IRAF

USO E DESENVOLVIMENTO

João Luiz Kohl Moreira

Departamento de Astrofísica
OBSERVATÓRIO NACIONAL

INTRODUÇÃO

IRAF é a sigla de *Image Reduction and Analysis Facility* e começou a ser desenvolvido em fins de 1982 por Doug Tody no *National Optical Astronomy Observatories*, em Tucson, AZ. Consistia numa unificação dos pacotes já existentes de redução de dados do Kitt Peak e Cerro Tololo. Posteriormente, em 1983, Doug Tody e Peter Shames do STScI fundiram o sistema com o SDAS (*Science Data Analysis System*) já em desenvolvimento para o tratamento de dados do Telescópio Espacial. Do trabalho das equipes dos dois Institutos produziu-se o atual IRAF e o STSDAS, plenamente compatíveis e distribuidos conjuntamente.

O IRAF-STSDAS é de domínio público e pode ser obtido através da aquisição de uma fita magnética, com custos apenas da unidade e do frete, ou então via FTP para "noao.edu".

A idéia inicial foi desenvolver um sistema em que os pacotes de redução para os diversos observatórios americanos pudessem ser integrados e assim distribuidos para os interessados. A época em que o IRAF passou a ser desenvolvido coincide com um momento em que outros sistemas de tratamento de imagens no mundo são desenvolvidos ou aperfeiçoados na linha de adaptação às facilidades de novos sistemas computacionais baseados no conceito de *Virtual Memory* tais como o já então consagrado *VAX* bem como as recém lançadas *Work Stations* baseadas no UNIX, agora, finalmente amadurecido depois de quase 20 anos no forno. Entre esses sistemas destaca-se o MIDAS (*Munich Image Data Analysis System*), produto do esforço europeu baseado no instrumental do ESO, no Chile, e o Star-Link, produzido pelos ingleses para, inicialmente suprir necessidades de processamento de dados da rádio astronomia, mas abarcando um conjunto grande de aplicações também na astronomia ótica. Apesar desses três sistemas estarem em plena atividade, o IRAF fixou-se quase como um padrão no mundo inteiro. As razões são diversas, porém as que parecem mais prováveis são:

1. O IRAF foi integralmente desenvolvido com base nas *Work Stations* SUN, se que tornou um padrão nas universidades e observatórios americanos. A partir daí ele migrou para outros sistemas como o VMS;

2. Foi assimilado por mais de 95% dos observatórios americanos, e sendo os americanos em maior número, por consequência, com mais produção , e, finalmente, mais unidos, a aplicação do IRAF tornou-se obrigação para aqueles interessados em trabalhos de cooperação com os americanos, o que hoje em dia é quase uma necessidade profissional.

3. Há de se convir que o IRAF resolve 99% das necessidades da astronomia ótica, com alto padrão profissional e rigor estatístico. Por outro lado, o IRAF é altamente consistente. Uma vez que você aprendeu como lidar com ele, saberá se adpatar rapidamente a novas tarefas, a desenvolver em cima delas, automatizar seus procedimentos e, o que é importante, obterá "helps" facilmente em 95% dos casos. Os outros sistemas pecam por não conseguirem apresentar as mesmas condições simultaneamente. Alguns serão melhores em certos aspectos. Mas nem todos apresentam todas as qualidades tão bem quanto o IRAF.

Entre as facilidades que o IRAF oferece encontramos o tratamento de:

- IMAGENS;
- ESPECTROS;
- FOTOMETRIA;
- FOTOMETRIA SUPERFICIAL;
- ESTATÍSTICA;
- PACOTES DE REDUÇÃO FINAL.

Ao tratamento de IMAGENS, chamamos todos os procedimentos de pré-redução e redução de imagens CCD e placas fotográficas tanto de campos estelares, galáticos, quanto de espectros bi-dimensionais. ESPECTROS são pacotes dedicados ao tratamento de espectros uni e bi-dimensionais, em calibração tanto fotométrica, quanto em comprimento de onda. FOTOMETRIA são pacotes dedicados a fotometria estelar, enquanto a SUPERFICIAL possibilita a análise morfológica, calibrada de galáxias e aglomerados. A ESTATÍSTICA do IRAF permite análise final, ajuste de funções e testes de verossimilhança, o que juntamente com os pacotes de REDUÇÃO FINAL permite obter os dados finais, prontos para publicação , inclusive com gráficos e tabelas, se assim o desejarmos. Como veremos adiante, o IRAF permite interface com todos os outros pacotes do sistema local, ou seja TEX, MONGO, GNUPLOT, etc.

Além dessas facilidades, o IRAF permite uma série de possibilidades de "fabricação" de dados artificiais. Como se sabe a produção de dados artificiais são de relevante importância para a implantação de novos programas e aplicações bem como de testes estatísticos de dados observados. Em alguns casos, porém com frequência menor que o desejado, tarefas do IRAF fornecem *demos*, que são muito úteis como ilustração para quem deseja aprender sua utilização.

Os manuais do IRAF são caros e desatualizados. Talvez esse seja o seu ponto fraco. Mas, nem por isso deixam de ser úteis. Está disponível para quem o desejar alguns textos mostrando roteiros de alguns pacotes de redução, escritos muitas vezes por usuários, por isso bastante úteis. Contudo, com a prática, aprendemos a obter as informações de seus "helps", sabendo obter todas as dicas deles.

O IRAF contém algumas interfaces para usuários e programadores locais. Entre elas, encontra-se:

- GTERM (Graphic Terminal);
- IMTOOL (Image Tool);
- IMFORT (Image Fortran library);
- SPP (Subset PreProcessor)
- SAOIMAGE (versão do IMTOOL do SAO);

- CL (Command Language).

O GTERM provém do início do IRAF, desenvolvido para o SunView e migrado para o RGB, da estação *GPX* da Digital. Apesar do SunView estar obsoleto, o GTERM é de bastante utilidade sobretudo para quem desenvolve. É onde o IRAF funciona a toda prova. Além disso, você pode chamar o "gterm" de qualquer janela "OpenWindow".

A "ferramenta" IMTOOL é um suporte gráfico para aqueles que trabalham com imagens bi-dimensionais. Também desenvolvido para o SunView e RGB. Todos os aplicativos IRAF são voltado para o IMTOOL que, em suma, estabeleceu um protocolo. Para o "OpenWindow", o SAO (Smithsonian Astrophysical Observatory) desenvolveu o SAOIMAGE que segue esse protocolo. Alguns usuários IRAF, sobretudo do FOCAS, preferem trabalhar com o IMTOOL por causa de sua "vocação natural" em interagir com o FOCAS. Igualmente, é chamado sem problemas pelo "OpenWindow".

Alguns astrônomos necessitam usar suas próprias aplicações pois não estão satisfeitos com o que o IRAF oferece. Nesse caso, temos duas alternativas: 1) Adaptar seus aplicativos ao ambiente do IRAF e integrá-los às *tasks* já existentes em sua configuração pessoal; 2) Usar sub-rotinas da biblioteca do IRAF para aproveitar o formato "IMH" com que os dados são armazenados em disco, e usar os aplicativos independentemente. Essa alternativa evita o transtorno de transformação de formatos que são longos e ocupam muito espaço em disco. Tanto para uma quanto para outra alternativa o usuário dispõe da biblioteca IMFORT e do compilador SPP. O primeiro é uma biblioteca de sub-rotinas FORTRAN que permite a manipulação de dados em formato "IMH" e interface com o IMTOOL e o segundo é um pré-compilador cuja linguagem se parece com o 'C' mas, segundo o *staff* do NOAO, permite as facilidades matemáticas do FORTRAN. O SPP tanto serve para implementar aplicações pessoais e independentes do IRAF, quanto para acrescentar novas *tasks* ao ambiente IRAF. Um exemplo de necessidade de aplicativos especiais vem do pessoal de observação de satélites internos dos grandes planetas devido aos cuidados necessários à subtração do céu. Vale acrescentar que esses são casos muito raros e não é recomendável a não ser quando for o caso de redução de dados provenientes de equipamentos de nova tecnologia. Essencialmente todos os aplicativos para a redução tradicional já existem no IRAF.

Finalmente temos a interface mais comum e conhecida por todos. É o 'CL', a interface ao usuário. CL, como indica seu nome, é um interpretador de comandos do IRAF, assim como o tem, o PC-DOS, UNIX, VMS, MTS, etc. Com o CL você interage com o sistema, lança os comandos e recebe os resultados; igualmente programa um conjunto de instruções e manda executar sob um só nome. É com o CL que você se comunica com o IRAF. A esse respeito vamos falar mais adiante pois ele merece um capítulo especial, senão, a maior parte de um manual sobre o IRAF.

O IRAF é dividido em "pacotes" (packages) que por sua vez podem ser divididos em sub-pacotes ou "tarefas" (tasks). Uma task pode ser um programa executável ou um conjunto de sub-tarefas (script). Esse último não é nada mais que um arquivo texto contendo um conjunto

de comandos IRAF, podendo ser modificado e re-executado. Comparando com o VMS, seriam os arquivos ".COM", com o PC-DOS, os ".BAT" e com o MTS, os arquivos 'SO'.

Um package IRAF está associado ou com um conjunto de pacotes de redução desenvolvido num determinado sítio ou com interfaces com o terminal em uso. Os principais packages IRAF são:

- NOAO. Conjunto de packages desenvolvido no NOAO dedicado sobretudo à redução de observações no Kitt-Peak. Os packages disponíveis do NOAO são:
    - artdata - Artificial data generation package
    - astrometry - Astrometry package (não implementado)
    - astutil - Astronomical utilities package
    - digiphot - Digital stellar photometry package
    - focas - Faint object classification and analysis package
    - imred - Image reductions package
    - mtlocal - Magtape i/o for special NOAO format tapes
    - nobsolete - Obsolete tasks to be phased out in a future release
    - nproto - Prototype (temporary, contributed) tasks
    - observatory - Examine and define observatory parameters
    - onedspec - One dimensional spectral red & analysis package
    - rv - Radial velocity analysis package
    - surfphot - Galaxy isophotal analysis package (não, implementado, ver STSDAS)
    - twodspec - Two dimensional spectral red & analysis package.

