The IRAF Manual for Beginners

いらっしゃいませ IRAF へ ようこそ♡

恐怖の大王襲来記念バージョンを元にした
日本女子大学版 (2002年6月1日 濱部　勝 改訂)

天文は好きだけど英語と計算機は大嫌いなすべての人と

ここで出会った学究の徒であり続けるであろう人々とその想い出

そして Lord of Lords、自称神であるあなたに

はじめに

このマニュアルは、

1. IRAF を本当に初めて使う人
2. IRAF のみならず、UNIX と触れ合い始めてまだ 4 日目ぐらいの人

を対象としています。“IRAF のすべてを知りたい” という人は、この辺でどうぞお引き取り下さい。“自分が IRAF を用いて何をしたいかすらわからないんです” …そんなあなたは、とりあえずこれを読んでみて下さい。きっと、“何がなんだか解んない” から、“解んないところは解った” と思える (はず) でしょうし、そうなってくれることそのものが本マニュアルの目的です。

使用上の注意

このマニュアルは著者 (及びその偉大かつ大恩ある先輩諸氏) の、全くの思い付きによって図らずも書かれ始めてしまったため、不備や思い違いもあるかと思いますが御了承下さい。なお本マニュアルを用いた結果生じた不都合について著者が負うべき責任はないものとします。

補足

本改訂版は、日本女子大学で IRAF を使用するにあたって、本書のオリジナルバージョンがそのまま適用できなかった部分を書き改めたものです。したがって、基本的な部分の変更はないと思います。

ただし、オリジナル版をなるべく削らず、付け足すことで改訂しようとしたので、若干無理を生じている箇所があるかも知れません。これについては意味が通じれば良かろうということで、御了承ください。

日本女子大学では、IRAF を Solaris と Linux の上で使用しており、FITS 画像ブラウザは、本書のオリジナル版で使われていた SAOimage の最新版、SAOimage ds9 を使用しています。また、本バージョンでは、Linux 版の IRAF の使用を想定していますので、Solaris 版とは異なるメッセージが表示されている部分もあります。

目 次

1　まずは言葉だよね

初心者を阻む最大の原因は、やはり言葉の壁ではないでしょうか。私は task (タスク) や package (パッケージ) という言葉を聞いた時に、“仕事?” 、 “包装?” …と、頭の中が “?” で埋まりました。ただただ日本語にすれば良い…という訳ではないことが伺える、微笑ましいエピソードですね。まずタスクとはなんでしょうか。タスクとは、 “ある目的を達成する、最小単位の命令” ぐらいに思って下さい。厳密な定義は、すくなくとも私は知りません。実際に動かす際には、コマンドとほぼ同義としておいてなんの不都合も生じないはずです。

次にパッケージについてお話ししましょう。
IRAF は、NOAO (アメリカ国立光学天文台) なるところで作成された、画像解析ソフトウエアです。いろいろなところで使われるようになるにつれ、より便利なタスクを作ってしまった人や、まったく新しいタスクを開発してしまった人が現れました。そこで、ある種の処理をするにあたり必要なタスクをまとめたり、似たような処理をするタスクをまとめたりしてみたくなりました。このようにして出来た “まとまり” を、パッケージと呼ぶことにします。パッケージの中にはタスクだけがあるかというとそうとばかりは限らず、パッケージの中にさらにパッケージが入っていたりします。ここでひるまず開け続けて下さい。らっきょうや、たまねぎを剥くよりは早く目的のタスクを発見すること間違いなしです。

今ここで、パッケージを “開ける” と書きました。正確には “ロードする” というようです。load のことです。IRAF に限らずアプリケーション (ここでは IRAF そのもののことで、一般的には特殊な目的を達成するために誰かが作ってくれたプログラム集大成のことぐらいに思って下さい) によっては、それを一回コンピュータに読んでもらわなくてはならない時があります。このような作業をロードといいます。ロードする対象は別にプログラムに限りません。データの入った補助記憶媒体 (こういう書き方は余計に訳がわからなくなる時がありますが、ようするに CD-R や DAT、MO などです。) を “読む” 時もロードというようです。ここで “読む” や “開く” はその時の気分で使い分けて下さい。

2　IRAF 始動の儀式

IRAF は UNIX 標準のアプリケーションではありません。従って、ある種の儀式というか、契約を結ばないといけません。これをメイクアイラフといいます。身も蓋もない名前ですね。(注意：メイクラブではありません。念のため。) 実際には、次のようにします。

まずは、適当なウインドウから

```
nursefan% xgterm &
```

と入力して[1] 、IRAF を使う時に標準的に使われる端末エミュレータ (あるいはターミナル) xgterm を起動してください。

xgterm がインストールされており、パス (path) が通っていれば[2]、新しいウインドウ (つまり端末あるいはターミナル) が開くはずです。
次に、

[1]最後の、& (and のマーク；アンパサンドと読みます) は省略できます。
付けた場合と付けない場合で何が違うかは確かめてみてください。

[2]UNIX のコマンド (プログラム) は色々なところに置かれているので、あらかじめシステムに、コマンドを起動する場合には、どことどこでコマンドを探しなさいということを教えておく必要があります。コマンドの在処が教えられている状態をパスが通っていると言います。

```
nursefan% mkiraf
```

[3]このように打ち込むと

```
-- creating a new uparm directory
Terminal types: xgterm,xterm,gterm,vt640,vt100,etc.
Enter terminal type:
```

と、何か terminal(ターミナル。コマンドを入力する窓のこと。kterm とか xterm などいろいろある。ちなみに kterm の “k” は “漢字” の “k” らしい。) を決めろみたいなことを言われます。個人的な趣味ですが、私は xgterm を奨めます[4]。いろいろなトラブルのもとになるので、kterm はやめておきましょう。ここでは、xgterm を選んだものとします。

```
Terminal types: xgterm,xterm,gterm,vt640,vt100,etc.
Enter terminal type: xgterm
A new LOGIN.CL file has been created in the current directory.
You may wish to review and edit this file to change the defaults.
```

どうやら儀式は終了したようです。その証はどこにあるのでしょうか。
ls で調べてみると、新しく “login.cl” なるファイルと “uparm” なるディレクトリが生成されているはずです。この二つが一体何なのかは、もう少し時間が経ってから徐々に知って下さい。あせってはいけません。すでにあなたは IRAF と契約をかわしてしまったのですから。

(補足) 儀式 **mkiraf は、同じディレクトリから出発して IRAF を使い始める場合、最初の 1 回だけ**行えば良く、毎回 mkiraf する必要はありません。

3 さあ、動かしてみよう

せっかく怪しい (?) 儀式をすませたのですから、その恩恵にあやかろうではありませんか。まずさっき設定したように、xgterm(他のターミナルを設定した方はそれに合わせたターミナル) を開き、login.cl と uparm がおいてあるディレクトリに自分がいることを確認します。もしなければ、自分が儀式を行ったディレクトリへ移動します。そこで、おもむろに “cl” と打ち込んでみて下さい。

```
nursefan% cl
```

すると嵐のように (マシンの処理速度によっては、冬眠直前の “かめ” のように) 何かの文字が駆け抜け、次のような状態があなたをまっています。

[3]ここで nursefan とあるのは気にしないで下さい。単なる架空のユーザー ID です。
この場合、`nursefan%`とある部分は、kterm や xgterm 等を起動した時に、いつも最初に表示されている、ユーザからの入力を促す文字列で、プロンプト (prompt) と言います。どのような文字列を表示させるかは、tcsh を使っている場合なら、`.tcshrc` 等といったファイルの中で指定できます。

[4]個人的な趣味ではなく、基本的に xgterm を使うべきと考えてください。IRAF のタスクのうちのあるものは xgterm の使用を前提として作られており、xgterm でないと使えない機能も出て来るからです (濱部注)

```
    NOAO PC-IRAF Revision 2.12EXPORT Thu May  2 19:48:29 MST 2002
    This is the EXPORT version of PC-IRAF V2.12 supporting most 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.  The following
    commands or packages are currently defined:

      dataio.     language.   obsolete.   softools.   system.
      dbms.       lists.      plot.       spiral.     tables.
      images.     noao.       proto.      stsdas.     utilities.
cl>
```

はい、おそろしいことに違う世界が開けてしまいました[5]。さっきまでは “%” (プロンプト。ターミナルがコマンドを受け付けてくれる状態だということを示しています。設定によっては、“>” だったり user ID が書いてあったりします。さっきの nursefan がそうです。) が出ていたのに、なくなってしまっています。代わりに “cl>” なんて書いてあります。

もしあなたが元の世界に帰りたいと思うなら、今すぐ “logout” といれて下さい。きっとなにもなかったかのように%を一つ返して、あなたは元の世界へ帰れます。つまり、 “cl>” があちらの世界のプロンプトなんですね。この時、上の英文をよく読めば困ることなど何一つないのでしょう。しかしここでは、あなたが私と同じくらい英語が苦手で、読む気力すらないくらいこの状況におびえているものとして話を進めます。よく見ると、

```
      dataio.     language.   obsolete.   softools.   system.
      dbms.       lists.      plot.       spiral.     tables.
      images.     noao.       proto.      stsdas.     utilities.
```

なんて書いてあります。これが噂のパッケージです。images. や、plot. のように、名前の後に “.” が付いています。これは、その名前が示すものがパッケージであることを示しています。 “.” が付いてないものはタスクです。では何をすればよいのでしょう。次ではもう少し具体的に話を進めます。

4 さっそく絵を表示しよう

IRAF は画像解析ソフトウエアです。したがって、人間さまには (おそらく) 読めないであろう画像ファイルを読んだり測ったり計算したりしてくれます。しかしプロセスはともかく、せめて元画像や結果画像だけでも見たい。当然ですよね。その前に重要なことですが、画像を表示する画像ウインドウは同じホストマシンで複数の人が同時に開くことはできません。誰か他の人が使ってないかどうか確かめてみて下さい (9.1.3 リダイレクトとパイプを参照して下さい)。もし誰か使っていたら、IRAF を知っている人にどうすればよいかを相談しましょう。確認がすんだら、いよいよ画像を表示させます。別のターミナルを立ち上げ、そこで

```
nursefan% ds9 &
```

[5]最初の数行は、IRAF のバージョンやプラットホーム (IRAF をその上で動かしている計算機の OS) によって異なります。
また、IRAF だけしかインストールされていない場合には、上の画面の、cl とある行の上の 3 行が若干異なります。例えば、spiral.、stsdas.、tables. などは、表示されません。

