこのスクリプトは、キーボードからの入力を待ち、q が押されると終了します。

```
#!/bin/bash

while [ "${key}" != "q" ];
do
echo "Hit any key:"
stty cbreak
#stty raw
key=‘dd if=/dev/tty bs=1 count=1 2>/dev/null‘
#stty -raw
stty -cbreak
echo -e "\nYou hit \"${key}\" key"
done
```

6　シェルスクリプトから IRAF を使う方法

前置きが非常に長くなりましたが、本講習の本題であるシェルスクリプトから IRAF を使う方法についてこれから見て行きます。

なぜ、わざわざシェルスクリプトから IRAF を呼んで使うかというと、それはかっこいいから！ではなくて、ちゃんとした理由があります。例えば、データにパイプライン処理をする事を考えて下さい。パイプライン処理では、データに、とある決まった処理を施します。このパイプライン処理を施す作業をシェルスクリプト化しておくと、大量のデータをパイプライン処理する時に同じコマンドを何度も打つ必要が無く便利です。また、例えば script.sh というスクリプトを作っておいて cron で定時にスクリプトが動くような仕掛けを作る事もできます。更には、nohup script.sh &とやってスクリプトを走らせた後、自分はコンピューターからログアウトしてコーヒーを飲んだりテレビを見たりしていても、script.sh だけはせっせと動いてデータ解析をさせておけるようにできたりもします (nohup は UNIX のコマンドで、nohup command &としてコマンドを実行すると、ログアウト後もコマンドを続けさせることができる)。このように、作業をシェルスクリプト化しておくと「楽」ができるのです。

講習会のサンプルデータを扱う限りにおいては、シェルスクリプト化する御利益をあまり感じられないかもしれません。まあ、将来、沢山のデータを取り扱う時のためと思って知識をつけてください。

それでは、なにはともあれ、シェルスクリプトから IRAF を使ってみましょう。

6.1　なにはともあれ、動かしてみる

% cat sample03.sh

```
#!/bin/bash

LOG="sample03.log"

cp -rp ～/login.cl .

( (cl -old | tee ${LOG}) 2>&1 ) <<EOF
images
imutil
unlearn imstat
imstat (images="${1}", fields="image,npix,mean,stddev,min,max", lower=INDEF,
        upper=INDEF, nclip=0, lsigma=3., usigma=3., binwidth=0.1, format=yes, cache=no)
logout
EOF
```

sample03.sh を使ってみて下さい。使い方は
bash sample03.sh ./img/MT8191.fits
等です。引数に画像の名前をとります。
シェルスクリプトの中に << があると、そのあとの文字列 (例では EOF) を EOF マークとして覚え、その直後の行から次の EOF マークの直前の行までを、コマンドへ順次渡していきます。この場合、渡されるコマンドが cl である事がわかるでしょうか？実はこれが IRAF のご本尊なのです。拍子抜けするほど簡単です。cl の後に－ old とついているのがミソで、IRAF v2.14 からはデフォルトの IRAF シェルが ecl になったのですが、その ecl ではなく昔から使われていた cl を呼び出すようにするからです。ecl は cl にヒストリ機能等や、BS/DEL key のハンドリングを足した物という位置づけになっています。

cl -old にパイプをつけて、tee という UNIX コマンドに渡しています。これは IRAF の標準出力と標準エラーをログにとるためにやっています。ログの名前は LOG という変数で指定しています。

IRAF のタスクはパッケージ化されており、imstat はそのヘルプを見て一番上に表示されているように、images.imutil というパッケージの中に入っています。そこでこのスクリプトでは images をロードし、その次に imutil をロードし、その中の imstat を使うようになっています。実は images はロードしなくても良いです。login.cl の中をよーく見てみてください。デフォルトで images はロードされています。なので、images と書いた一行を削ってもちゃんと動きます。やってみてください。

これだけで、勘のいい人は全てを悟るかもしれません。そういう人は、自習して更なる高みへと進んで下さい。

6.2 オプションをつけてみましょう

先の例では、例えばシグマクリッピングのシグマが 3 で決めうちになっていたりします (nclip=0 だとシグマに何を入れようと結果は変わらないけれども)。シグマの値を変えたいと思った時等、毎回ソースコードを書き換えるのは面倒です。これをインタラクティブに変えたい場合はどうすれば良いかと言うと、前置きの所で学んだオプションをつければ良いのです。やってみましょう。
% cat sample04.sh

```
#!/bin/bash

LOG="sample04.log"
cp -rp ～/login.cl .

#初期値の代入
lsigma=3.0
usigma=3.0
lower="INDEF"
upper="INDEF"
filename="abc.fits"
format="yes"
nclip=0


help(){
    printf "Example: ./sample04.sh -f ./image.fits -a 2 -b 2 -l -999999 -u 999999\n" 1>&2
    printf "Options: -h : show this help\n" 1>&2
    printf "Options: -a [lsigma] : default ${lsigma}\n" 1>&2
    printf "Options: -b [usigma] : default ${usigma}\n" 1>&2
    printf "Options: -l [lower] : default ${lower}\n" 1>&2
    printf "Options: -u [upper] : default ${upper}\n" 1>&2
    printf "Options: -f : [filename]\n" 1>&2
    printf "Options: -n [nclip] : default ${nclip}\n" 1>&2
    printf "Options: -t : format no : default yes\n" 1>&2
}

while getopts "ha:b:l:u:f:n:t" opt
do
    case $opt in
        h)help;
            exit 0;;
        a)lsigma=$OPTARG;;
        b)usigma=${OPTARG};;
        l)lower=${OPTARG};;
        u)upper=${OPTARG};;
        f)filename=${OPTARG};;
        n)nclip=${OPTARG};;
        t)format="no";;
    esac
done

# エラー処理
# filename が明示されてない時、エラーメッセージを出して終了。
if [ ${filename} = "abc.fits" ]
```

```
then
    echo "You must input a filename with the -f option"
    exit 1
fi
# lower >= upper な状況の時は、エラーメッセージを出して終了。
if [ ${lower} != "INDEF" -a ${upper} != "INDEF" ]
then
    bool=‘echo "${upper} - ${lower} > 0" | bc -l‘
    if [ ${bool} -ne 1 ]
    then
        echo "[upper] must be larger than [lower]"
        exit 2
    fi
fi

echo "\n\n imstat ${filename} with : nclip=${nclip}, lower=${lower}, upper=${upper},
  lsigma=${lsigma}, usigma=${usigma} \n\n" | tee -a ${LOG} 2>&1

( (cl -old | tee -a ${LOG}) 2>&1 ) <<EOF
images
imutil
imstat (images="${filename}", fields="image,npix,mean,stddev,min,max", lower=${lower},
upper=${upper}, nclip=${nclip}, lsigma=${lsigma}, usigma=${usigma}, binwidth=0.1,
format=${format}, cache=no)
logout
EOF
```