Esses packages cobrem a quase totalidade das necessidades de pré-redução- redução de dados observados em um observatório tradicional.

- STSDAS. Desenvolvido pelo pessoal do STScI para análise dos dados do Telescópio Espacial. Nele, encontra-se aplicativos bastante úteis na análise de Brilho Superficial. Os packages principais são:
    - contrib - User-contributed software
    - fgs - Calibration & analysis tasks for the Fine Guidance Sensors.
    - fitsio - FITS input/output for Space Telescope data (images and tables).
    - fitting - Curve fitting tools.
    - foc - Calibration & analysis tasks for the Faint Object Camera.

  - fos - Calibration & analysis tasks for the Faint Object Spectrograph.
  - fourier - Fourier analysis.
  - gasp - Access the HST Guide Star Catalog on CD-ROM.
  - hrs - Calibration & analysis tasks for the High Resolution Spectrograph.
  - hsp - Calibration & analysis tasks for the High Speed Photometer.
  - isophote - Elliptical isophote image analysis.
  - playpen - Miscellaneous experimental tasks.
  - problems - Submit an STSDAS software problem report to STScI.
  - statistics - Statistical analysis software.
  - stlocal - STScI local applications and data base access utilities.
  - stplot - STScI graphics tasks (for images and tables).
  - synphot - HST synthetic photometry tasks.
  - testdata - Tools for creating artificial images.
  - timeseries - Time series photometry data reduction and analysis.
  - tools - Generic data handling and utility tools.
  - ttools - Table manipulation tools.
  - wfpc - Calibration & analysis tasks for the Wide Field/Planetary Camera.
- CTIO. Criado para as aplicações específicas de Cerro Tololo. É subdividido em:
  - fabry - Fabry-Perot package
  - apropos - Print help information about a subject
  - bin2iraf - Convert a binary file into an IRAF image
  - bitstat - Make bit statistics of image frames
  - colselect - Select column ranges
  - compairmass - Compute airmass
  - coords - Compute celestial coordinates
  - cureval - Evaluate the fit computed by CURFIT
  - dfits - Diplay FITS file headers
  - eqwidths - Compute equivalent widths
  - ezimtool - Easy IMTOOL

- fft1d - One dimensional FFT
- findfiles - Find files recusively through directories
- fitrad - Fit a function to circular image subrasters
- fixtail - Fix last image lines or columns
- focus - Compute telescope focus
- growthcurve - Correct magnitudes using the growth curve for them
- gkitocad - Convert GKI metacode files into autoCAD DXF files
- helio - Compute heliocentric Julian date
- imcreate - Create an image
- imextract - Extract a list of image sections from an image
- immatch - Match one or two dimensional images
- imsort - Sort images by header parameter values
- imtest - Image headers and pixel files test
- irlincor - Correct IR imager frames for non-linearity
- lambda - List wavlength and pixel values
- magavg - Average output from MAGBAND for a single object
- magband - Compute the average magnitude for a list of bandpasses
- mapkeyword - Replace image header keyword values with other values
- midut - Compute UT at midpoint of observation
- mjoin - Join lines in text files by matching them
- mkapropos - Make apropos database
- pixselect - List pixel values within a certain range
- spcombine - Intercative combine task
- sphot - Star photometry
- statspec - Compute statistical error from object and sky spectra
- wairmass - Compute weighted airmass
- manuals - CTIO manuals (must load CTIO to access)

- IMAGES. São conjuntos de packages e tasks voltados para a interação com o terminal gráfico, no caso SUN, IMTOOL. Note-se, especialmente, o package "tv", onde está a task "display", que joga a imagem na janela gráfica, seja o IMTOOL ou seu emulador, o SAOIMAGE.
- PLOT. É voltado para a interação com terminais do tipo Tektronics e seus equivalentes. Pode-se obter um "contour" de uma imagem, ou o "plot" de um espectro uni ou bi-dimensional, linha a linha. É a interface ideal para quem trabalha com terminais remotos, porém nada se pode fazer com relação à visualização de imagens em níveis de cinza, tarefa possível apenas com o IMTOOL ou SAOIMAGE. Por isso, terminais do tipo "xterm" são interessantes por causa da compatibilidade com o SAOIMAGE.
- DATAIO. Conjunto de tasks para a conversão de formatos, por exemplo, leitura e escritura de fitas FITS, leitura e escritura de arquivos ASCII, etc.
- LOCAL. Área dedicada a implementações locais.

Uma vista mais apurada verá que os diferentes pacotes se interseccionam, isto é, algumas tasks de diferentes packages fazem a mesma coisa, ou quase a mesma coisa. Essas aplicações acabam ganhando preferência de acordo com o gosto de cada um. Com o tempo escolhemos nossas aplicações preferidas. É importante frisar que o IRAF, como qualquer implementação completa, demanda tempo para ser satisfatoriamente compreendido. Não se trata de um sistema "amigável", e nem poderia devido ao seu largo espectro de aplicações. Uma vez "adaptado", o usuário tende a utilizá-lo em larga escala e muitos possuem aplicações automatizadas tornando suas tarefas muito mais confortáveis.

1 O IRAF NA SUA MÁQUINA

Instalado, o IRAF tem uma "conta". Isto é, a instalação do IRAF exige que seja criada uma conta como se existisse um usuário chamado "IRAF". Esta conta não deve ser disponível ao usuário. Ela é, exclusivamente, acessada pelo "IRAF-manager", aquele encarregado de fazer a manutenção do IRAF. Esse é a prática comum aos grandes pacotes tais como o MONGO e o VISTA. Alguns administradores tem por hábito manter uma conta para cada software instalado. Tendo uma conta especial, o IRAF possue um "diretório - login" ou "default directory". Geralmente ela é '/usr/local/iraf/local' para as SUNs. No DAF esse diretório é '/home2/iraf/local'. Dentro desse diretório estão os principais pacotes disponíveis na fita de distribuição. Alguns pacotes independentes que são agregados ao IRAF são guardados em outros diretórios para evitar confusão no momento de atualizações de novas versões. O diretório-login do IRAF é, portanto, importante para a definição dos packages e tasks rodando dentro dele. **Para o IRAF um package está associado a um diretório, geralmente do mesmo nome**. Na notação IRAF, o seu diretório-login possue um valor simbólico: "iraf$". Isto quer dizer que se num dado momento, dentro do IRAF, você fizer: "cd iraf$", você se colocará no diretório-login do IRAF, onde quer que você esteja trabalhando, qualquer que seja a máquina, esteja ele onde estiver. Mas cuidado. Em princípio você

não poderá escrever nada nele, pois esse diretório não lhe pertence. Além disso, **não trabalhe dentro dessa ou outras áreas que não lhe pertencem, pois o IRAF não irá funcionar. Todas as tarefas de I/O vão falhar**. Dentro da mesma lógica, se você fizer "cd stsdas$", você se colocará dentro do diretório do package STSDAS, "cd noao$" levará você ao diretório do package NOAO. Essa prática somente é interessante para aqueles curiosos em saber onde estão as coisas no IRAF, pois toda a utilização dos packages será transparente ao usuário. Não é nem necessário que o usuário saiba onde está o IRAF. Basta saber que ele funciona. Você poderá, contudo, copiar para o seu próprio diretório o conteúdo de toda uma package e modificá-la na medida de suas necessidades. Isso é recomendável apenas para quem está desenvolvendo novas aplicações. Mas para chegar a esse ponto, quem está começando terá de percorrer ainda um caminho, não tão espinhoso, mas um tanto longo.

2 FINALMENTE COMEÇAR

Para você trabalhar com o IRAF será preciso inicializar o seu diretório. Para compreender melhor o sentido disso, vamos passar rapidamente no caminho que o IRAF faz para se permitir interagir com o usuário.

Invoca-se o IRAF através dos comando "cl", que como vimos é a interface usuário. O "cl" é um "script - file". No VAX ele se chama fisicamente "CL.COM" e se existisse um IRAF para PC ele se chamaria "CL.BAT". Você pode ler o seu conteúdo pois trata-se de um arquivo texto. Ele está, geralmente, em '/usr/local/bin'. No DAF ele está em '/home2/bin'. Após fixar valores simbólicos para algumas variáveis de ambiente, inclusive o "iraf$", ele executa então o binário "iraf$bin/cl.e". Entre outras coisas, esse executável procura pelo arquivo "login.cl" no diretório corrente, aquele diretório em que você está quando faz "cl". A ausência deste fará o "cl" mandar uma mensagem de erro. Em seguida ele fornecerá o prompt "`cl>`". Na existência da mensagem de erro é melhor sair do IRAF, com um "logout". Esse "logout" fará você sair do IRAF, mas não de sua sessão. Outra questão importante é que cada task que você rodar vai testar a existência do sub - diretório "./uparm". Na ausência deste a task não vai funcionar. Por isso, é necessário inicializar o IRAF para o seu diretório. Para isso você dará o comando:

```
% mkiraf
```

Você terá de responder a algumas perguntas. Invariavelmente você terá de dizer qual é o tipo de terminal que você estará usando (*xterm, vt100, gterm, tek4010, etc*). Aqueles que vão usar o IRAF sob o OpenWindows, o melhor é responder *xterm*, os que vão usar sob SunView, respondam *gterm*, os que vão usar PCs com Kermit emulando terminal gráfico, usem o *tek4010*, ou *tek4014*. Uma vez executado o "mkiraf", você terá o arquivo "login.cl" e o diretório "./uparm". Tudo isso no diretório que você está.