としてみて下さい。きっと、見たこともない窓が飛んで来るでしょう。ここではこの窓を画像ウインドウと名付けることにします。IRAF を立ち上げた xgterm に戻って、プロンプトの後に “display dev$pix 1” とします (この最後の 1 については 6.2 私の出会ったタスクたちを参照して下さい)。

```
cl> disp dev$pix 1
```

不思議なことに、なにやら渦巻いたものが表示されているはずです。この画像はサンプル画像ですから、あなたの記憶にない画像でも心配しないで下さい。決して惚けたわけではありません[6]。ところでパッケージやタスクを実際に使うためには、あらかじめそれを読み込んでおかなくてはならないことを先ほど述べました。しかしものによっては、IRAF の世界へやって来ると同時に読み込まれているようなタスクやパッケージもあります。例えばこの display などがそうです。

では、しばらく遊んでみましょう。まず、画像ウインドウの上部にコンソールパネルみたいなものが付いています。ここでは一つ一つを説明することはしません。壊れることはまずないので、やりたいようにパネルをいじくりまわして下さい。きっとあなたの想像以上のことは起きません。Color は色を変えそうで実際その通りですし、Zoom は画像の拡大や縮小なんかを司るところらしい…といった具合です。

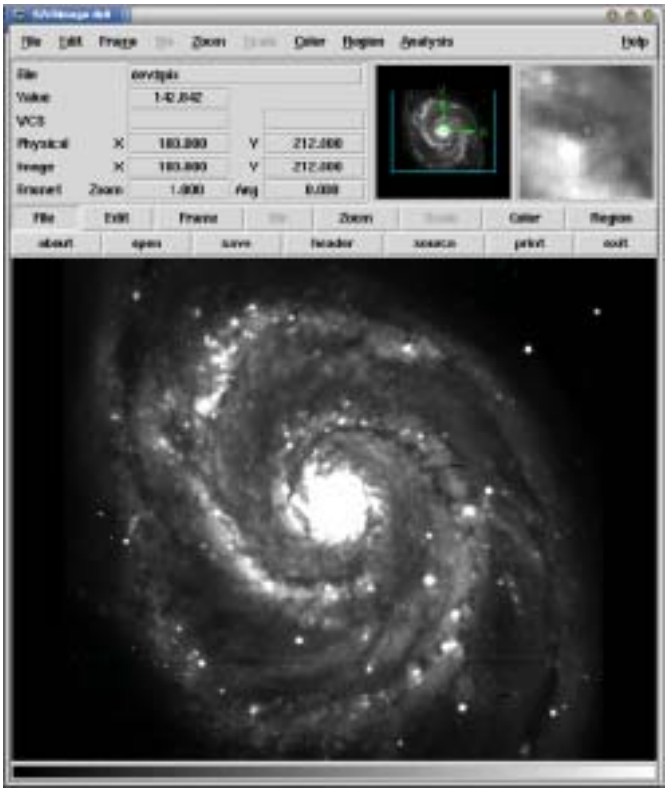

図 1: 怪しい渦巻き。M51 ともいう。

[6]実際にあなたのデータを処理する時は IRAF を立ち上げた後、そのデータがおいてあるディレクトリへ移動するか、データのありかを相対パス (→ 9.1.2 パス) で指定して下さい。例えば ディレクトリを移動した後 “NGC4762B1.fits” などというファイルを表示したければ、“display NGC4762B1.fits 1” として下さい。

ところでこのサンプル、しばらくみとれてしまうぐらい綺麗な絵ですね。どうですか、そろそろ感動が不安に変わるころでしょうか。“だから何?”という気持ちになってきましたね。

今まで簡単に、画像の解析と一言で言って来ましたが一体何をするのでしょう?
ある人は、得られた画像から明るさを測りたいかもしれません。別のある人は、スペクトルについて解析したいかもしれません。さらに、画像を足し合わせたり引いたりしたいかもしれません。このように、具体的に“何をしたい”という目的を持って初めてソフトウエアを活かせます。
“何を知りたいからデータを解析するのか”
これを明確にするのは計算機ではなく、あなたです。そのためのタスクがきっと IRAF にはあるでしょう。一つ具体的にやってみます。

明るさを測る

試みに、サンプル画像の明るさを測ってみましょう。明るさを測るタスクは“phot”といいます。ここではすでに、saoimage や ximtool などに画像= dev$pix が表示されているとします。
目的：明るさを測る。
タスク：phot

```
cl> phot
ERROR: task ‘phot’ not found
```

いきなり終わってしまいました。どうやらエラーらしいです。おまけに“そんなタスクは見つからない”などとぬかしています。計算機とはなんと不愉快なものなのでしょう。はっきりいって生意気です。
少し冷静さを取り戻して考えましょう。なぜこういうことになったのでしょうか。このような時一番考えられるのは、“そのタスクの入ったパッケージをロードしていない”ということです。(注意：IRAF のバージョンによっては含まれていないタスクがあったりします。詳しいことは熟練者に尋ねて下さい。) こういうことは、比較的よくあるアクシデントです。後の章で、あるタスクがどのパッケージに含まれるかの調べ方を軽くお話しします。とりあえず今はそれを知っているものとして、“ロード”していきます。

```
      dataio.     language.   obsolete.   softools.   system.
      dbms.       lists.      plot.       spiral.     tables.
      images.     noao.       proto.      stsdas.     utilities.
```

ロードするには、ここに出ているパッケージ名を打ち込んで下さい。ここでは、noao を開けます。すると、以下のようになります。

```
cl> noao
      artdata.      digiphot.     nobsolete.    onedspec.
      astcat.       focas.        nproto.       rv.
      astrometry.   imred.        observatory   surfphot.
      astutil.      mtlocal.      obsutil.      twodspec.
no>
```

ここでよく見ると、digiphot. のように“.”が付いているものと observatory のように付いていないものがあります。observatory は、タスクだということがわかりますね。ところで、まだ phot は見つかりません。次

はなんとなく名前の似ている digiphot を開いて見ましょう。

```
no> digiphot
      apphot.   daophot.  photcal.  ptools.
di> apphot
      aptest        findpars@     pconvert      polymark      psort
      center        fitpsf        pdump         polypars@     qphot
      centerpars@   fitsky        pexamine      polyphot      radprof
      daofind       fitskypars@   phot          prenumber     wphot
      datapars@     pcalc         photpars@     pselect
ap>
```

さて、掘って掘って掘りまくりですね。お蔭でやっと見つけました、phot でございます。 @なんかが目に入ってしまった方はいらっしゃいますか? お目が高いですね、ここではよくわからないから無視して下さい。ではこれをどうするか? とりあえずやってみましょう。

```
ap> phot
Input image(s):
```

また、思うように行きません。かなり悲しくなってきます。しかし落ち着いてよく読むと、 “どの画像だよ?” と聞いてるだけです。ここでは、表示した画像と同じ dev$pix を入れてみましょう。

```
ap> phot
Input image(s): dev$pix
```

すると、見慣れた dev$pix の画像上に小さな丸いカーソルが出来ます。このカーソルを測りたい場所に合わせて、スペースを叩いて下さい。

```
Warning: Graphics overlay not available for display device.
dev$pix    251.48   273.62  435.5197   14.840  err
```

こんな感じの文章が出て来たら、画像ウインドウの上で “q” と叩いてみて下さい。xgterm にごちゃごちゃした文字が浮かんできます。この時の “q” で画像ウインドウから xgterm にカーソルが移ります。

```
[Hit return to continue, n next image, q quit, w quit and  save parameters]
```

なんだかよくわからないけど、とりあえず “q” を叩いてみて下さい。phot を終えることになります。プロンプトが帰って来ると同時に、今あなたがいるディレクトリに pix.mag.1 というファイルが生成されているはずです。ここに測定 (測光という) データが目一杯入っています。あなたが欲しいデータはきっとこの中にあります。とてもとても膨大な (しかし機械にとっては大したことのない) 量のデータがありますので、自分がなにを欲しかったのか見失わないようにして下さい。

ここで、プロンプトのうしろにタスク名を入れた後、画像ファイル名を入れました。このように、コマンドやタスク名のあとに入れるものを引数 (ひきすう) と言います。例えば、“phot は、引数として画像ファイル名をとります” などという表現をします。IRAF に限らず、UNIX のコマンドにも引数をもつものがあり、無くては動かないコマンドやあってはいけないコマンドもあります。まるで英語の他動詞と自動詞みたいですね。

さて、賢明な皆さんは今ごろ大混乱でしょう。一体どの範囲をどうやって測ったんだ?…など。そうなんです。phot に限らず、ほとんど全てのタスクにはパラメータが付いてます。例えば phot の測光半径を決めるパラメータをいじれば、測光範囲を変えられます。パラメータを変えることによって、データに適した形でタスクを実行することができます。 次の章ではちょっとそんな話をしてみましょう。

5 phelp と epar

5.1 phelp

4 章を終えたところで、プロンプトは da> となっているはずです。ここで、最初のプロンプト cl> に戻りましょう。da>bye → di> bye → no>bye (p.24 参照) とします。すると、cl> に戻ります。パラメータを変える…と一言で言っても、どんなパラメータがあるかすらわからないですよね。さらに、それが一体何をするパラメータなのかもわからないですよね。こんな時に使うタスクに phelp というものがあります。当然引数をとり、ここでは “知りたいタスクの名前” を引数とします。まあ、なによりやってみましょう。例えば phot なら次のようにします。

```
ap> phelp phot
```

すると、例によってえらいことが始まります。

```
PHOT (May00)              noao.digiphot.apphot              PHOT (May00)



NAME
    phot -- do aperture photometry on a list of stars


USAGE
    phot image


PARAMETERS

    image
        The list of images containing the objects to be measured.

    skyfile = ""
        The list of text files containing the sky values, of the
        measured objects, one object per line with x, y, the sky value,
        sky sigma, sky skew, number of sky pixels and number of
        rejected sky pixels in columns one to seven respectively. The
        number of sky files must be zero, one, or equal to the number
        of input images. A skyfile value is only requested if
        fitskypars.salgorithm = "file" and if PHOT is run
        non-interactively.
…
```

さて (適当に) 読み解いてみましょう。最上段にある、

```
noao.digiphot.apphot
```

は、左から “noao. の中の digiphot. の中の apphot. の中にある” ということを示しています。こうして、どのパッケージを開けば良いか調べられますね。この先は “名前” とか、 “使い方” とか書いてあります。問題は、

```
PARAMETERS
```