sample04.sh を使ってみて下さい。使い方は、例えば、
bash ./sample04.sh -f ./img/MT8191.fits -n 10 -a 2 -b 2 -l -100 -u 500
です。
ソースコードの

```
bool=‘echo "${upper} - ${lower} > 0" | bc -l‘
if [ ${bool} -ne 1 ]
```

の部分が、なんだこりゃ?と思った人もいるかもしれません。これは、if の条件式では、整数しか比較できないためそれを克服するための僕なりの工夫です。UNIX の bc コマンドに式を評価させると、式が真なら 1 を、偽なら 0 を返します。これを使って式を評価させ、結果を bool という変数に代入し、その後 if で数値比較を行っています。繰り返しになりますが、if の条件文の数値比較は整数の比較しかできないので、実数の比較をするときはこのようにすると良いと思います。もちろん bc を使わない他のやり方もあると思いますが、僕はこれがシンプルだと思うのでこの方法を使っています。

6.3 自分で作った CL スクリプトをシェルスクリプトから呼び出して使う

自分で作った CL スクリプトも同じように使えます。
% cat sample05.sh

```
#!/bin/bash

LOG="sample05.log"
cp -rp ～/login.cl .

( (cl -old | tee -a ${LOG}) 2>&1 ) <<EOF
! echo ""
```

```
! echo ""
task iterstat=/フルパス/iterstat.cl
iterstat image="${1}"
logout
EOF
```

こんな感じです。! echo "" なんていうのが 2 回入っています。これを無くすと、task iterstat=/path/iterstat.cl という風にきちんとタスク登録のコマンドを書いているのにも関わらず、次の行で iterstat を使おうとすると、そんなタスクは見つからないと IRAF が怒って終わってしまいます。これは本当の超常現象なので、そんなもんだと諦めて、! echo ""等の、何の実害もない命令をタスク登録の前に 2 つ入れておきます。そうする事で正常に動きます。ここらへんが、IRAF のなんだそりゃ、な所なんですが、まあ、おおらかな気持ちで使いましょう。なんていっても IRAF は無料なんだし。
sample05.sh の中身を、自分の環境に合わせてフルパスの部分を書き換えて使ってみて下さい。使い方は
bash ./sample05.sh ./img/MT8191.fits
等です。

7 実習

7.1 その 1

sample05.sh を改造して、iterstat.cl のパラメーターである nsigrej,maxiter,print,verbose,lower,upper を、sample04.sh のようにオプション制御できるようにしたシェルスクリプトを作成してみてください。

7.2 その 2

img ディレクトリの中にある MT から始まる fits ファイルを次々と読みこみ、そのヘッダの DATA-TYP を調べ、次のような内容のテキストファイルを作成する CL スクリプトを作成して下さい。

```
% cat tekitou
MT8191.fits     OBJECT
MT8192.fits     OBJECT
MT8193.fits     OBJECT
MT8194.fits     OBJECT
MT8195.fits     OBJECT
MT8196.fits     OBJECT
MT8197.fits     OBJECT
MT8198.fits     OBJECT
MT8690.fits     DARK
MT8692.fits     DARK
MT8694.fits     DARK
MT8696.fits     DARK
MT8698.fits     DARK
MT8700.fits     DARK
MT8702.fits     DARK
MT8704.fits     DARK
MT8706.fits     DARK
MT8708.fits     DARK
MT8710.fits     DARK
MT8712.fits     DARK
MT8714.fits     DARK
```

7.3 その 3

その 2 で作った CL スクリプトを内部で呼び出し、その結果のテキストファイルから、UNIX コマンドを使って、DARK の画像だけを選び出し、ダーク画像を median combine し、スーパーダークを作るシェルスクリプトを作ってください。例：sed -n '/DARK/p' tekitou > dark.list とし、dark.list を median combine すれば良い。

7.4 その 4

その 3 で作ったシェルスクリプトに更に内容を書き足し、今度は OBJECT フレームのみを抜き出し、OBJECT フレーム毎にスーパーダークを引き算し、その後フラットで割るシェルスクリプトを作って下さい。

7.5 余裕でできちゃった人は

簡単すぎて余裕でできちゃった、時間が余ってしょうがない、なんていう人は、 第 2 回 IRAF 講習会の吉田 道利氏のテキストとサンプルシェルスクリプトをダウンロードし、「より進んだ実習」の内容をやると良い。そんなのすぐできちゃった、という人は何をしたら良いか僕に相談してください。そういう人は僕と共同研究しましょう。