Quando você já inicializou o IRAF e ao fazer "cl" você receber a mensagem de erro, significa que você não está naquele diretório que anteriormente você inicializou com "mkiraf". Isso porque

o IRAF é "sensível" ao diretório que você está e não à conta que você tem naquela máquina. Esse aparente contra senso tem uma razão simples de ser. Sendo o IRAF um sistema que depende altamente de uma "personalização" você pode querer ter várias "personalizações" diferentes. Por exemplo: se for o caso de um astrônomo trabalhar com imagens, espectros e campos de galáxias ele pode criar diferentes diretórios para cada área: "./espectros", "./imagens", "./campos", etc. Em cada diretório ele inicaliza com 'mkiraf' e então trata de especificar a aplicação para o IRAF, carregando packages e estabelecendo uma "personalização" de cada aplicação.

Uma vez rodado o "mkiraf", você não precisa e nem deve mais fazê-lo, a não ser se recomendado pelo administrador. A partir daí e em diferentes sessões, você roda o IRAF chamando simplesmente 'cl'.

3 USANDO IRAF

O "cl" é o comando que você passa para o sistema entrar no ambiente IRAF. Quando você dá esse comando, ele começa por atribuir valores a algumas variáveis de ambiente, as chamadas *environment variables*. Exemplo:

```
set iraf "path"/iraf/
set user `whoami`
set HOME "~/"
```

E assim por diante. São variáveis necessárias ao IRAF para que ele possa rodar corretamente. Em seguida ele vai ler o conteúdo do arquivo "iraf$'arch'/hlib/irafuser.csh", onde 'arch' é a arquitetura ou sistema instalado, por exemplo, "unix", "vms", etc. Nesse arquivo, encontra-se também uma série de outras atribuições. Depois ele vai procurar por "iraf$pkg/cl/clpackage.cl" onde as tasks e pacotes originais são definidos, depois ele lê no mesmo diretório, o "extern.pkg" que são pacotes externos, mas compatíveis (STSDAS, por exemplo) e distribuidos conjuntamente. Finalmente ele procura o "login.cl" e em seguida testa a existência de "loginuser.cl", no seu diretório corrente, se ele existir, então será consultado. É nesse último arquivo que costumamos fazer as "personalizações". Finalmente, o "cl" oferecerá o prompt "`cl>`".

3a) Brincando de gente grande....

Muitos gostam de experimentar o uso de softwares pois facilita a sua compreensão. Como roteiro para uma primeira intromissão no IRAF eu recomendo o seguinte: tenha o cuidado de se situar no terminal que voce declarou quando inicializou com "mkiraf". Se foi "xterm", abra uma janela xterm e trabalhe lá, se foi "gterm", faça o mesmo com gterm, etc. Em seguida faça.

```
% cl
cl> plot
```

Ao fazer "cl", você terá na tela uma mensagem do tipo:

```
Setting terminal type to xterm...
    NOAO SUN/IRAF Revision 2.10EXPORT Thu Apr 30 15:41:01 MST 1992 (Rev. 1)
    This is the EXPORT version of Sun/IRAF V2.10 for SunOS-4.1.

    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:

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

Ao fazer "plot", você terá:

```
  calcomp     gkidir      imdkern     phistogram  sgidecode   surface
  contour     gkiextract  implot      pradprof    sgikern     velvect
  crtpict     gkimosaic   nsppkern    prow        showcap
  gdevices    graph       pcol        prows       stdgraph
  gkidecode   hafton      pcols       pvector     stdplot
```

Ou seja, você entrou no package "plot". Um "help" em cada pacote dará a descrição sumária de cada comando disponível. Em seguida você faz:

```
cl> contour dev$pix
```

Deve aparecer na tela gráfica o gráfico de contornos de uma galáxia espiral, mais precisamente a M51, na constelação de *Canes Venatici*. Se você quiser sair do IRAF, nessa altura, basta fazer "logout". Atenção, esse "logout" não fará você sair do sistema. Você ainda estará "logado". Dê um "logout" novamente ou escolha "exit" no MENU se você estiver no SunView ou OpenWindow.

Você pode continuar pesquisando um pouco mais. Para aquele que pode acessar o package "ctio" pode usar o utilitário "apropos" que ajuda bastante na hora de pesquisar o que se pode fazer no IRAF.

```
cl> ctio
ct> apropos photometry
```

Eis o resultado de "apropos photometry" que obtive na implementação do DAF:

```
digiphot       - Digital stellar photometry package                    [up]
                 (noao)
```

```
apphot        - Aperture Photometry Package (noao.digiphot)
photpars      - Edit the photometry parameters (noao.digiphot.apphot)
sphot         - Star photometry (ctio)
photman       - Photometry Manual (ctio.manuals)
stphot        - Photometry Manual for PHOTRED (in progress) (ctio.manuals)
synphot       - HST synthetic photometry (stsdas)
timeseries    - time series photometry data reduction and analysis (stsdas)
focphot       - FOC photometry package (stsdas.foc)
elapert       - create polygon files for use in APPHOT polygonal photometry
                (stsdas.isophote)
calcphot      - Calculate synthetic photometry for a given spectrum and
                (stsdas.synphot)
```

O "apropos" procura na base de dados de todos os "helps" do IRAF tudo que contém "photometry". Com o tempo tende-se a melhorar esse tipo de utilitário pois muitos "helps" não são normalizados para as mesmas palavras chaves. Outra limitação é que o "apropos" não acessa o FOCAS pois o este package tem seu próprio "help" (FOCASMAN). Contudo, esse é um meio poderoso de se "achar" tasks escondidas em todo o IRAF pois o "apropos" não se limita às tasks do "ctio".

Outra facilidade é o "help". O help fornece todas as informações a respeito de um comando, dando também a qual package pertence. Isso permite que possamos localisar uma dada task, sabendo em qual package ela se encontra. Isso é de fundamental importância pois não são poucos – sobretudo aqueles que não treinaram em centros onde o IRAF está instalado, ou aqueles que não conhecem a capacidade de "ajuda" do IRAF – a reclamar que o IRAF é um verdadeiro 'labirinto'.

Caso você goste dos "Quick Cards", isto é, de espécies de "colas" ao seu lado enquanto estiver usando o IRAF, mande imprimir numa impressora PostScript o arquivo stsdas$doc/user/UserQuick-Ref.ps. Você terá duas páginas impressas, que você pode colar uma contra a outra ou tirar uma cópia "xerox" de cada na mesma folha, em frente e verso. Em seguida, dobre em três partes de maneira a ter um "cartão" de pequenas dimensões.

Última possibilidade de conhecer o IRAF através de suas próprias facilidades é o conjunto de "demos". São "scripts" construidos especialmente para mostrar os procedimentos para diversas formas de redução de diferentes tipos de dados. Veja como o "apropos" listou o conjunto "demo":

```
ccdtest       - CCD test and demonstration package (noao.imred.ccdred)
demo          - Run a demonstration of the CCD reduction package
                (noao.imred.ccdred.ccdtest)
apdemos       - Various tutorial demonstrations (newimred.apextract1)
demos         - Demonstrations and tests (newimred.argus)
demos         - Demonstrations and tests (newimred.goldcam)
demos         - Demonstrations and tests (newimred.kpcoude.fiber)
```

```
demos          - Demonstrations and tests (newimred.kpcoude.slit)
demos          - Demonstrations and tests (newimred.nessie)
demos          - Demonstrations and tests (newimred.specred)
```

Recomendo fortemente que você leia com carinho esses "scripts". A maneira de acessá-los é bem simples. Você carrega o package desejado, por exemplo, o "ccdred" e vai para o seu diretório. Os passos a serem feitos são:

```
cl> noao
no> imred
ir> ccdred
cr>ccdtest
cr> cd ccdtest$
```

Dentro desse diretório você deverá encontrar o arquivo "demo". Você, então, poderá fazer:

```
cr> page demo
```

Tire uma impressão, com "print demo" e rode esse programa, com o cuidado de, antes, voltar ao seu diretório, com "cd". Acompanhe os passos com a impressão ao lado.

3b) Comandos "caseiros".

Uma vez rodando o "cl", existem dois tipos de instrução:

1. O que carrega um package;
2. O que executa uma task.

Ambos são feitos com a mesma sintaxe, isto é, entrando-se com o nome pertinente na linha de comando. Por exemplo:

```
cl> dataio
```

Carrega o pacote "dataio", que contém tasks de leitura e escritura de dados em uma unidade de i/o. Uma vez carregado o pacote aparece na tela a lista das tasks e/ou pacotes disponíveis dentro dele. Não é mais necessário chamar um pacote carregado na sessão corrente a menos que você dê um "bye". O comando "bye" descarrega o pacote atual. Se voccê der o "dataio", por exemplo, receberá o prompt "da>" para indicar o pacote recém carregado. Porém, se de "dataio" você quiser carregar o pacote "noao", que está um ramo acima na árvore não será necessário dar um "bye" e depois "noao". De "da>" você poderá entrar "noao":

```
da> noao
```

Assim você terá carregado os pacotes "dataio" e "noao", podendo lançar, indistintamente tasks dos dois pacotes. Para saber os pacotes que você tem carregados basta dar o comando:

```
cl> package
```

Uma vez lançada uma task, ela será executada tomando três tipos de parâmetros: *required*, *automatic*, e *mode*. Esses parâmetros são tomados do arquivo de mesmo nome e extensão ".par". Primeiro o IRAF procura no diretório corrente o diretório "uparm", e dentro dele o seu arquivo de parâmetros. Se não encontra esse arquivo, então ele vai procurar por ele no diretório [package]$, onde [package] é o nome do package que contém a dita task. Exemplo: a task "rfits" que lê arquivos FITS em fita, possue o arquivo de parâmetros "dataio$rfits.par", pois pertence ao package "dataio". Uma vez que você rodou a task "rfits", se for a primeira vez, então "rfits" vai procurar por "dataio$rfits.par", e, em seguida, copiá-lo para o seu diretório "uparm". Ela "aprenderá" os valores dos parâmetros que você fornecer, por exemplo, o nome da unidade de fita a ser lido e na próxima vez que você rodar essa task, ela já saberá o nome da unidade e pedirá confirmação. Essa é a função do sub-diretório "uparm" em seu diretório: permitir que a task "aprenda" os seus parâmetros pessoais. Se quiser que a task "desaprenda" essa personalização você pode dar o comando "unlearn [nome da task]". Isso terá como consequência o desaparecimento do arquivo "[package].par" de seu diretório "uparm". Assim, a próxima vez que você rodar essa task, ela terá que ir buscar no seu diretório de origem o arquivo '.par' relativo a ela. Exemplo de utilização de "rfits":