なんて書いてあるやつです。英語いっぱいです。どうしましょう?
これでは自分が必要なパラメータがわかんないじゃないですか。
実のところ、ここはあきらめて読むか、誰か熟練している人に尋ねるしかありません。正直いって嫌になるくらい膨大なパラメータがある時もありますが頑張って下さい。もちろんしばらくの間は、全部のパラメータを理解する必要はないでしょう。ただなんとなく、 “お、これは必要っぽいなあ” ぐらいを感じられれば良いのではないでしょうか。パラメータの変え方は次の 5.2 でお話します。

さて、パラメータをどんどん眺めていくとこんなのが出て来ます。

```
DESCRIPTION

    PHOT computes accurate centers, sky values,  and  magnitudes  for  a
    list  of  objects in the IRAF image image whose coordinates are read
    from the text file coords or the image display  cursor,  and  writes
    the  computed x and y coordinates, sky values, and magnitudes to the
    text file output.
```

実際にはもっともっと続きます。もう私にとっては、単なる苦行でしかありません。誰がこんなん読むのでしょう? ここにはこのタスクの目的や使われ方、他のタスクとの関係などなどいろいろなことが書いてあります。言うまでもなく、タスクの結果を正しく理解するには、よく読むことが必要です。読めるなら読んだ方がいいです。読めない人はいろいろやってみて、使い方を体で憶えて下さい。

5.2 epar

phelp で調べた結果、必要なパラメータを見つけることが出来たとします。そこで今度はその値を変えてみましょう。epar というタスクです。引数はタスク名です。

```
cl> noao
      artdata.      digiphot.     nobsolete.    onedspec.
      astcat.       focas.        nproto.       rv.
      astrometry.   imred.        observatory   surfphot.
      astutil.      mtlocal.      obsutil.      twodspec.
no> digiphot
      apphot.   daophot.  photcal.  ptools.
di> apphot
      aptest        findpars@     pconvert      polymark      psort
      center        fitpsf        pdump         polypars@     qphot
      centerpars@   fitsky        pexamine      polyphot      radprof
      daofind       fitskypars@   phot          prenumber     wphot
      datapars@     pcalc         photpars@     pselect
ap> epar phot
```

するとこうなります。

```
                                I R A F
                  Image Reduction and Analysis Facility
PACKAGE = apphot
   TASK = phot

image   =              dev$pix  The input image(s)
skyfile =                       The input sky file(s)
(coords =                     ) The input coordinate files(s) (default: image.co
(output =              default) The output photometry file(s) (default: image.ma
(plotfil=                     ) The output plots metacode file
(datapar=                     ) Data dependent parameters
(centerp=                     ) Centering parameters
(fitskyp=                     ) Sky fitting parameters
(photpar=                     ) Photometry parameters
(interac=                  yes) Interactive mode ?
(radplot=                   no) Plot the radial profiles in interactive mode ?
(icomman=                     ) Image cursor: [x y wcs] key [cmd]
(gcomman=                     ) Graphics cursor: [x y wcs] key [cmd]
(wcsin  =             )_.wcsin) The input coordinate system (logical,tv,physical
(wcsout =            )_.wcsout) The output coordinate system (logical,tv,physica
(cache  =             )_.cache) Cache the input image pixels in memory ?
(verify =            )_.verify) Verify critical parameters in non-interactive mo
(update =            )_.update) Update critical parameters in non-interactive mo
(verbose=           )_.verbose) Print messages in non-interactive mode ?
(graphic=          )_.graphics) Graphics device
(display=           )_.display) Display device
(mode   =                  ql)

                                                              ESC-? for HELP
```

もう驚きませんね。ここでパラメータをいじっていく訳です。ここでは上下の矢印キーでパラメータを移動し、間違った場合は Delete キーで消します。では、実際にパラメータをいじってみます。例えば、

```
(radplot=                   no) Plot the radial profiles in interactive mode
```

を変えるには、カーソルをここまで持って来てから yes と書いてやると

```
(radplot=                  yes) Plot the radial profiles in interactive mode
```

こうなります[7]。
さて、“そんなの当り前じゃないか” という声が聞こえてきそうです。私もそう思います。このように、radplot=yes として phot を実行すると図 2 (p.18) のようなウインドウが現れます。
では、

```
(fitskyp=                     ) Sky fitting parameters
```

には何を書いたらいいんでしょう？　実は、このように何も書いてないところには何か他の隠れタスクが埋まってる時があります。これを呼びおこすには、“かっこ” の中の空白になっている部分で：(コロン) を打って

[7] radial profiles とは、測光範囲内の中心から外に向かって明るさ (つまりピクセルのカウント値、ピクセルとカウント値については、簡単にですが後述します。) がどのように変化していくかをグラフにしたものです。実際にやってみてください。

下さい。そうすると、ウインドウ下部に：が出てきます。(ミニバッファみたいなものだと思って下さい。)
そこで、：の後に e と打ち込みます。こんな具合です。

```
(fitskyp=                      ) Sky fitting parameters
(photpar=                      ) Photometry parameters
(interac=                   yes) Mode of use
(radplot=                   yes) Plot the radial profiles in interactive mode
(verify =                   yes) Verify critical parameters in non-interactive mo
(update =                    no) Update critical parameters in non-interactive mo
(verbose=                    no) Print messages in non-interactive mode
(graphic=              stdgraph) Graphics device
(display=              stdimage) Display device
(icomman=                      ) Image cursor: [x y wcs] key [cmd]
More
:e
```

すると、隠れタスクのパラメータが出て来ます。

```
                    Image Reduction and Analysis Facility
PACKAGE = daophot
   TASK = fitskypars

(salgori=              centroid) Sky fitting algorithm
(annulus=                  10.) Inner radius of sky annulus in scale units
(dannulu=                  10.) Width of sky annulus in scale units
(skyvalu=                   0.) User sky value
(smaxite=                   10) Maximum number of sky fitting iterations
(sloclip=                   0.) Lower clipping factor in percent
(shiclip=                   0.) Upper clipping factor in percent
(snrejec=                   50) Maximum number of sky fitting rejection iteratio
(sloreje=                   3.) Lower K-sigma rejection limit in sky sigma
(shireje=                   3.) Upper K-sigma rejection limit in sky sigma
(khist  =                   3.) Half width of histogram in sky sigma
(binsize=                  0.1) Binsize of histogram in sky sigma
(smooth =                    no) Boxcar smooth the histogram
(rgrow  =                   0.) Region growing radius in scale units
(mksky  =                    no) Mark sky annuli on the display
(mode   =                    ql)

                                                          ESC-? for HELP
```

ここで変えたいパラメータを変えていけばよいのです。

少々厄介な話ですが、似たようなタスクとタスクでパラメータを共有しているものがあります。そこでそれらをひとまとめにして、新たなタスクとしているようです。事実、

```
PACKAGE = apphot
   TASK = fitskypars
```

と、タスク名が fitskypars となっていることがわかります。もちろん phelp もありますから、このパラメータ一つ一つを調べることも出来ます[8]。パッケージを開けていった時に出て来た、@のついたタスクがこれだっ

[8]むしろ調べて下さい。

たんですね。私も今知りました。

パラメータの設定を終えたら、さっきと同じ要領で：を押して下さい。この時カーソルはどこにあっても構いません。また画面下部が入力受け付け状態になります。ここで設定変更を有効にしたければ wq と、無効にしたければ q! といれて下さい。

```
   ...
(rgrow  =                   0.) Region growing radius in scale units
(mksky  =                   no) Mark sky annuli on the display
(mode   =                   ql)


:wq
```

これで、一つ前のタスク (この場合 phot) のパラメータ入力画面に戻ります。さらにもう一度：を押して q または wq を打ち込んでやれば epar を終えます。これでパラメータの設定を変えることができました。あとは使うだけです。

5.3 本当になにかかわったの?

“今までの話では何がなんだか訳わからん、具体的に見せてよ。”
お怒りごもっともですね。ちょっとやってみましょう。例によってお題は phot です。4 章までで pix.mag.1 というファイルができているはずですね。では、パラメータを変えて測光してみましょう。5.2 でやったように photpars までたどり着いてください。ここで、

```
(apertur=                   3) List of aperture radii in scale units
```

を書き換えることにします。一体何をしているのでしょうか?4 章でできた pix.mag.1 は、apertur=3 でできたファイルでした。　また少しウンチク (インチキではない…と思う) をたれましょう。これは、アパーチャと呼ばれるパラメータです。口径とか直径とかいう意味があったと (自信ありませんが) 記憶しています。しかしここでは測光 “半径” を示しています。単位はピクセルです。ピクセルという言葉を初めて目にする方もいらっしゃいますよね?　これは、CCD カメラの画素のことです。ちょっと前の話ですが世の中がデジカメ、デジカメ大騒ぎだった頃、電車の吊革広告なんかのメーカーのうたい文句に
“史上初! 56 万画素を実現”
なんて書いてあって、そのとなりにライバルメーカーの広告で
“デジタルカメラとは思えない滑らかな画像。業界初の 100 万画素を達成!”
なんて書いてあったりして、何がなんだかわからなかった画素ってやつです。誤解をおそれずに大雑把な表現をすれば、CCD のチップとはフィルムの代わりに用いる感光素子のことで、小さな小さな方形素子を縦横に並べたものです。phot で得られる測光データはこの “小さな小さな方形素子がそれぞれ受光した量に応じた数字” で明るさを表現しています。ここで CCD を構成する “小さな小さな方形素子[9]” をピクセル (つまり画素)、“受光した量に応じた数字” をカウント値といいます。したがってここで測られる明るさは、 “ある x,y 座標から半径 r[ピクセル] 以内のカウント値の合計” と言えますね。さて測光結果を示してみましょう。xgterm などのターミナルで more や less や cat で覗いてみます。

[9]もしかしたら方形に限らないかもしれませんが、今のところ聞いたことはありません。

```
#K IRAF       = NOAO/IRAFV2.12EXPORT    version    %-23s
#K USER       = mhamabe                 name       %-23s
#K HOST       = hydra                   computer   %-23s
#K DATE       = 2002-06-01              yyy-mm-dd  %-23s
#K TIME       = 16:30:45                hh:mm:ss   %-23s
#K PACKAGE    = apphot                  name       %-23s
#K TASK       = phot                    name       %-23s
#
...
```

このファイルの由来が書いてあります。この後いろいろ難しいことが書いてありますが、最後まで見てみると、

```
...
#N IMAGE               XINIT     YINIT     ID    COORDS                LID    \
#U imagename           pixels    pixels    ##    filename              ##     \
#F %-23s               %-10.3f   %-10.3f   %-6d  %-23s                 %-6d
#
#N XCENTER    YCENTER    XSHIFT  YSHIFT  XERR    YERR           CIER CERROR   \
#U pixels     pixels     pixels  pixels  pixels  pixels         ##   cerrors  \
#F %-14.3f    %-11.3f    %-8.3f  %-8.3f  %-8.3f  %-15.3f        %-5d %-9s
#
#N MSKY           STDEV          SSKEW          NSKY   NSREJ     SIER SERROR   \
#U counts         counts         counts         npix   npix      ##   serrors  \
#F %-18.7g        %-15.7g        %-15.7g        %-7d   %-9d      %-5d %-9s
#
#N ITIME          XAIRMASS       IFILTER               OTIME                  \
#U timeunit       number         name                  timeunit               \
#F %-18.7g        %-15.7g        %-23s                 %-23s
#
#N RAPERT   SUM           AREA       FLUX          MAG    MERR   PIER PERROR   \
#U scale    counts        pixels     counts        mag    mag    ##   perrors  \
#F %-12.2f  %-14.7g       %-11.7g    %-14.7g       %-7.3f %-6.3f %-5d %-9s
#
pix                  258.000   259.000   1     nullfile              0      \
   257.754    258.928    -0.246  -0.072  0.005   0.006          0    NoError   \
   817.8819       155.6405       38.32154       940    6        0    NoError   \
   1.             INDEF          INDEF                 INDEF                  \
   3.00     115774.3      28.58008   92399.2       12.586 0.011 0    NoError
```

となっています。きっと全部重要な情報だとは思うのですが、やっぱりよく解りません[10]。忍耐強く眺めると、

```
pix                  258.000   259.000   1     nullfile              0      \
```

どうやらこのあたりが カーソルで指定した x, y 座標っぽいです。下にもちょっとずれた数字がありますが、きっと似たようなものでしょう[11]。そして最終行に

```
   3.00     115774.3      28.58008   92399.2       12.586 0.011 0    NoError
```

[10] 下の 5 行より上の部分の行頭に#のついたところは、最後の 5 行の数値の説明、つまりどんなパラメータがどういう形式で書かれているかが書いてあります。表示形式は Fortran の書式で書かれているので、詳しくは Fortran の教科書で調べてください

[11] 自分が選んだ光度中心と、計算機が選んだ光度中心の座標が書いてあります。

こうあります。きっと一番左がアパーチャサイズ、その次がカウントの合計でしょう。つまり、今欲しかったデータです。他の数字については割愛します。phelp を読んだり、熟練者を呼んだりして下さい。次に、アパーチャサイズを変えます。再び epar で photpars まで戻り、値を思い切って清水の舞台から飛び降りるつもりで 15 ぐらい (…スケールの小さい話ですね) にしてみましょう。

```
(apertur=                    15) List of aperture radii in scale units
```

このように apertur のパラメータを変えた後、

```
ap>phot dev$pix
```

と、phot を再び実行します。今度は図 3（p.18）のようなウインドウが現れます。また生成されたファイルを覗いてみます。ファイル名は pix.mag.2 のはずです。

```
pix                    258.000   259.000   1     nullfile              0      \
   257.754    258.928    -0.246  -0.072  0.005   0.006          0    NoError   \
   817.8819       155.6405       38.32154       940    6        0    NoError   \
   1.             INDEF          INDEF                  INDEF                  \
   15.00    853558.6      707.012    275306.3      11.400 0.022 0    NoError
```

期待した通り値が変わっています。こうして、パラメータを変えられるようになりました。他のタスクでも同様です。phelp を見ても解らなかったら、とりあえずいじりまくってみるというのも手段の一つですね。実際の振る舞いを見て解る時の方が多い人も少なくないでしょう。その結果もし、取り返しのつかないくらいいじくりまくってしまった場合には、unlearn というコマンドで、タスクのパラメータを元 (デフォールト) に戻すことができます。

ただし、今の例の場合は、実は 2 つのタスクが関わっているので、以下のようにする必要があります。

```
unlearn   phot
unlearn   photpars
```

6 私が出会った task たち

IRAF と出会ってからまだ一年余り、酸いも甘いも知るには短すぎる時間です。そんな私が知っている、本当にわずかなタスクを以下にまとめておきます。かなりの偏りがあるかとは思いますが御了承下さい。

6.1 タスクを探してくれるタスク〜refer〜

“なんだったけなあ、あのタスク”、“キーワードはわかってるんだけど、いいタスクないかしらん?”そんな時、非常に心強いタスクがあります。refer というタスクです。やってみましょう。 例えば私にとって未知の世界である、スペクトルに関するタスクを (恐いもの見たさで) 探してみましょう。 もう、どきどきです。

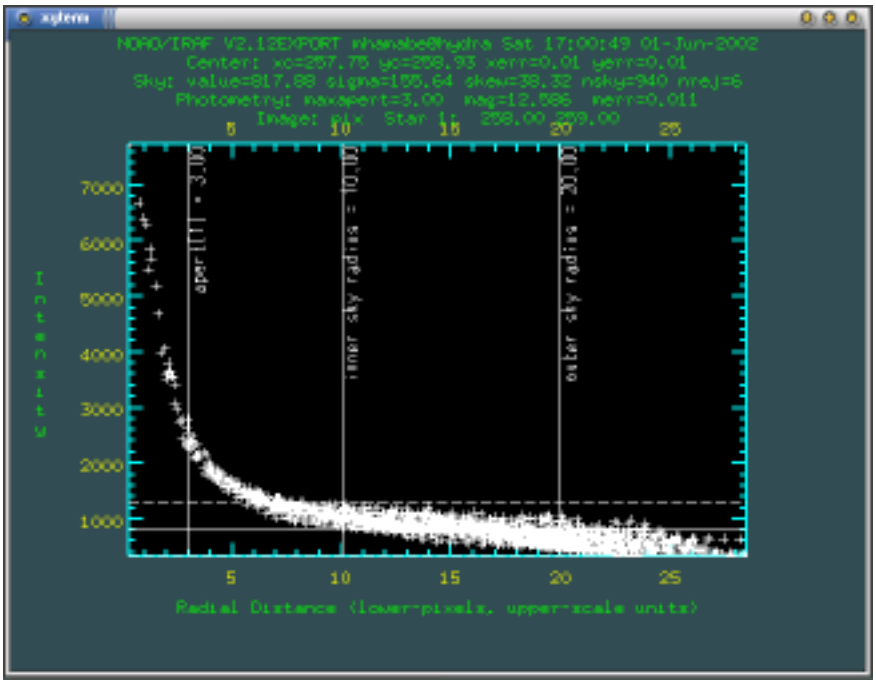

図 2: ラディアルプロファイル。パラメータをいじる前。

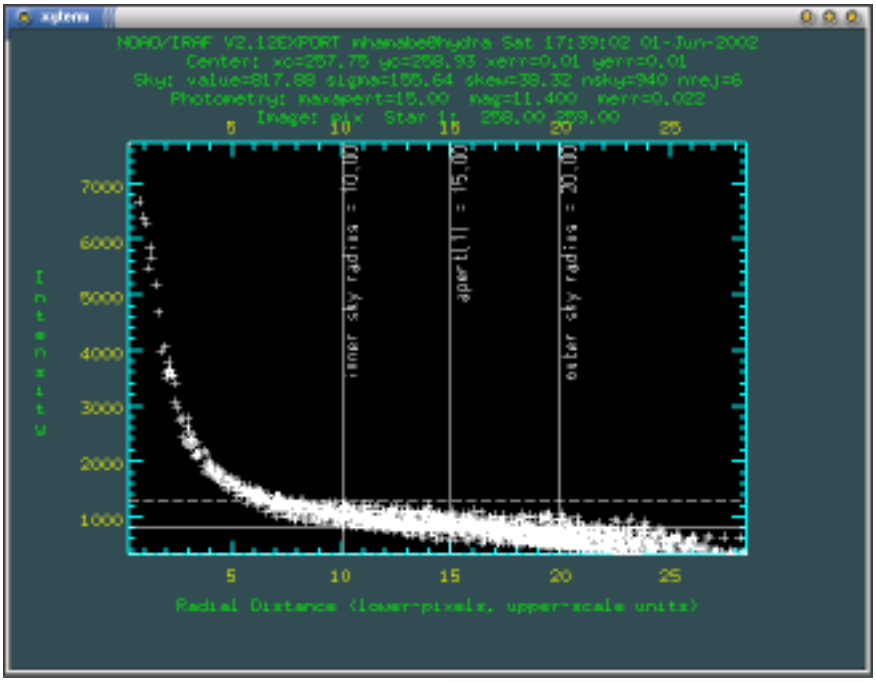

図 3: ラディアルプロファイル。パラメータをいじった後。アパーチャの値が変わっている。この図 2、図 3 については p.24 も参照。

```
cl> refer spectrum
searching the help database...
     ecidentify - Identify features in spectrum for dispersion solution
       identify - Identify features in spectrum for dispersion solution
         mkspec - Generate an artificial spectrum
         mkspec - Generate an artificial spectrum (obsolete)
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
         sarith - Spectrum arithmetic
        sinterp - Interpolate a table of x,y pairs to create a spectrum
          slist - List spectrum header parameters
          slist - List spectrum headers
```

このように、引数で指定したキーワードを含む全てのタスクを出してくれます。お気に召したものがあればphelp で確認して下さい。これを使いこなせるようになる頃には、きっと既に熟練者として名を馳せていることでしょう。

6.2 私の出会ったタスクたち

私が格闘し、時には振り回されながらもなんとか使うことができたタスクです。簡単に紹介します。もっと賢い使い方や重要なオプションがあったりするので、使いこなせてきたなあ…なんて思ってきたら必ず phelp でマニュアルを見て下さい。一意性がある限り、タスク名は省略できます。省略名の例を ( ) の中に示しておきます。

display (disp)

画像を表示します。なにはともあれ表示します。 引数に画像ファイル名をとります。ところで saoimage や ximtool は 4 枚の画像データを読み込んでそれを貯めておくことができます。そして、どの倉庫にデータを入れておくのかを指定しなくてはなりません。そこで、オプションとして数字を書いておきます。こんな具合です。

```
cl> disp sample1.fits 1
cl> disp sample2.fits 2
cl> disp sample3.fits 3
cl> disp sample4.fits 4
```