```
cl> dataio
da> rfits mtg 1-12 data
```

Vai ler os arquivos de 1 a 12 na fita "mtg"[1] e guardá-los nos arquivos "data001.imh", "data002.imh"..."data012.imh"[2]. Na próxima vez, você pode simplesmente fazer:

```
da> rfits
```

O programa vai pedir para confirmar a unidade (mtg), os arquivos (1-12) e o nome de saída (data). Ao mudar qualquer um desses parâmetros a próxima chamada virá com os valores atualizados. Sendo imprecindíveis para a execução do programa, são os chamados parâmetros *required*. Além desses parâmetros a task "rfits" possue outros como "make_image", um parâmetro do tipo booleano (toma valores TRUE ou FALSE), "long_header" (idem) que são do tipo *automatic*. Eles têm valores default e assim permanecerão a não ser que você os modifique explicitamente na linha de comando ou edite o arquivo "uparm$rfits.par". Por exemplo, se você der o comando:

```
da> rfits mtg 1-12 long+ make-
```

O programa vai somente listar o cabeçalho dos arquivos, sem, no entanto, criar arquivos de dados em disco. Esses parâmetros não são atualizados com esse procedimento. Somente o serão se você editar o arquivo correspondente.

---

[1] "mtg" na implementação do DAF é a unidade de exabyte.

[2] Na verdade não é bem assim. Os arquivos ".imh" contém tão somente o cabeçalho do frame. A matriz dos dados será guardada em um arquivo ".pix" no diretório "imdir$", mas isso já é uma outra história e é melhor voltarmos a essa questão mais tarde.

O último tipo de parâmetro é o *mode*. Ele pode ser: 'q', 'l' e 'a'. O *mode* 'l' quer dizer *learn*, o que faz dos parâmetros *required* passíveis de modificação, ou seja, de "aprender"; o 'q', *query* é o que faz o programa pedir para confirmar os parâmetros *required* e por último 'a' define que todos os parâmetros do programa são declarados *automatic* sem que ele pergunte ou peça confirmação. Mais a frente veremos como editar arquivos de parâmetros.

3c) Comandos "estrangeiros".

Digamos que você queira rodar um programa que não faz parte do pacote IRAF, por exemplo: mongo. Basta, para isso que você anteceda o comando com um "!":

```
cl> !mongo
```

À saída do mongo, você volta ao IRAF no ponto em que você estava quando entrou no mongo. Da mesma forma você poderá fazer com seus próprios programas ou qualquer outro que seja reconhecido pelo sistema operacional corrente.

Você pode, por outro lado, definir que um dado comando de sistema seja reconhecido pelo IRAF de forma transparente. Por exemplo, na implementação do DAF, existe um comando "ontofits" que é usado para transformar as imagens do PDS-ONSAD em arquivos FITS. Assim, pode-se transportar essas imagens para qualquer sistema de tratamento de dados, inclusive o IRAF. Pode-se incorporar esse programa ao IRAF através do comando:

```
cl> task $ontofits = "$foreign"
```

A partir daí a cada vez que se quer rodar o "ontofits" basta fazer:

```
cl> ontofits
```

O programa vai rodar como se fosse parte integrante do IRAF. O mesmo pode-se fazer com "mongo", TEX, etc. O "$" na frente do "ontofits" tem uma função específica, bem diferente do "$" presente nas definições de diretório IRAF. Essa é a indicação de que a "task" não tem arquivo de parâmetros, particularidade única dos comandos nativos. Assim, se você quer incoporar o "mongo", você deve fazer: `task $mongo = "$foreign"` e o mesmo com outros comandos. Da mesma forma, você pode incluir programas próprios, guardados em diretórios seus. Digamos que você tenha um diretório "progs" onde estão guardados seus programas executáveis, exemplo, monique.exe, pfeifer.exe, andrea.exe. Defina suas próprias tasks da seguinte forma:

```
cl> reset progs = home$progs
cl> task $modelo = progs$monique.exe
cl> task $sylvia = progs$pfeifer.exe
cl> task $paquita = progs$andrea.exe
```

Esse exemplo serve para ilustrar que o nome de suas tasks não são necessariamente o nome de seus programas. Uma vez definidas as tasks, você pode rodar seus programas simplesmente fazendo:

```
cl> modelo
.
.
cl> paquita
```

O papel do "reset progs..." é de definir o diretório "progs" que será reconhecido pelo IRAF com `progs$`. O `home$` é o seu diretório "default", aquele em que você está quando inicia a sessão. Se por acaso, o seu diretório de executáveis chama-se "exec" e está sob o diretório "progs" que por sua vez está sob o seu diretório "default", você deve definir o "reset" assim: `reset execs home$progs/exec`. Se por acaso você produzir um programa com arquivo de parâmetro[3] você deverá definir a task com `task novo = progs$novo.e` onde "novo" é o nome do programa que o emprestará à task. Todas essa definições podem ser feitas de uma vez por todas dentro de arquivos específicos para isso. O exemplo do "login.cl" é ilustrativo e veremos em que arquivo podemos fazer essas definições na sequência do estudo do "login.cl".

3d) Como é o "login.cl".

Como já vimos, o "login.cl" é o arquivo que o "cl" consulta imediatamente antes de fornecer o seu prompt. Vejamos como é um "login.cl" típico na implementação do DAF, versão 2.10, intermeado com comentários:

```
# LOGIN.CL -- User login file for the IRAF command language.

# Identify login.cl version (checked in images.cl).
if (defpar ("logver"))
    logver = "IRAF V2.10EXPORT April 1992 revision 1"

set home = "/home3/users/kohl/"
set imdir = "/home4/iraf/tmp/kohl/"
set uparm = "home$uparm/"
set userid = "kohl"
```

Note que o sinal "#" server para tornar tudo o que vem a direita na linha um comentário. Não tem significado para o IRAF. Por outro lado, são definidas variáveis simbólicas através dos "set"s. Os valores dessas variáveis são obtidos no "cl" através do "$" imediatamente após seu nome. Por exemplo `home$` fornece o valor da variável "home", que no caso é `/home3/users/kohl`. Esse, por sinal, é o meu diretório login. Se você se perguntar qual é a diferença entre o "set" e o "reset", a resposta é simples. "set" atribui um valor para uma nova variável, enquanto o "reset" faz o mesmo para uma variável já existente. Caso essa variável também seja nova, o "reset" faz o papel de "set", porém se você usar "set" para uma variável já existente, ocorrerá um erro. O `imdir$`

[3] Isso só é possível seguindo os procedimentos padrão de compilação e instalação de novas tasks dentro do IRAF. Não trataremos disso aqui, havendo, contudo, manuais específicos para tal.

é o diretório onde o IRAF vai guardar a matriz dos dados, os chamados arquivos ".pix". Quando virmos a estrutura de dados do IRAF, discutiremos mais o significado do arquivo ".pix". Em todo caso, se você quiser declarar o `imdir$` para o seu diretório login, basta fazer:

```
set imdir = home$
```

```
 Set the terminal type.
if (envget("TERM") == "sun") {
    if (!access (".hushiraf"))
print "setting terminal type to gterm..."
    stty gterm
} else if (envget("TERM") == "xterm") {
    if (!access (".hushiraf"))
print "setting terminal type to xterm..."
    stty xterm nl=44
} else {
    if (!access (".hushiraf"))
print "setting terminal type to xterm..."
    stty xterm
}
```

Nesse ponto é definido o terminal escolhido para se trabalhar, seja o "gterm", "xterm", etc. Esse não é o exemplo ideal para quem trabalha com um terminal "tek4014", por exemplo. Poder-se-ia acrescentar as linhas

```
} else if (envget("TERM") == "tek4014") {
    if (!access (".hushiraf"))
print "setting terminal type to tek4014..."
    stty tek4014
```

logo após a definição: `stty xterm nl=44`. Para aqueles que trabalham com PC, usando um emulador de terminal como o "KERMIT", e conectado à SUN via porta serial, pode substituir na linha `if (envget(...` o "tek4014" pelo nome do terminal em cuja porta está identificado. Esse nome pode ser obtido fazendo `printenv TERM` no UNIX. Assim as linhas determinando o terminal no IRAF, se por exemplo esse terminal chama-se "VT102", seriam assim:

```
} else if (envget("TERM") == "vt102") {
    if (!access (".hushiraf"))
print "setting terminal type to tek4014..."
    stty tek4014
```

Façamos uma explanação sumária do que está acontecendo: "if" se explica por ele mesmo, não é necessário entrar em detalhes. O "envget" faz a função do "printenv" do UNIX, ou "show

symbol" do VMS. Quando, sob o UNIX[4], você faz `printenv TERM`, você obtém o tipo de terminal definido para aquela janela. Esse é o princípio do "envget" no IRAF. Um equivalente dessa função é a subrotina "getenv" do FORTRAN-UNIX ou o C-UNIX. O termo "access" indica se o arquivo em questão existe no diretório corrente, o "!" na frente do "access", 'nega' o valor da variável, isto é, `!access(".hushiraf")` terá o valor TRUE se o arquivo ".hushiraf" não existir, FALSE se for o contrário. O resultado é que se você tiver o arquivo ".hushiraf" em seu diretório não será emitida mensagem ao seu terminal. O teste da existência desse arquivo é feito em outras ocasiões, de maneira que se ele existe, nenhuma mensagem de "apresentação" será transmitida ao terminal. Essa sistemática se inspira no próprio UNIX, que possui arquivos ".hush" para inibir mensagens de apresentação. O conteúdo desse arquivo não é consultado. Testa-se apenas a sua existência. Ele poderá ter dimensão 0. Pelo "script" acima, então, se não existir o arquivo ".hushiraf" no diretório, então você terá a mensagem: "setting terminal to...". A seguir vem a linha `stty xterm` ou outro terminal escolhido. Esse é um comando **IRAF** que estabelece o terminal em que este vai comunicar[5] O equivalente UNIX-C-Shell desse comando é o `setenv TERM`.