こうしてひとたび倉庫にデータを入れておけば、読み込まなくてもすぐに表示できますね。どの倉庫のデータを表示するのかを指定するには、ds9 (SAOimage) や ximtool のコントロールパネルで行います。画像の比較を行ったりする時は大変便利です。

files

リストを作ります。例えば 1999 年 1 月 29 日の観測データ 991029 というディレクトリの中身が次のようになっていたとします。

```
cl> ls
obj0.fits  obj4.fits  obj8.fits   bias2.fits  flat0.fits
obj1.fits  obj5.fits  obj9.fits   bias3.fits  flat1.fits
obj2.fits  obj6.fits  bias0.fits  bias4.fits  flat2.fits
obj3.fits  obj7.fits  bias1.fits  bias5.fits  flat3.fits
```

この時、まず名前が obj で始まるデータだけをリストにしたいとします。files をやってみましょう。

```
cl> files obj?.fits
obj0.fits
obj1.fits
obj2.fits
obj3.fits
obj4.fits
obj5.fits
obj6.fits
obj7.fits
obj8.fits
obj9.fits
```

ここで ? はワイルドカード (→ 9.1.4 ワイルドカード) です。ところで…、確かにリストになりました。でも、これでは今のところどうにもなりません。今私が欲しいのはリストのファイルです。リストのファイルがあれば、例えば imarith などのタスクで一括処理 (バッチ処理ともいう) を行うことができます。そこでこれをファイルに落してみましょう。9.1.3 の リダイレクトを使います。

```
cl> files obj?.fits > obj.list
```

こうすることによって、さっき画面に出力されたリストの結果が obj.list というファイルに詰め込まれているはずです。こうしてリストができました。次は新しく result.list というリストを作成してみましょう。新しく作るファイルの名前を、result?.fits とします。そうすると、先ほど作った obj.list の中に書いてあるファイルの名前をちょっと書き換えるだけで済みますね。リストの中身のファイル名を一部分だけ書き換えてみましょう。

```
cl> files %obj%result%?.fits > result.list
```

この、%ではさまれた部分は、 “obj を result に書き直せ” ということを意味します。result.list を見てみると (!については 22 ページ参照)、

```
cl> !more result.list
result0.fits
result1.fits
result2.fits
result3.fits
result4.fits
result5.fits
result6.fits
result7.fits
result8.fits
result9.fits
```

と、確かに新しいリストになっています。%ではさまれた部分はどんな文字列でも構いません。

ところで、リストについて軽く触れておきましょう。ここでいうリストとは、“1 行に付き 1 つだけファイルの名前を書き記したもの” です。それ以上でもそれ以下でもありません。それだけです。他に何も書きません。実際にタスクを実行する際は、リストに記したファイルがカレントディレクトリ (→ 9.1.1 ディレクトリ) にあるか、相対パス (→ 9.1.2 パス) で指定しないといけません[12]。例えば隣の “data” というディレクトリの中の “flat1.fits” というファイルを指定したければ、

```
../data/flat1.fits
```

とリストに書いて下さい。

imarith

画像の計算をします。これだけではなにがなんだかわかりませんね。どのような時に画像を計算したくなるのかについて、簡単に触れておきましょう。あなたが CCD を用いて取得した画像データがあるとします。そしてそこには何か目的のもの (つまりオブジェクト)、例えば dev$pix なら銀河が写っているとします。しかし何もしなければ、このフレームには熱雑音やバイアスなどがいっしょに写っていることになりますね。[13]そこでこの画像から、例えばバイアスだけが含まれるようなフレームを引きたいとします。オブジェクトフレームを obj.fits 、バイアスフレームを bias.fits、その結果を result.fits とする時、

```
cl> imarith obj.fits - bias.fits result.fits
```

としてください。result.fits というファイルが生成されているはずです。ちなみに演算子は、+ - * /の基本的な 4 つです。他にもありますが、詳しくは phelp をどうぞ。ここでは、obj.fits の各ピクセルのカウント値から bias.fits の各ピクセルのカウント値を引いています。他の演算でもおなじです。これは画像どうしの計算でしたが、画像と数を計算することもできます。例えばあるオブジェクトについて、露出した時間 (つまり積分時間) を一定にしないことには複数の画像を比較できません。そこで、露出時間が 600 秒のフレームを obj1.fits、240 秒のフレームを obj2.fits とする時、obj2.fits を obj1.fits にそろえたければ次のようにします。

```
cl> imarith obj2.fits * 2.5 result1.fits
```

こうして生成された result1.fits というフレームは、それぞれのカウント値が 2.5 倍されています。同じように、必要に応じて各種の演算を行ってみて下さい。また、imarith も含めたいくつかのタスクはリストを使うことができます。具体的なファイル名の代わりに、リストを書き込んだファイルをタスクに入れてみましょう。

```
cl> imarith @obj.list - bias.fits @result.list
```

として下さい。ここで “ほにゃらら.list” の前に “@” を付けておくことで、そのファイルの中に書いてあるファイルを演算処理してもらえます。ただし、リストの中に書いてあるファイルは、カレントディレクトリ (→ 9.1.1 ディレクトリ) に置いてあるか、リストの中で相対パス (→ 9.1.2 パス) を用いて指定する必要があります。詳しくは、files を見てみて下さい。

[12]もっとも、処理の結果生成されるファイルはある訳ないですが
[13]このあたりの詳しい解説は、CCD の本や観測技術にについての本を参照してください (→ 8 独り立ちの日)。

imcombine (imcomb)

複数の画像を足し合わせたデータの、平均をとったりメジアンをとったりします。各々のピクセルの平均を取るのかメジアンを取るのかは、epar で変更します。複数のフレームをコンバインすることでデータの質をあげていくことができます。その理屈については、もっと理屈っぽい書物を読んで下さい。

```
cl> imcomb obj?.fits result.fits
```

とすれば (?は 9.1.4 ワイルドカードを参照して下さい)、該当する画像ファイルのデータを足し合わせて、各々のピクセルの平均を取るか、メジアンを取るかしたものを result.fits に出力します。さらっと書き流しましたが、とっても重要なタスクです。

imstatistics (imstat)

画像の情報を教えてくれます。例えばカウントの最大、最小値や平均値、標準偏差などを教えてくれます。ここでは、これがどんな時に必要なのかは触れません。しかし、きっとこのタスクが必要な時がきっとくるでしょう。どうかその時は、このタスクのことを思い出してやって下さい。必ずやあなたを助けてくれることでしょう。

ehistory (e)

ヒストリ (履歴) 機能です。直前に実行したタスクから、いくつか前に実行したタスクを実行できます。やってみましょう。

```
cl> imarith obj2.fits * 2.5 result1.fits
cl> ehistory









imarith obj2.fits * 2.5 result1.fits
```

このように、ウインドウの一番下に直前に実行したタスクが出て来ます。もっと前に戻りたければ↑（矢印キー）を押し続けて下さい。書き直す事もできます。書き直したいところまで戻って del キーを押して下さい。

!

UNIX のコマンドの中には、IRAF の世界でそのまま使えるものもあります。しかし多くのものは使えません。そこで IRAF に、“これは UNIX のコマンドですよ”と教えてやることによって使えるようにしてみましょう。例えば、カレントディレクトリのテキストファイルをちょっと見てみたくなったとします。そんな時は

```
cl> !more logfile
```

とすると、logfile というファイルの中身が表示されます。もちろん他のコマンドでも O.K です。

bye

あなたが今開けているパッケージを片付けます。なんの話かといいますと、パッケージをロードしまくると、難しい話は抜きにしてコンピュータにとっても負荷がかかる時があります。そうなるとあなたもわたしもさあ大変! 1 つ 1 つの作業が遅くて遅くて…ということになってしまいます。そんな時は使ってないパッケージを閉じてしまいます。やってみましょう。

```
da> bye
di> bye
no> bye
cl>
```

プロンプトに注目すると、自分が今開けているパッケージがわかりますね。bye とすると 1 つ上のパッケージへ戻っていくのが見てとれます。こうしていらないパッケージをお片付けできます。ちなみに何かの手違いで、例えば noao を 3 回開いてしまったら必ず 2 回は bye して下さい。

?, ??

今自分がいるパッケージに含まれるタスクなりパッケージなどを教えてくれます。

```
cl> ?
      dataio.     language.   obsolete.   softools.   system.
      dbms.       lists.      plot.       spiral.     tables.
      images.     noao.       proto.      stsdas.     utilities.
```

こんな具合です。??は、全部のタスクを教えてくれます。

phot

もう今さらいいでしょうか? 詳しくは本文を読んでみて下さい。

pradprof

radial profile (ラディアルプロファイル) を出してくれます。ラディアルプロファイルについては、5.2 epar で触れた通りです。epar で、出力をリストにするかグラフにするかを指定できます。もちろん、どれくらいの範囲を出力するかも決められます。リストにした場合には、そのままだと画面にだらだら流れていってしまうので、リダイレクト (9.1.3 リダイレクトとパイプを参照してください) でファイルを指定してそこに書き込みましょう。epar でパラメータを、ちょっとだけ見てみます。

```
(list   =                 yes) List instead of plot profile?
```

今は、yes になっているので “プロットする代わりにリストにする” ことになっています。no にすると 18 ページの図 2 や 図 3 のような窓が飛んで来ます。さて、yes のままタスクを実行すると、

```
pl> pradprof dev$pix 255 265
# [dev$pix]   xcntr: 257.54   ycntr: 261.14
   4.85   2083.
   4.41   2741.
   4.17   3666.
   4.16   4062.
   4.39   3573.
   4.81   2773.
   4.73   1917.
…
```

このリストはこのままずーっとずーっと続きます。今は適当なところで切ってしまいました。 引数は、画像ファイル名、x 座標、y 座標です。後は機械が勝手に中心合わせしてくれます。 リダイレクトを使えば、

```
pl> pradprof dev$pix 255 265 > dev$pix.list
```

として、dev$pix.list なるファイルにリストを書いてくれます。

normalize

画像ファイルを規格化してくれます。パラメータを何もいじらなければ、各ピクセルのカウント値の平均が 1 になるようにしてくれます。ただしファイルを上書きしてしまうので、注意が必要です。1 度コピーしておいた方がよいでしょう。

imcopy

身も蓋もなく、画像ファイルをコピーしてくれます。使い方は、

```
cl> imcopy before after
```

で、before と同じ内容の画像ファイルを after という名前で作成してくれます。

imcalc

パッケージ stsdas を持っていれば使えます。画像を計算してくれるのですが、二項演算だけでなく何項でも演算できます。最初のうちは知らなくてもいいのではないでしょうか? ただ、そんなタスクもあるんだということは知っておいて損はないでしょう。

imdelete(imdel)

画像ファイルの消去をしてくれます。IRAF では rm コマンドは使えません。
画像ファイル以外の一般のファイルを消去する場合には、delete (del) が使えます。
FITS 画像の場合は、imdelete も delete もあまり変わらないのですが、使い分けることをお奨めします。

imrename(imren)

ファイルの名前を変えてくれます。

```
cl> imren before after
```

before に古いファイル名、after に新しいファイル名を入れて下さい。

7 とっても簡単なルーチンワーク

慣れてくると、同じ処理を対象を変えて繰り返したり、複数のタスクを順番に何度も実行したりすることが大変苦痛に感じられる時があります[14]。なんとかならんもんでしょうか? そこでこういったルーチンワーク (繰り返し処理) をあらかじめ何かのファイルに書いておいて、それを実行させてみます。例えば取得した画像フレームを、処理に入る前に 1 枚 1 枚目で確認していきたいとします。そこでこんなファイルを作成します。

```
display sample1.fits 1
!sleep 10
display sample2.fits 1
!sleep 10
display sample3.fits 1
!sleep 10
display sample4.fits 1
!sleep 10
display sample5.fits 1
!sleep 10
display sample6.fits 1
```

これは、 “sample1.fits を表示しろ” の後 “そのまま 10 秒眠れ” を全てのフレームに関してただ書き並べただけです。!については、6.2 私の出会ったタスクたちを参照して下さい。さて、こうやって書き並べたファイルを実行させればいいわけです。そのためには久しぶりに新たな儀式を行わなければなりません。このファイル (そうですね、名前を check.cl とでもしましょうか) をタスクにしてしまう儀式は以下の通りです。

```
cl> task $watch = check.cl
cl>
```

儀式はあっけなく終了です。だまされたと思って、watch をやらせてみましょう。

```
cl> watch
```

画像が順番に出てくるはずです。実際にあなたのデータで試してみて下さい。ここで watch が 新しく定義されたタスク名です。1 度 IRAF から logout してしまうと、この定義は無効になります。再定義して下さい。

実際には、ある程度順序だったことを書き込んでおけばその通りに動きます。例えば複数枚あるバイアスフレームをコンバインして、その結果をオブジェクトフレームから引いて、フラットでオブジェクトを割って… といった具合です。最初にたった 1 つのタスクを定義してしまえば、あとはまとめて自動でやってくれます。ただし、記述に間違いがあった場合にはそこでタスクが終了します。

デバッグ (プログラムやスクリプトといった、この手の記述を修正し役に立つものに直すこと) はちゃんとやりましょう。6.2 でも簡単に触れましたが、オブジェクトとかバイアスなどといった言葉は然るべき経験 (観測

[14] そしてそれを苦痛と感じた時、あなたはすでに計算機の世界に呑まれ始めています。

や処理) を通じて学んで下さい[15]。

いろいろ出来るようになった後にここを読むと良いかもしれません。

8　独り立ちの日〜一次処理、またの名をリダクション〜

さあ、このマニュアルの役目もそろそろ終りに近付いています。きっとここまで読み進めてきたみなさんは、なんとなく IRAF を使えるようにはなってきたけれど、なんとなく何をすれば良いかわからない… そんな風な印象をもっているのではないでしょうか?　そこで、なにをするにしても絶対必要な一次処理 (リダクション[16]) について簡単に触れ、本マニュアルの使命を終えようと思います。

8.1　やっぱり言葉からだよ、うん

一次処理ってなんざんしょ?　これまでそれとなく触れてきましたが、もう少しまじめに説明してみます (あんまり自信ありませんが)。

8.1.1　一次処理

冷却 CCD を用いた撮像では、暗電流やバイアスを引いたり、フラットで割ってピクセルごとの感度ムラを補正したりします。この処理を一次処理、またの名をリダクションと呼びます。なんだか急に難しい話になってしまいました。順を追って説明 (した振りをして軽く流) します。

8.1.2　ダークとバイアス

難しい話は抜きにして.... と言いたいところですが、ちょっとだけ我慢してください。CCD の仕組みは、とってもとっても大雑把に言って光電効果です（かなり違うけど)。しかし、光が CCD にあたらなくても、温度が高いと CCD チップの中で余計な電荷が生じてしまいます。このような電荷を**暗電流** (=Dark current) といいます。そこで、天体を撮像したのと同じ時間露出して暗電流だけを撮像し（このようなフレームを**ダークフレーム**といいます)、それを天体が写っているフレーム（**オブジェクトフレーム**といいます）から引き去ることによって暗電流分の電荷をキャンセルしたものと見なします。また、たとえ露出時間が 0 秒でも、CCD のカウントはゼロにはなっていません。これは CCD から電荷を読み出すために必要なものだとここでは簡単に考えてください。というわけで CCD にあたった光の量を求めるためには露出時間 0 秒のフレーム、バイアスフレームをオブジェクトフレームから引き去る必要があります[17][18]。

8.1.3　フラット

CCD チップを構成するピクセルは、各々が必ずしも同じ感度をもっているとはかぎりません。あるピクセルは 100 個の光子で 20 個の電子を吐き出すかもしれませんし、他のピクセルは 4 個しか吐き出さないかもしれません。ではどうしましょう? 理想的には、一定の強さの光をチップに均等に当てたものを撮像します。これを**フラットフレーム**といいます。このフレーム[19]を、まずこのフレーム自身のカウントの平均値で割ってや

[15]次のセクションで簡単にまとめておきますが。
[16]reduction　日本語では「整約処理」と言いますが、これは天文学用語で、例えば物理分野の人には通じない言葉です
[17]詳しいことは巻末参考文献 6 やそれに類する本、半導体工学などの書物を見て下さい。
[18]8.1.2 は監修者が大幅に書き換えました。
[19]以下の処理の前にフラットからもバイアスやダークを引いておかなければいけません

ります。そうすることで、各々のピクセルのカウント値が “1 ±ほんのちょっぴり” になりますね[20]。

このフレームでオブジェクトフレームを割ってやると、それぞれのピクセルごとの感度のムラをキャンセルできます。もう訳がわかりませんね。後で具体的にやりますから、わからない方も心配なさらずに[21]。

8.2 独り立ちの日

8.2.1 1 歩 1 歩確実に…

いよいよ独り立ちです。今あなたは次のようなデータを持っているとします。

```
cl> ls
bias0.fits  bias5.fits  flat4.fits  obj0.fits  obj5.fits
bias1.fits  flat0.fits  flat5.fits  obj1.fits  obj6.fits
bias2.fits  flat1.fits  flat6.fits  obj2.fits  obj7.fits
bias3.fits  flat2.fits  flat7.fits  obj3.fits  obj8.fits
bias4.fits  flat3.fits  flat8.fits  obj4.fits  obj9.fits
```

まず、バイアスの平均値を作りましょう。リストを作ってみましょうか。その後 imcombine を行います。パラメータは average にしておいたとします。

```
cl> files bias?.fits > bias.list
cl> imcombine @bias.list bias.fits
```

こうして、複数のバイアスフレームをコンバインして、データの質を高めたバイアスフレームが 1 枚出来ました。

さて、これを他の全てのフレームから引きます。またリストを作りましょうか。その後引き算します。

```
cl> files obj?.fits > obj.list
cl> files flat?.fits > flat.list
cl> files %obj%obj_b%?.fits > obj_b.fits
cl> files %flat%flat_b%?.fits > flat_b.fits
cl> imarith @obj.list - bias.fits @obj_b.list
cl> imarith @flat.list - bias.fits @flat_b.fits
```

ここまでで、オブジェクトからもフラットからもバイアスを引くことが出来ました。その結果は、それぞれ obj_b.list と flat_b.fits で定められた名前で生成されているはずです。さて、フラットを 1 枚にまとめてみましょう。今度はちょっと枚数が多いので、メジアンをとってみましょうか。いちいち epar をするのも面倒なので、一時的にパラメータを変更します[22]。

次のようにしてみて下さい。

```
cl> imcombine @flat_b.fits flat.fits combine=median
```

[20] もしピクセルごとに感度ムラがなければ、理想的な光を受けた各ピクセルは同じカウント値を示し、その平均値で割ると全てのピクセルのカウントは 1 になるはずです。つまり、“±ほんのちょっぴり” が感度ムラを表している訳です。

[21] これでわかったあなたは素晴らしい!

[22] 次の 8.2.2 でも威力を発揮します。

さて、このままオブジェクトを割るとえらいことになります。フラットのカウント値にもよりますが、オブジェクトフレームのカウント値が異常に小さくなってしまいます。そこで、このフレームが感度ムラのみを表すような規格化をしましょう[23]。そのために、このフレームのカウント値を見てみます。

```
cl> imstat flat.fits
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
           flat.fits   4190209    24224.     2179.        0.    65469.
```

カウントの平均値は 24224 だということがわかりました。これで割ってみましょう。

```
cl> imarith flat.fits / 24224 nflat.fits
cl> imstat nflat.fits
#               IMAGE      NPIX      MEAN    STDDEV       MIN       MAX
          nflat.fits   4190209        1.   0.08995        0.     2.703
```

ようやく、規格化されたフラットができました。これでオブジェクトを割ってやれば、“ノイズを引いて、感度ムラを補正したオブジェクトフレーム”を得られますね。おっとっと、最後に得られるフレームの名前のリストを作っておきましょう。

```
cl> files %obj%reduc%?.fits > reduc.list
cl> imarith @obj_b.list / nflat.fits @reduc.list
```

こうして、リダクションを終えました。きれいな画像が出来ているはずです。

8.2.2 そして、飛躍

次に、これを一気にやってしまいましょう。そんなタスクを作ってしまえばいいのです。問題は、imstat のところです。これればかりは計算機にまかせられません[24]。そこで、自動的に規格化してもらいましょう。normalize を使います。normalize は恐いので、一応コピーもとっておきましょう。

さて、ファイルに一連の処理を書き連ねていきます。ファイル名は、例えば reduction.cl なんてどうでしょう。今あなたが何をしようとしているのかを書いて、コメントアウト (ひとり言。計算機が無視してくれるように書き込んでおくこと) しておくとさらに良いでしょう。#を行の頭に付ければコメントアウトできます。