```
# Uncomment and edit to change the defaults.
set editor = vi
set printer = lpr
set stdimage = imt800
set stdimcur = stdimage
set stdplot = lpr
set clobber = no
set filewait = yes
set cmbuflen = 512000
set min_lenuserarea = 24000
set imtype = "imh"
```

Aqui são declaradas variáveis usadas para identificar "devices" e interfaces. O "editor = vi" significa que quando você fizer "edit [nome do arquivo]" será acionado o editor "vi" do UNIX. No VAX o "editor" seria declarado "= edt". Igualmente, quando você faz "print [nome do arquivo]", será acionado o "lpr" da Sun-Unix. O "stdplot" é o "plotter" a ser usado quando se for imprimir um gráfico ou imagem. A variável "clobber" tem uma função interessante: quando declarada "no", ela terá o valor FALSE. Nesse caso você pode escrever um arquivo imagem ou texto no lugar de um arquivo existente com o mesmo nome. Caso seja declarada "yes", ao contrário, ela terá o valor TRUE e o sistema impedirá que você escreva qualquer arquivo em disco, caso ele já exista. Como a linha indica, seu valor "default" é FALSE e tudo que você escrever em disco será aceito, mesmo que o ou os arquivos já existam. A última variável que merece atenção, por enquanto é o "imtype".

[4] Estamos nos referindo ao *C-shell*. Caso você trabalhe sob o *Bourne shell*, procure se informar junto ao seu System Manager

[5] Aqueles habituados ao UNIX não devem confundir com o comando de mesmo nome rodando no UNIX. Nesse caso, o `stty` modifica variáveis de interação com o terminal.

Seu valor default é "imh". Outro valor poderá ser o "hhh". O primeiro indica o formato do NOAO enquanto o segundo, o STSDAS. Ambos são aceitos de maneira transparente pelo IRAF.

```
# IMTOOL/XIMAGE stuff.  Set node to the name of your workstation to
# enable remote image display.
#set node = ""

# CL parameters you mighth want to change.
#ehinit   = "nostandout eol noverify"
#epinit   = "standout showall"
showtype = yes

# Default USER package; extend or modify as you wish.  Note that this can
# be used to call FORTRAN programs from IRAF.

package user

task $adb $bc $cal $cat $comm $cp $csh $date $df $diff = "$foreign"
task $du $find $finger $ftp $grep $lpq $ls $mail $make = "$foreign"
task $man $mon $mv $nm $od $ps $rcp $rlogin $rsh $ruptime = "$foreign"
task $rwho $sh $spell $sps $strings $su $telnet $tip $top = "$foreign"
task $touch $vi $emacs $w $wc $less $rusers $sync $pwd = "$foreign"

task $xc $mkpkg $generic $rtar $wtar $buglog = "$foreign"
#task $fc = "$xc -h $* -limfort -lsys -lvops -los"
task $fc = ("$" // envget("iraf") // "unix/hlib/fc.csh" //
    " -h $* -limfort -lsys -lvops -los")
task $nbugs = ("$(setenv EDITOR 'buglog -e';" //
    "less -Cqm +G " // envget ("iraf") // "local/bugs.*)")
task $cls = "$clear;ls"
```

Aqui estão declaradas os comandos do sistema reconhecidos pelo IRAF, da forma como explicamos em *Comandos estrangeiros*. Como o próprio comentário do arquivo diz, você pode extender esses comandos a qualquer outro que você deseja (ex., "mongo"), desde que seja reconhecido pelo sistema instalado.

```
if (access ("home$loginuser.cl"))
    cl < "home$loginuser.cl"
;

keep;   clpackage
```

Aqui são tomadas as intruções pessoais do usuário. Repare que ele vai buscar o arquivo "loginuser.cl" no "home", ou diretório login. Por isso, a "personalização" desejada não deve ser feita aqui, a não ser que seja apenas UMA personalização. O "keep" tem a mesma função que o "wait" no UNIX, isto é, o sistema "espera" que os comandos anteriores sejam completamente executados antes de seguir em frente. O "clpackage" é um pseudo-package. São carregados os packages declarados no arquivo iraf$unix/hlib/clpackage.cl. São alguns dos packages nativos do IRAF.

```
prcache directory
cache   directory page type help
```

Esses dois comandos são importantes de serem compreendidos. A cada vez que você entra com um comando, o IRAF vai procurar no disco, dentro do PATH previamente especificado, carregar o programa na memória e então executá-lo. O "prcache" encarrega-se de carregar na memória e deixá-lo lá mesmo que este não esteja sendo executado. Essa possibilidade acelera enormemente a execução de programas grandes como o "display", que coloca uma imagem na tela, ou então em tarefas de execução contínua de uma mesma task, dentro de um "loop", por exemplo. O "prcache" geralmente carrega todo o pacote em que a dada task está, enquanto o "cache" carrega a task. Por isso, caso você queira acelerar suas tarefas, o melhor a fazer é carregar a task com "prcache" e em seguida, "cache". No caso dado no "login.cl", o `prcache directory` carrega todo o pacote que contém esse comando (no caso o "system"), enquanto o `cache directory page type help` carrega essas task do "system" e o deixa na memória. Ao invocá-las, serão executadas imediatamente.

```
# Print the message of the day.
if (access (".hushiraf"))
    menus = no
else {
    clear; type hlib$motd
}


# Delete any old MTIO lock (magtape position) files.
if (deftask ("mtclean"))
    mtclean
else
    delete uparm$mt?.lok,uparm$*.wcs verify-
```

Encontramos novamente o ".hushiraf" cuja presença no diretório inibirá o envio da "mensagem do dia" que o administrador IRAF local reserva para você. As linhas subsequentes não tem maior interesse pelo momento.

```
# List any packages you want loaded at login time, ONE PER LINE.
images          # general image operators
plot            # graphics tasks
```

```
dataio          # data conversions, import export
lists           # list processing

# The if(deftask...) is needed for V2.9 compatibility.
if (deftask ("proto"))
    proto       # prototype or ad hoc tasks

tv              # image display
utilities       # miscellaneous utilities
noao            # optical astronomy packages

keep
```

Eis aqui a oportunidade de você personalisar definitivamente a sua aplicação. Coloque imediatamente antes do "keep" todos os pacotes relativos ao tipo de redução que você quer fazer. Pode-se perguntar por que não colocar a personalisação integral em um só "login.cl" no nosso diretório login. Não há impedimento algum. Dois motivos podem ser dados para evitar essa prática.

1. Por uma questão de organização. É difícil administrar arquivos de tipos diferentes e sobretudo administrar o que se deve apagar quando o seu administrador vier reclamar que você está ocupando muito espaço em disco, ou o que é pior, quando você estoura sua quota e deve arranjar espaço livre;
2. Você sobrecarrega a memória do computador com coisas que não serão usadas. Isso pode prejudicar a você mesmo, sobretudo se a máquina que você tem não é poderosa. Pode-se perguntar quando isso pode acontecer. Essa pergunta um administrador pode responder com "Quase todo dia!".

Por isso, a melhor sistemática é aquela em que você "partilha" suas aplicações em diferentes "IRAFs", para diferentes diretórios.

Veja, por exemplo, como seria os principais packages que um astrônomo "espectral" deveria carregar em "login.cl":

```
noao
ccdred
twodspec
onedspec
plot
images
tv
```

De acordo com sua conveniência, você pode carregar outros pacotes além desses ou no lugar desses. Tudo isso vai por conta do gosto de cada um, pois em "stsdas", "ctio", por exemplo existem tasks coincidentes.

Para terminar o "login.cl", o "keep" espera que tudo seja carregado e feito, então termina sua tarefa e o "cl" oferece o prompt `cl>`.

3e) Como se estrutura um arquivo imagem.

Um arquivo "IMH" é dividido em duas partes. O "cabeçalho", conhecido com "header" e a matriz de dados, o chamado arquivo "PIX". A primeira parte encontra-se, geralmente, no seu diretório de trabalho, por exemplo "OBJ001.IMH". Associado a esse, num diretório reconhecido pelo IRAF como "imdir$", encontra-se o arquivo "PIX", com o mesmo nome mas com terminação ".PIX", por exemplo "OBJ001.PIX". Esse diretório "imdir$" é pré-definido pelo administrador IRAF local, mas você pode mudá-lo no "login.cl". Veja a listagem desse arquivo acima e você verá onde modificar esse valor. Alguns usuários gostam de declarar o "imdir$" como o seu diretório de trabalho, por exemplo, `imdir = HOME$iraf/`. Questão de gosto, mas deve se evitar essa prática sobretudo se o seu sítio não tem muito espaço em disco. Isso porque o administrador, nesses casos, reserva uma partição especialmente para guardar os arquivos "PIX". Caso a direção desses arquivos seja modificada pode haver um acumulo de dados excessivo na partição de usuários, pois os arquivos "IMH" são relativamente pequenos, no entanto os arquivos "PIX", dependendo do caso podem chegar a uma dimensão muito grande.

Essa formatação de dados é feita de maneira a otimizar a ocupação de espaço e a velocidade de tratamento. Alguns usuários têm me procurado para dizer que seus dados guardados em ASCII são menores do que os arquivos IRAF. Eu costumo responder que essa diferença em favor dos arquivos ASCII é provavelmente aparente. Via de regra essa aparente redução de tamanho só é possível em imagens de uma só linha. Não é grande vantagem "salvar" seus dados em arquivos ASCII, se você ainda não terminou sua redução.

Se você tem um arquivo chamado "obj001.imh" no seu diretório, dê o seguinte comando para saber o conteúdo do "header".