[23] ケースバイケースなのですが、フラット 1 枚 1 枚を規格化してからコンバインした方がデータの質が高いという場合もあります。
[24] まかせる方法もありますが、このマニュアルのレベルを遥かに超越するのでここではまかせられないことにします。

```
files bias?.fits > bias.list
files obj?.fits > obj.list
files flat?.fits > flat.list
files %obj%obj_b%?.fits > obj_b.fits
files %flat%flat_b%?.fits > flat_b.fits
files %obj%reduc%?.fits > reduc.list

#リストの作成

imcombine @bias.list bias.fits combine=average

#バイアスフレームをコンバイン

imarith @obj.list - bias.fits @obj_b.list
imarith @flat.list - bias.fits @flat_b.fits

#バイアスを引く

imcombine @flat_b.fits flat.fits combine=median

#フラットをコンバイン

imcopy flat.fits cflat.fits

#予備のコピー

normalize cflat.fits

#cflat.fits の規格化

imarith @obj_b.list / cflat.fits @reduc.list
```

これをタスクにしてしまいましょう。

```
cl> task $nursefan = reduction.cl
```

完成です。じゃあ、タスク nursefan は実行できるか…というとこのままでは間違いなく止まるでしょう。何故でしょうか? normalize がロードされていません。タスク nursefan を実行させる前にロードしておかなくてはなりません[25]。

[25] この例に限らず、スクリプトの中で用いるタスクは全てあらかじめロードしておかなくてはなりません。

```
cl> noao
      artdata.      digiphot.     mtlocal.      observatory   surfphot.
      astrometry.   focas.        nobsolete.    onedspec.     twodspec.
      astutil.      imred.        nproto.       rv.
no> imred
      argus.      ctioslit.   generic.    irred.      kpnoslit.
      bias.       dtoi.       hydra.      irs.        specred.
      ccdred.     echelle.    iids.       kpnocoude.  vtel.
im> generic.
      background  darksub     flatten     normflat
      cosmicrays  flat1d      normalize
ge>
```

これで O.K です。後は、nursefan を実行するだけです。きっと大きな感動が待っていることでしょう!

Coffee Break

これで IRAF に関してのお話しは終りです。お疲れ様でした。少ししょうもない話をしましょう。
私は本当に計算機が嫌いでした。今でも決して好きではありません。大学に入学し強制的にワークステーションに触らせられるまでは、ファ○リーコンピュータ以外のコンピュータと面識がなく、最後に触ったキーボードはフ○ミリーベーシック (こんなのご存知ですか?) という有り様でした。ところが、そんな私に大転機が訪れます。どういう訳だか、私は国立天文台にもぐり込んで卒業研究をすることになりました。自ら望んだこととはいえ、私にとってそれはそれは恐ろしい世界でした。なにをやるにしても計算機に頭を下げなくてはならないのです。しかも、やらないことには卒業できなくなってしまいます。ああ、なんたる悲劇!! 正直いって人生で 6 番目ぐらいの危機でした。七夕の短冊に “コンピュータが僕のいうことをききますように” って書いたぐらいです[a]。
それでも時間は無情にも流れていきます。止まってくれません。学会は目前です。もう藁をも掴むつもりで掴んだものは、計算機との親密な関係を雰囲気で醸し出す 2 人の先輩でした。彼らは偉大でした。あいかわらず全くプログラミング言語を 1 つも知らない私に、プログラムを書いてくれたりします。もう仏のようです。右も左もわからない私に、“まあ、マニュアル読んで” の一言を言い渡しいなくなってしまいます。もう鬼にしか見えません。時計を見ながら泣きたくなる時もあります。そうこうしているうちに、気が付けばずいぶん計算機に慣れてしまっていました。驚いたのは私です。“一体どうしちゃったのかしらん?” そしてこのマニュアルが生まれました。

“この 1 年間に身につけたことを、去年の僕にプレゼントしよう”

こんなコンセプトのもと、こんなマニュアルを思い付きで書き始めてしまいました。計算機嫌いのみなさまは、IRAF という暗闇を歩く時きっと不安を感じることでしょう。そんな方々にとってこのマニュアルが、“懐中電灯がみつかるまでのロウソク” になってくれたらと思っています。

[a] 実話。その時私の愛すべき友人、自称神である S は言いました。“…じゃあ、ちゃんと言うことをきく言葉を使おうよ…” 私はあの屈辱を忘れない。

9　ちょっとだけ UNIX

9.1　はじめの一歩…から三歩目ぐらいまでのお話

UNIX に関しては既に名著がたくさん出版されていますし、何かを書けるほど立派なことを知っている訳ではないのですが簡単に触れておきます。

9.1.1　ディレクトリ

私が最初に躓いた考え方です。正直なところ現在でも正確に認識している自信がありません。

よく言われる表現は “タンスの引出し” です。なんでも突っ込んでおけます。例えば文章を書いたファイルやプログラム、数字を並べただけのリストなどを、いろいろ無造作に置いてあるおもちゃ箱みたいなものですね。この箱を “ディレクトリ” と呼びます。特にあなたが login した直後に入っている “引出し” のことを “ホームディレクトリ” と呼びます。
ただしこのまま放っておくと、なにがなんだかわからなくなってしまいますよね。複数の作業を同時に行っている時などは致命的に混乱が生じる時があります。そこでこの “引出し” を整理してみましょう。整理の仕方は人によって好みがありますが、作業ごとに作成されたファイルをまとめて一つの新しい引出しに入れてみます。こうすると “引出しの中に新しい引出し” ができますね。こうして出来た “引出し” に作成されたファイルを突っ込んでおくと少しさっぱりします。

実際にやってみましょう。今、ホームディレクトリにいるとします。ここで “ls” と打つと、 このディレクトリの中のファイルをモニターに表示してくれます。

```
nursefan% ls
list1.list  list3.list  sample2.f   sample4.pl  text2
list2.list  sample1.c   sample3.pl  text1       text3
```

ここでさっきの例に従い、作業 1、作業 2、作業 3 で使ってるファイルをまとめてみます。新しくディレクトリを作成してから、ls で見てみましょう。

```
nursefan% mkdir work1
nursefan% ls
list1.list  list3.list  sample2.f   sample4.pl  text2       work1
list2.list  sample1.c   sample3.pl  text1       text3
```

work1 というディレクトリが作成されています。それではさっそく放りこんでみます。

```
nursefan% mv list1.list work1
nursefan% mv sample1.c work1
nursefan% mv text1 work1
nursefan% ls
list2.list  sample2.f   sample4.pl  text3
list3.list  sample3.pl  text2       work1
```

list1.list などがなくなってしまいました。では、今いるディレクトリから work1 に “入って” 調べてみましょう。ディレクトリを移動するコマンドは cd です。

```
nursefan% cd work1
work1% ls
list1.list  sample1.c   text1
```

ちゃんとありましたね。このようにしてまとめていくと、だいぶディレクトリが整理されます[26]。ところで言葉の話ですが、あなたが “現在いるディレクトリ” を “カレントディレクトリ” といいます。直前の例では、カレントディレクトリ=work1 です。この先いろいろなコマンドを憶えて、それらをそれなりに使いこんでいくと “カレントディレクトリ” と “その一つ上のディレクトリ” 、そしてホームディレクトリというのが結構頻繁に必要になってきます。“で?” と言われてしまいそうですね。もう少しの辛抱です。そこでなにかのコマンドを使う際に、引数としてこれらのディレクトリを取る時、カレントディレクトリを “.” 、その一つ上のディレクトリを “..” 、ホームディレクトリを “~” で省略して表現します。これは憶えておくと便利です。

9.1.2 パス

“パス” というのはディレクトリのありかを示す “道筋” だと思って下さい。当然道にはスタートとゴールがあります。

スタート = あなたが login しているマシンの一番上部にあるディレクトリ (root: ルートディレクトリ)
ゴール = カレントディレクトリ

とした時の “道筋” を “絶対パス” といいます。これに対して “相対パス” とは、カレントディレクトリから他のディレクトリ (イメージしづらいとは思いますが、例えば mv や cp などを行う時に指定するディレクトリをイメージして下さい。) への “道筋” のことをいいます。また、このような絶対パスや相対パスでディレクトリを指定する時は、ディレクトリとディレクトリの間を / で区切ります。
詳しくは UNIX の本を参照して下さい。

9.1.3 リダイレクトとパイプ

例えば ls とやると、モニターにカレントディレクトリのファイルが表示されます。このコマンドの振る舞いを、 “カレントディレクトリの中のファイルを調べて” “その結果をモニターに表示する” という 2 つの部分に分けて考えてみましょう。この時後の部分をモニターではなく、なにかのファイルに書き込みたいとします[27]。そこで、出力先をモニターからファイルに変えてみます。

```
nursefan% ls
list2.list  sample2.f   sample4.pl  text3
list3.list  sample3.pl  text2       work1
nursefan% ls > test
nursefan%
```

[26] ここで出て来たコマンドの使い方は簡単に後述します。
[27] なんでそんなことをしたいのかは問わないで下さい。私も知りません。

ここで “>” は、 “左の結果を右に出す” ということを意味していると思って下さい。こういう処理を難しい言葉でいうと “リダイレクト処理” といいます。実際に、more かなんかで test を覗いてみると

```
nursefan% more test
list2.list
list3.list
sample2.f
sample3.pl
sample4.pl
test
text2
text3
work1
```

となって、リストになってファイルに書かれています。

似たような処理をもう一つ。今度は、あるコマンドの結果を他のコマンドの入力にしたいと言う場合を考えてみます。なんだかわからないからやってみましょう。ネタは、プロセス (…わかんないですよね、こんなん書いても。ようするに、今計算機が行っている 1 つ 1 つの仕事というか命令というかそのようなもの) を調べるコマンドである ps と、ある文字列 (そのまんま。文字の羅列のこと) を検索する grep です。

```
nursefan% ps -aux | grep saoimage
halffishman   16861  0.0  1.0  2068 2540  p1  I    12:39AM    0:00.10 saoimage
nursefan      16902  1.4  1.0  2088 2580  p4  S+   12:40AM    0:00.09 saoimage
```

これは “ps -aux の膨大な量の結果を、saoimage という “文字列” で grep にかけた” ということです (-aux とありますが、この “-なんちゃら” はオプションといい、コマンドにオプションをつけない時とはちょっと違う結果を出してくれるものです)。その結果、私以外に halffishman なる人が saoimage を使っていることが判ります。ここで | は、 “左のコマンドの結果を右のコマンドの入力とする” ということを意味していると思って下さい。こういう処理を難しい言葉でいうと “パイプ処理” といいます。

これらのことをより正確に理解するためには、標準入力と標準出力、標準エラー出力などを学ばないといけません。UNIX の本を参照して下さい。

9.1.4 ワイルドカード

ある種のコマンドで、引数に似たような名前のファイル名を使いたいとします。同じ処理を一括して行う方法はないものでしょうか? こんな時に使えるのがワイルドカードです。トランプのジョーカーみたいなものなどと比喩されます。* は複数の文字列を、?は 1 文字だけを任意の文字として扱います。例えば次のようにカレントディレクトリの中で、list だけを抜きだしたくなったとします。

```
nursefan% ls
list1.list  omake1.list  sample1.c   sample4.pl   text3
list2.list  omake2.list  sample2.f   text1
list3.list  omake3.list  sample3.pl  text2
nursefan% ls *.list
list1.list  list3.list   omake2.list
list2.list  omake1.list  omake3.list
nursefan%
```

ここで * は list1. だろうが omake3. だろうがなんでも良いことになります。

?は 1 文字ですから、

```
nursefan% ls text?
text1  text2  text3
nursefan%
```

となって、1 でも 2 でも 3 でも良いことになります。これは、rm や mv などでとても便利なことが実感されるはずです。

9.2 簡単な UNIX コマンド

詳しくはマニュアルを見て下さい。最後にマニュアルの見方について簡単に触れます。

pwd

カレントディレクトリを絶対パスで教えてくれます。とりあえずやってみて下さい。特に感動のあるコマンドではありませんが、あとで “知ってて良かったこのコマンド” best 3 にはいること間違いなしです。

ls

カレントディレクトリのファイルやらなんやらを教えてくれます。オプションに-a を付けると、. ファイルも表示してくれます。 -l を付けるとデータ量やモード情報などを、 -F を付けると、ファイルなのかディレクトリなのかその他なのかを教えてくれます。

ファイル名の後に / が付いていればディレクトリを * が付いていれば実行ファイルを、@が付いていればシンボリックリンクを示しています。実行ファイルやシンボリックリンクについては UNIX の本を参照して下さい。

cd

ディレクトリを移動します。引数を付けなければ自分のホームディレクトリへ、カレントディレクトリにあるディレクトリ (例えば sample なるディレクトリ) への移動は

```
nursefan% cd ./sample/
```

とします。

mv

ファイルやディレクトリを移動したり、名前を変えたりします。使い方は、

```
nursefan% mv file directory
```

とすれば、ファイルを指定されたディレクトリへ移動します。ここでディレクトリは絶対パスでも相対パスでも指定できます。ファイルやディレクトリの名前を変えたい時は

```
nursefan% mv before after
```

とすれば、before というファイル (またはディレクトリ) が after という名前に変わります。

ps

自分がマシンにやらせているプロセスを教えてくれます。他人が行わせているプロセスも知りたかったらオプション[28]に -a をつけて下さい。 -u とか -x とか全部まとめて -aux とすれば、いろんなことがわかります。

mkdir

カレントディレクトリの下 (というか、中というか。普通は下) にディレクトリを作ります。引数に、好みの名前を付けて下さい。ただし、後で見た時にわかる名前をお勧めします。

rm

ファイルを削除します。引数にファイル名を指定して下さい。オプションに -i を付けると削除する前に “本当にいいの? 後悔しない?” と聞かれますので、事故を予防できます。

rmdir

rm のディレクトリ版です。

chmod

基本的にファイルは他人からも見られるし、誰かのファイルも見ることが出来ます。しかしモードを変えることによって、ある程度は自由に変更することができます。(何を言っているかわからないとは思いますが、大目に見て下さい。) 実際にやってみます。まずは、ls -lF をやってモードを調べます。

```
nursefan% ls -lF
-rw-rw-rw-   1 nuresefan students    54778 Feb 20 18:26 hidebu
drwxr-xr-x   2 nuresefan students      512 Feb 19 02:53 awabyu/
```

これは、hidebu なるファイルと awabyu なるディレクトリについての情報です。F もついているので、awabyu に/が付いています。

ここで左に注目してみると、-rw-rw-rw- と、何かの暗号のような項目が 10 項目あります。詳しくは私も知りませんが、いちばん左の - に d とあればディレクトリ、- ならファイルです。

ところでユーザーはファイルに対して “読む” “書く” “実行する” の、3 種類の働きかけを行うことが出来ます。

しかしユーザーにも種類があって、“ファイルの作成者 (つまりあなた自身)” “同じグループの人” “赤の他人” がいます。

残った 9 項目は、左から 3 項目がファイルの作成者、中の 3 項目は同じグループの人、最後の 3 項目は赤の他人がそれぞれ “読む” “書く” “実行する” を行えるかどうかを示しています。 各々の 3 項目は “読む” “書く” “実行する” が対応します。つまり、読めるなら “r” を、書けるなら “w” を、実行できるなら “x” を、禁止ならば “-” を記します。

[28] オプションの書き方は OS によって異なります。エラーになるときはマニュアルを参照してください

一般的には、“他人にファイルを見せてもいいけど、書き換えられたり実行されるのは困る”というモードを選びます。次のようにして下さい。

```
nursefan% chmod 644 hidebu
nursefan% ls -l
-rw-r--r--   1 nuresefan students    54778 Feb 20 18:26 hidebu
drwxr-xr-x   2 nuresefan students      512 Feb 19 02:53 awabyu/
```

ここで 644 という数字は、最初のうちはそういうものだと思って下さい。同じことをディレクトリに対して施す時は、755 とします[29]。

cp

ファイルをコピーします。 例えば

```
nursefan% cp hekoheko pukapuka
```

とすると hekoheko をコピーして、 pukapuka という名前を付けたファイルを作成します。もちろん相対パスや絶対パスで指定できます。例えば 1 つ上のディレクトリの中の abesi というファイルを、カレントディレクトリの下にある awabyu というディレクトリの中に hidebu という名前でコピーしたければ、

```
nursefan% cp ../abesi ./awabyu/hidebu
```

として下さい。

more

ファイルの中身を覗き見します。引数にはファイル名を与えて下さい。スペースキーで前に進めます。途中で見るのをやめる時は q として下さい。

よく似たコマンドに **less** があり、Linux などでは less の方が一般的です。

grep

文字列の検索をします。例えばカレントディレクトリの中にある全てのファイルの中から nurse という文字列のみを探したければ、

```
nursefan% grep nurse *
```

として下さい。ここで*はワイルドカードです。もちろん具体的なファイル名でも構いません。検索の対象をファイル名のみとするなら、-f オプションを付けます。カレントディレクトリのファイル名を検索してくれます。

[29] 4 は読む、2 は書く、1 は実行するをそれぞれ許可します。全部許可するなら、4+2+1=7 としますし、実行だけ許可するなら 0+0+1=1 とします。例えば 644 とは、“自分は読み書きを、同じグループの人と赤の他人は読むだけ”を許可し、701 だったら、“自分は読み書き実行を、同じグループの人は全部禁止、他人は実行だけ”を許可します。詳しくは、UNIX の本を参照して下さい。

lpr

プリンターに出力します。引数は出力したいファイル名です。オプションを付けなければデフォルト (初期の設定のことだと思って下さい。正しくは省略とか、そんな意味だったはず。) のプリンターに出力します。

lpq

誰がプリンターにジョブ (プロセスとほぼ同義。どんな時に使い分けるのかはよく解りませんが、慣習に従ってここではジョブという言葉を使います) を送っているかを調べることが出来ます。

man

マニュアルを出してくれます。マイナーなコマンドにはないものもあります。引数にコマンド名をとります。頑張ってよく読んで下さい。

10 より真剣に学ぶために

本マニュアルは対象を全くの初学者に限定して書かれているため、浅いところや不完全なところが文字通り溢れんばかりの構成になっています。当然より進んだ使用法を知りたい方もいらっしゃることでしょう。そのためにまず必要なのはやる気と根気、そして英語の辞書ではないでしょうか? しかしここで冗談ばかり言っている訳にはいきません。幸いにしてより進んだマニュアルが既にリリースされていますので、どうぞそちらを御参照下さい。

“たのしい IRAF” は、本マニュアルの次のステップとして最適だと思われます。よりたくさんのタスクについて、懇切丁寧な解説が載っています。

“IRAF クックブック” は、日本語のマニュアルとしては最高のマニュアルだと思われます。 IRAF のみならず、計算機、観測機器、データ処理等の、より高いレベルでの理解が必要な方は是非とも御一読下さい。

参考文献

[1] 能丸 淳一、西原 英治, 楽しい IRAF

[2] IRAF クックブック

[3] ローカルユーザーズマニュアル

[4] 奥村 晴彦, LaTeX2ε 美文書作成入門, 技術評論社, 1997.

[5] 山口 和紀 (監修), The UNIX Super Text(上、下), 技術評論社, 1992.

[6] 福島 英雄, 天文アマチュアのための冷却 CCD 入門, 誠文堂新光社, 1996.

おわりに

一年前、メールの読み書きぐらいしか満足に出来なかった私が、どういう訳だかこんな簡易マニュアルを作成することになりました。お蔭様で LaTeX 2_ε についても半ば強制的に学ぶことができました。家庭用ゲーム機を除いて“コンピュータ”と名のつくものから逃げまわってきた私がよくもまあ、という感を拭い去れません。折に触れ書いてきましたが、このマニュアルの対象は全くの初学者が意識されています。そこで、ある程度は自助努力によって救われるようにとの願いを込めて、私が陥ったトラブルと対処法については紙面を割きました。計算機アレルギーが今よりも激しかった当時を思い起こして初歩の初歩から書いたつもりではありますが、必要以上に丁寧だったり、あるいは逆に説明が足りなかったりすることもあるとは思います。それもこれも浅学非才な私による執筆ということで何卒御容赦下さい。

最後になりますが、嫌がる私を無理矢理計算機に縛り付け、拷問にも似た教育 (言い過ぎか?) を施して下さった東京理科大学大学院の山本直孝氏と木下大輔氏に、心から感謝を致します。

国立天文台 広報普及室付家事手伝い 副主任
(電気通信大学 電気通信学部)
中村 隆

1999 年 3 月

改訂 田口弘子 / 監修 青木賢太郎
2000 年 6 月

改訂 濱部　勝
2002 年 6 月