```
cl> imheader obj001 l+
```

O conteúdo integral do header será mostrado na tela. Você notará que na primeira linha do header está declarado o tipo de variável declarada para os dados. Essa declaração é imprescindível para a interpretação dos dados contidos em "PIX". Os dados que estão em "PIX" serão interpretados segundo a indicação do tipo de variável: se "integer", "real", "double", etc.

Exemplo Uma aplicação

Vou dar um exemplo de aplicação, reduzindo uma noite de observação feita no LNA em 15/11/91, por Lício da Silva (DAN - ON), a quem agradeço pela cessão a título de demonstração. São espectros tomados no espectrógrafo Cassegrain no telescópio de 160 cm. Apesar de se tratarem de espectros, a sequência mostrada pode ser feita exatamente para o caso de campos estelares ou galáticos, salvo menção contrária. Tratando-se de redução CCD, não importa o tipo de objeto observado, pois o interesse é tão somente recuperar a confiabilidade das observações.

O que temos.

Tipicamente os frames vindos do LNA possuem um nome associado ao objeto observado e com extensão indicando o "tipo" do objeto. No caso temos uma convenção que Lício usou. Ex.:

Tipo da imagem	Extensão
Objeto de estudo	.ima
bias	.idk
obscuro	.idk
calibração	.icl
flat field	.iff

Via de regra os arquivos estão em formato FITS* Após criar o diretório "lna15no91" com `mkdir lna15no91` fiz `mkiraf` e declarei o terminal "GO140", porque estou trabalhando através de um PC, com *kermit* emulando "TEK4014". A experiência mostra que essa é a melhor escolha de terminal no "mkiraf" nesse caso. Em seguida, usei o `vi` para editar o "login.cl" e introduzir os pacotes "imred" e "ccdred'.

A fita que me foi cedida por Lício da Silva é um "exabyte" em forma de "tar". Nesse caso bastou-me simplesmente "untargar" a fita nesse meu diretório. A fita também pode estar em formato IEEE criado pelo comando "wfits" do IRAF. Nesse caso devemos entrar no "cl" e recuperar os dados através de "rfits".

* Por questões de compatibilidade com algumas implementações do INPE, os arquivos FITS vêm com os "bytes" invertidos porque essa é a convenção usada em PCs. É preciso, pois, alertar os técnicos do LNA para *swapear* os bytes antes de gravar em fitas ou disquetes. Por desinformação, alguns colegas não advertem os técnicos do LNA. Por isso, muitas vezes temos arquivos FITS com os "bytes" invertidos o que produz dados absolutamente irreais. Para corrigir esse problema (além de voltar ao LNA e pedir delicadamente para gravar tudo de novo, com os bytes na ordem correta), o IRAF permite uma pequena modificação no programa "rfits". Vamos omitir comentários a respeito porque tal assunto extrapola os objetivos desse humilde texto. De qualquer forma, insista junto ao "IRAF-manager" local, se for o caso, com essa pequena dica.

Entrei no ambiente IRAF:

```
% cl
```

Se o seu caso for o descrito logo acima, você deve dar os comandos:

```
> dataio
> rfits [unidade] 1-[N] Fits
```

onde [unidade] é a unidade lógica da fita. Depende da unidade, e do sítio. Na implementação do DAF a referida unidade lógica é o "mtg", para o caso do exabyte e "mtf" para o caso do cartucho de 150 Mby. [N] é o número final do arquivo a ser recuperado. Se você quer todos os arquivos, use "*".

Uma vez dentro do IRAF, é bom dar uma olhada no conteúdo de nosso diretório. Após "untargar" os dados do Lício, eis o conteúdo de nosso diretório.

```
hd19745ne.icl   hd19745nha1.icl hd19745nha2.icl hd30171nha.icl irj199nha1.icl
irj199nha2.icl cup01.iff        cup02.iff       cup03.iff       cup04.iff
cup05.iff       cup06.iff       cup07.iff       cup08.iff       cup09.iff
cup10.iff       bbw76.ima       cd473307.ima    deltataua.ima   deltataub.ima
fuoria.ima      fuorib.ima      gamaeria.ima    gamaerib.ima    gamataua.ima
gamataub.ima    hd19745a.ima    hd19745b.ima    hd286746.ima    hd287927.ima
hd288012.ima    hd30171a.ima    hd30171b.ima    hd39853a.ima    hd39853b.ima
hd787a.ima      hd787b.ima      hen001a.ima     hen001b.ima     hen373.ima
irj008.ima      irj160.ima      irj199a.ima     irj199b.ima     irj888a.ima
irj888b.ima     sky30001.ima    sky30002.ima    sky30003.ima    sky30006.ima
sky30007.ima    sky30008.ima    sky30009.ima    sky30010.ima    bias5s01.idk
bias5s02.idk    bias5s03.idk    bias5s04.idk    bias5s05.idk    bias5s06.idk
bias5s07.idk    bias5s08.idk    bias5s09.idk    bias5s10.idk
```

Vemos que existem "imagens", "flat-fields", "bias", "céu", etc. A primeira coisa a fazer é transformar esses arquivos FITS em arquivos ".imh". Para ajudar a identificação posterior, aproveitei a convenção usada no LNA e pelo Lício para separar diferentes classes, fornecendo nomes finais diferentes. Para isso, adotei a prática de guardar os nomes dos arquivos em "lists".

```
> ls *.idk > Zerofits
> ls *.iff > Flatfits
```

E assim por diante. Assim, pude, em seguida, transformar os arquivos para o formato ".imh", dando os nomes aos diferentes arquivos segundo sua classe. Explicando: de posse dos arquivos de nome, Zerofits, Flatfits, Skyfits e Calibfits, usei o "rfits" da seguinte forma:

```
> rfits @Zerofits * zero
```

O "@" na frente do arquivo Zerofits significa que o comando precedente deve tomar o "conteúdo" do arquivo e lá tomar o nome do arquivo a ser tratado. No arquivo Zerofits estão os nomes dos

arquivos "BIAS". Logo, serão esses arquivos que serão transformados. O "*" em seguida deve ser usado no caso dos arquivos FITS estarem no disco e não no fita. E, finalmente, o "zero" é o "prefixo" dos arquivos de saída. Assim, ao final da execução, teremos arquivos chamados "zero0001.imh", "zero0002.imh" e assim por diante, com seus respectivos pares ".pix" no diretório "imdir$". Da mesma fora, faz-se com os outros:

```
> rfits @Calibfits * calib
> rfits @Objectfits * obj
> rfits @Flatfits * flat
```

Tenho, então, todos os dados transformados ao formato de trabalho do IRAF. Não preciso mais dos FITS. Usamos, da mesma maneira os arquivos "list" para apagá-los:

```
> del @Zerofits
> del @Calibfits
> del @Objectfits
> del @Flatfits
```

É bom evitar apagar os arquivos "list". Como veremos mais tarde eles serão uteis.

De posse dos arquivos, teremos que seguir os procedimentos de redução contidos no package "ccdred". As tasks desse package fazem um conjunto poderoso de tratamento. Em princípio a redução é automática, sobretudo para as observações provenientes do Kitt Peak ou Cerro Tololo. Para imagens vindo do LNA, diríamos que é "semi-automática". O roteiro que descrevo abaixo é baseado no "A User's Guide to CCD Reductions with IRAF" de Phillip Massey, 1989. É o primeiro artigo do item "KPNO Cookbooks - User's Guides" do Volume 2B do "conjunto azul" da documentação do IRAF. Perdoem-me a "chupada", inclusive das piadas. Mas trata-se de um documento muito bem escrito e recomendo a todos aqueles que querem se aprofundar no assunto.

Antes, vamos fazer alguns comentários a respeito de observações CCD. Considere o que vem a seguir como parênteses explicatório.

Ea) Tratamento de Imagens CCD.

A observação com o CCD exige algumas medidas observacionais afim de assegurar a qualidade dos resultados. O surgimento desse dispositivo representou um avanço admirável nas técnicas de observação mas o aprofundamento dos estudos de suas características mostra um conjunto de precauções a fim de se obter dele o máximo de sua capacidade. Essas medidas são listadas abaixo e se referem sobretudo à qualidade de calibração. O pessoal do IRAF desenvolveu rotinas padronizadas para o tratamento dessas medidas especialmente para os sítios de KPNO e CTIO. Mas, como as características dos CCDs científicos são as mesmas, pode-se com facilidade transportar essas aplicações para observações provenientes do LNA, por exemplo. Antes de entrarmos na redução propriamente dita, vamos a uma breve discussão dessas práticas observacionais.

Começo do parênteses: Exposições auxiliares.

– Bias frame ◦ Reconhecido como "zero frame" dentro do IRAF. São exposições com zero tempo de integração. Serve para estimar o nível do ruido eletrônico intrínsico. Devido a sua natureza aleatória deve ser feito em quantidade. A subtração de 1 bias frame da imagem aumenta o ruído nos dados de um fator de $\sqrt{2}$. A subtração do bias produto de uma combinação de 25 bias frames aumenta esse nível de apenas 10%. Fundamental.

– Dark frame ◦ Longas exposições com o diagrama fechado. Reconhecido como "dark frame" no IRAF. Recomenda-se obter poses tão longas quanto a mais longa exposição da noite. Serve para medir o incremento de ruido eletrônico durante a pose. Sua quantidade, apesar de muitas serem benvindas, é consequência da capacidade observacional do programa e qualidade das noites. Aproveite noites fechadas. Necessária para observações de brilho superficial e espectrofotometria sem "overscan lines", isto é, CCDs sem regiões não expostas durante a observação.

– Flat field frame ◦ Exposições da cúpula iluminada por uma luz branca (dome flat field) ou exposições diretas por uma lâmpada de quartzo na rede de difração (projector flat field). Reconhecida como "flat frame" pelo IRAF. Uma vez iluminado uniformemente, esse tipo de exposição mostra a variação pixel a pixel de sensibilidade. Para fotometria, deve ser obtido para cada filtro utilizado uma vez que cada pixel possui uma curva de resposta espectral diferente. Na prática, o flat field é capaz de corrigir apenas as variações locais, não sendo capaz de corrigir as variações de baixa frequência espacial devido a inomogeneidades da iluminação. Ideal que se chegue a ∼10.000 eletrons. Fundamental.

– Twilight flat frame ◦ Especialmente usado para resolver problemas de correção de iluminação na fenda em espectroscopia. Expõe-se o CCD à luz do crepúsculo (ou alvorada, se se está a ponto desesperado). Nesse caso deve-se levar em conta que essa luz tem aproximadamente o espectro do sol e assim levar esse fato em consideração. No caso de fotometria, preste atenção nas variações rápidas do brilho do céu, e, sobretudo, não aponte para o Sol. Contagem ideal de ∼10.000 eletrons. Necessário para brilho superficial e espectros.

– Blank dark sky frame ◦ Visto que a cor do crepúsculo é diferente da do céu noturno muitos preferem expor o CCD a uma região "sem objetos". Alguns observatórios têm informações das coordenadas dessas regiões. Se não for o caso, é uma boa política fazer você mesmo uma base de dados

nesse sentido. Região "vazia" é um conceito relativo e depende da profundidade que se observa. É de bem fazer um "smooth" desses frames caso não haja variações na resposta em cor na região do CCD tratada. Uma alternativa ao "twilight flat field" para brilho superficial e espectrofotometria. Ideal de $\sim 10.000\, e^-$.

– Fringe frame
  - Algumas linhas de emissão do céu provocam franjas de interferência no caso de fotometria com filtros "broad band". Isso é raro e não tenho conhecimento de coisa parecida nos CCDs do LNA.

Fim do parênteses.

A primeira providência a ser tomada será acertar o uso do comando `setinstrument`. Esse comando costuma tomar valores dos arquivos em "ccddb$[sítio]". O valor de [sítio] na versão original do IRAF pode ser "kpno" ou "ctio". Dentro desses diretórios estão "registrados" alguns "instrumentos". Exemplo: dentro do diretório "ccddb$kpno" encontra-se um arquivo chamado "echelle.dat" e outro chamado "echelle.cl". Eles constituem um associação lógica com o espectrógrafo "ECHELLE" instalado em Kitt Peak. Se suas observações vêm desse instrumento, você deve iniciar seu trabalho de redução com:

```
> setinstrument echelle
```

Da mesma forma você pode fazer com "coude" ou "specphot". No entanto, pode ser que suas observações não venham do Kitt Peak. Se fosse o caso você não estaria lendo esse texto. Caso você tenha observações do CCD de Cerro Tololo, você terá que mudar os parâmetros:

```
> setinstrument ccd dir=ccddb$ site=ctio
```

Não esqueça o '/' no final do nome do diretório. Em seguida ele vai dar uma "revisão" dos parâmetros de "ccdred" e "ccdproc". Essa revisão nada mais é do que o editor de parâmetros do IRAF chamado "epar". As eventuais mudanças serão vistas adiante, portanto você sai desse editor com ":q", como no "vi".

Mas você pode estar precisando reduzir observações do LNA. É preciso, pois, tomar umas medidas de inicialização de uma vez por todas. Mudar parâmetros que poderão ser aproveitados para as próximas observações, e modificar aqui e ali conforme eventuais mudanças no LNA. As principais dificuldades com relação à geração de um "instrumento" do LNA no IRAF diz respeito a:

1. No "cabeçalho dos frames do LNA não consta nenhum item relativo ao tempo de exposição. Isso introduz um trabalho a mais nas reduções;
2. O programa de aquisição de dados do CCD não é nada "amigável" na atribuição dos títulos no cabeçalho do arquivo FITS. Ao observador apressado é vedada essa sistemática pois toma tempo. O resultado é que os arquivos vêm com títulos espúrios restando apenas o expediente

de os nomes dos arquivos, em caso de disquete e fita "tar" estarem num formato mnemônico, como é o caso das observações de Lício da Silva, e os relatórios das observações.

Essas duas dificuldades podem ser facilmente suprimidas pelo pessoal do LNA se já não o foram. O fato é que isso depende da demanda dos astrônomos usuários e espera-se que a conscientização dessas necessidades de nossa parte trará os benefícios desejados (elimina-se aqui a parcela de culpa do LNA).

Devemos, pois, criar uma base de dados para o instrumento usado. Uma vez criada, essa base de dados será utilizada para todas as observações posteriores, sendo assim o trabalho feito uma só vez. Para reduzir as observações gerei uma base de dados e coloquei-a no meu próprio diretório "lnadb$". Se você tiver boas relações com o "manager" local, você pode convencê-lo a criar a base de dados "lna" e colocar como um subdiretório de "ccddb$" que pertence ao IRAF. Assim, ela pode ser usada por outros astrônomos e da mesma forma, configurações de outros "instrumentos" do LNA escritas por outrem podem ser aproveitadas por você.

Para gerar minha base de dados, criei um sub-diretório chamado "lnadb" dentro do diretório "lna15no91". Se fosse reduzir observações posteriores com o mesmo instrumento, criaria esse subdiretório "lnadb" no meu "home". Em seguida, editei o meu "login.cl" e introduzi a linha:

```
reset lnadb=./lnadb/
```

E entrei o comando "cl". Nesse momento a minha base de dados está reconhecida como "lnadb$". Para introduzir os parâmetros necessários, preferi tomar por base o instrumento "specphot" do "kpno". Então copiei os arquivos "ccddb$kpno/specphot.*" para o meu diretório "lnadb$".

```
> copy ccddb$kpno/specphot.* lnadb$speccass
```

"speccass" passa a ser, em princípio, a base de dados do Espectrógrafo Cassegrain do LNA. Mas é preciso editá-lo e modificá-lo em função das necessidades desse instrumento. Quero, entre outras coisas, introduzir uma linha declarando o tempo de exposição. Vamos a uma explicação sumária do significado dos arquivos - instrumentos e seus parâmetros. Quando você entra o comando "setinstrument", ele vai, procurar no diretório definido no parâmetro "diretory" o arquivo cujo nome coincide com o intrumento dado no comando (ex. ccd) e extensão ".cl" e executa o programa contido nesse arquivo. Vejamos o que contém o arquivo "speccass.cl" após as modificações que fiz de modo a definir corretamente o espectrógrafo Cassegrain do LNA:

```
# Generic routine for setting parameters.

ccdred.pixeltype = "real real"
ccdred.verbose = yes
ccdred.logfile = "logfile"
ccdred.plotfile = ""
ccdred.backup = ""
ccdred.instrument = "lnadb$speccass.dat"
```

```
ccdred.ssfile = "subsets"
ccdred.graphics = "stdgraph"
ccdred.cursor = ""

ccdproc.ccdtype = ""
ccdproc.fixpix = no
ccdproc.overscan = no
ccdproc.trim = yes
ccdproc.zerocor = yes
ccdproc.darkcor = no
ccdproc.flatcor = yes
ccdproc.readcor = no
ccdproc.scancor = no
ccdproc.readaxis = "line"
ccdproc.biassec = "image"
ccdproc.trimsec = "[3:57,150:1150]"
ccdproc.interactive = yes
ccdproc.function = "chebyshev"
ccdproc.order = 1
ccdproc.sample = "*"
ccdproc.naverage = 1
ccdproc.niterate = 1
ccdproc.low_reject = 3
ccdproc.high_reject = 3
ccdproc.grow = 1
```

Como se vê, define-se aqui um série de parâmetros para a package "ccdred" e a task "ccdproc". É bom que se explique que o comando do tipo:

```
> [nome do package ou task].[nome do parametro]=[valor]
```

muda o valor do parâmetro para o referido package ou task. Muda-se o valor do parâmetro dessa forma ou diretamente através de

```
> epar [nome do package ou task]
```

vai-se até a linha contendo o nome do parâmetro e escreve-se o seu valor seguido de [ENTER] para confirmá-lo. Sai-se de "epar" com um ":wq", exatamente como no "vi". Mas a mudança no valor dos parâmetros na base da declaração direta na linha de comando é o ideal para "scripts" como o "speccass.cl".

Seguido à execução do "script" ".cl", o "setinstrument" consulta o arquivo ".dat" (ex. ccd.dat). Esse arquivo ontém definições particulares. No caso do LNA, veja como deixei o arquivo "speccass.dat".

```
subset ""

imagetyp CCDTYPE
exptime INTEG
```

Esse arquivo contitui-se de linhas onde palavras chave das tasks do "ccdred" são associadas a palavras chaves contidas no "header" dos arquivos de dados. Assim se o "header" do arquivo de dados contém a palavra INTEG, seu valor será transferido para a variável `exptime`, o que para o "ccdred" significa tempo de exposição. Se os seus flat-fields foram tomados com tempos diferentes, não importa se para espectros ou fotometria, essa variável é fundamental. Da mesma forma, a palavra CCDTYPE no "header" do seu arquivo deverá indicar o tipo de observação feita, se bias ('zero'), flat field ('flat'), objeto ('object'), etc. O valor dessa palavra chave será transferido para "imagetyp". Quando quero impor um valor nulo para uma variável como "subset" eu coloco "" como acima. Essa variável, "subset" pode ser usada por aqueles que usam diferentes filtros fotométricos. Se você declarar o "subset" "filter", quando o `ccdproc` for combinar os flat fields ele vai levar em consideração a palavra chave "FILTER" e combinar os flat field apenas com os frames contendo o mesmo valor para ela. Igualmente quando for corrigir as imagens do flat field, serão consideradas correções com arquivos de mesmo filtro. Essa possibilidade é para livrar-nos da tarefa de prestar atenção do que corrigir com o que, coisa que nos leva a muitos erros.

O procedimento de consultar o arquivo ".dat" permite compatibilizar o package "ccdred" com o que o programa de aquisição de dados de um instrumento escreve no "header" dos arquivos FITS. Como já disse antes, o programa de aquisição do CCD do LNA não escreve no header uma palavra chave contendo o tempo de exposição, mas duas palavras contendo o início da exposição (TIME-BEG) e do seu fim (TIME-END). Com isso duas alternativas nos são dadas: editar o "header" de cada arquivo e introduzir a palavra INTEG (veja arquivo "speccass.dat" mostrado acima) com o tempo de exposição obtido da diferença (TIME-END - TIME-BEG); ou mudar os fontes dos programas de "ccdred", de maneira a fazer essa operação automaticamente. O ideal e mais natural seria fazer com que o programa do LNA introduzisse essa palavra no header no momento da aquisição.

Antes de prosseguir vamos agora, escolhido e editado nosso arquivo - instrumento, fazer:

```
> setinstrument speccass dir=lnadb$ site=""
```

E vamos acertar os "headers" das nossas imagens. Sem outra coisa a fazer, o mais fácil foi escrever uma rotina de maneira a que se introduza essa variável a mão. Aproveitei também para dar o título correto para cada arquivo pois ao transformar de FITS para IMH, eu perdi os nomes mnemônicos. Então eu transferi esses nomes dos arquivos FITS para o título dos dados em IMH. Para tanto, tirei uma listagem do arquivo Objectfits. Em seguida gravei os nomes dos arquivos obj*.imh em um arquivo "obj.lis", para depois escrever o programa*:

* Lício me informou que ele, por sua vez, escreveu um programa em FORTRAN fazendo a mesma coisa. Quem se interessar basta entrar em contato com ele.

```
cl> ls obj*.imh > obj.lis
cl> !ls calib*.imh >> obj.lis
cl> list='obj.lis'
cl> while(fscan(list,s1) != EOF){
>>>  hselect(s1,"TIME*","yes")
>>>  print(,'Entre tempo de exposicao:')
>>>  if(scan(s2) != EOF)
>>>   ccdhed(s1,"exptime",s2,type="real")
>>>  else
>>>  return
>>>  ccdlist(s1)
>>>  print(,'Entre nome do objeto:')
>>>  if(scan(s2) != EOF)
>>>   ccdhed(s1,"title",s2,type="string")
>>>  else
>>>  return
>>>  ccdhed(s1,"imagetyp","object",type="string")
>>> }
```

Imediatamente após entrar, o programa começa a executar listando o valor das palavras chaves "TIME-BEG" e "TIME-END", tarefa feita por "hselect". Em seguida ele vai imprimir: "Entre tempo de exposicao:" e ficará esperando pelo dado, tarefa feita por "scan(s2)". Coube a mim fazer a diferença e entrar com o tempo de exposição em segundos, minutos ou milisegundos. O importante é a consistência, uma vez colocado o tempo em segundos, não se poderá mais colocar em outras unidades. Minha experiência mostrou que, apesar de declarar o "exptime" como sendo "real", tasks como "ccdlist" reconhecem valores como inteiros e truncam os digitos do ponto decimal. Por medida de prudência, portanto, é melhor escolher suas unidades de tempo de maneira que os valores sejam inteiros.

Após recolher o valor do tempo de exposição, o programa coloca o header do arquivo com as palavras chave que interessam ao "ccdproc", entre elas o título do arquivo. Isso é para se ter uma forma de referência de qual arquivo estamos manipulando, já que estamos lidando com uma lista de arquivos. Em seguida ele pede para entrar com o nome do objeto. É bom que se esclareça que o nome do objeto não é a mesma coisa que o nome do arquivo que **contém** os dados do objeto. Para cada arquivo, seguindo a listagem de Objectfits, coloquei o nome do objeto coincidindo com o mnemônico colocado como nome do arquivo FITS original. Assim, mantenho ligado o arquivo sabendo com o quê estou mexendo.

Enfim, sem nada perguntar, o programa introduz a palavra IMAGETYP no arquivo com o valor "object". Mais tarde ao ver o header dos arquivos através de "imhead", você verá a palavra chave "CCDTYPE", no lugar de "IMAGETYP". Veja o arquivo "speccass.dat" listado acima para entender o que se passou. Essa tarefa que acabei de indicar pode ser um pouco enfadonha por

causa das operações de subtração dos tempos de fim e início de cada exposição. Mas, para consolo, poderíamos dizer que 50% do trabalho já foi feito.

Devemos fazer os mesmos tipos de modificações para os frames do tipo "flat".

```
cl> ls flat*.imh > flat.lis
cl> list='flat.lis'
cl> while(fscan(list,s1) != EOF){
>>>  hselect(s1,"TIME*","yes")
>>>  print(,'Entre tempo de exposicao:')
>>>  if(scan(s2) != EOF)
>>>   ccdhed(s1,"exptime",s2,type="real")
>>>  else
>>>  return
>>>  ccdhed(s1,"title","DOME FLAT",type="string")
>>>  ccdhed(s1,"imagetyp","flat",type="string")
>>> }
```

Voce pode perceber que foram introduzidos o tempo de exposição, o nome e o tipo de objeto nesse programa. Mesma coisa faz-se com os "sky flats":

```
cl> ls sky*.imh > sky.lis
cl> list='sky.lis'
cl> while(fscan(list,s1) != EOF){
>>>  hselect(s1,"TIME*","yes")
>>>  print(,'Entre tempo de exposicao:')
>>>  if(scan(s2) != EOF)
>>>   ccdhed(s1,"exptime",s2,type="real")
>>>  else
>>>  return
>>>  ccdhed(s1,"title","SKY FLAT",type="string")
>>>  ccdhed(s1,"imagetyp","flat",type="string")
>>> }
```

Finalmente devemos atualizar os arquivos "bias":

```
cl> ls zero*.imh > zero.lis
cl> list='zero.lis'
cl> while(fscan(list,s1) != EOF){
>>>  ccdhed(s1,"imagetyp","zero",type="string")
>>>  ccdhed(s1,"title","BIAS",type="string")
>>> }
```

Note que aqui você não precisa entrar com dado algum, pois o "bias" é assumido ter tempo de exposição nulo.

Antes de, finalmente, lançar os comandos para tratar as imagens, aproveito para fazer uns comentários acerca dos parâmetros de "ccdproc" que temos a oportunidade de modificar quando fazemos `setinstrument`.

fixpix=no
- O "ccdproc" permite que você forneça um arquivo com a listagem dos "bad pixels", ou seja, pixels "mortos" no meio do CCD. Eu procurei dentro dos frames "flat field", na região que tratei, pixels com valores muito abaixo da média e não encontrei. Decidi que o CCD em questão não possui tais pixels, pelo menos nada que não possa ser corrigido pelo flat field. Sem "bad pixels", o parâmetro "fixpix=no" indica que o "ccdproc" não deve fazer a correção. Caso "fixpix" tenha valor "yes", a operação será feita com o arquivo "badpixels" "default" o que será uma operação errônea, e se não encontrar esse arquivo, a operação será interrompida com uma mensagem de erro.

overscan=no
- Aqui o "ccdproc" faz a correção de "bias" no "overscan". Como vimos anteriormente, é o trecho da imagem contendo o "bias" obtido ao mesmo tempo que a exposição é passada ao computador. Nas imagens que tenho, esse "overscan" não existe. Logo devo impedir o "ccdproc" de tentar fazer essa correção.

trim=yes
- Escolhi uma região "útil" do CCD no parâmetro "trimsec" (ver mais abaixo). Por isso, indico que a operação de extração dessa região deve ser feita.

zerocor=yes
- Digo que o "ccdproc" deve fazer a correção do bias. Editar essa variável é indiferente pois vou mudá-la na linha de comando no momento conveniente.

darkcor=no
- Não houve exposição de obscuro. Essa correção não deve ser feita.

flatcor=yes
- Deve ser feita correção de flat field. Mas, da mesma forma que a correção do bias, vou mudar esse valor em função da minha conveniência.

illumcor=no
- Sem correção de iluminação na fenda. Aqueles que trabalham com espectroscopia, no entanto, geralmente terão necessidade de fazê-lo. Nesse caso, você deve declarar `illumcor=yes` Para isso, siga a secção 4.9 do artigo de Phillip Massey. Não vou demonstrar aqui porque as poses do céu mostraram fortes linhas de abserção. Talvez seja necessário "consertar" essas linhas, substituindo-as por valores interpolados caso não haja outra alternativa.

fringecor=no ◦ Sem correção de franjas de interferência.

readcor=no ◦ Essa variável indica que as images "bias" devem ser transformadas em 1 linha através das médias das outras e então corrigir as outras imagens. Tem a vantagem de diminuir os efeitos do ruído mas destroi a correção de 2^a ordem. Eventualmente para ser usado em caso de "bias" muito ruidosos.

scancor=no ◦ Nas nossas imagens não existe uma região "fora" medindo o bias em tempo real. Junto com "overscan=no", devemos colocar "scancor=no".

trimsec='[3:57,150:1150]' ◦ Olhando as imagens verifiquei que a região do CCD entre as linhas 1 e 150 têm uma péssima relação sinal/ruído. Assim, defini a região entre as linhas 150 e 1150 e as colunas 3 e 57 como sendo aquelas a serem tratadas. É sempre bom tirar as bordas... Uma vez terminada a operação de tratamento, as imagens finais conterão essa região extraída dos frames originais.

low_reject=3 ◦

high_reject=3 ◦ Quando combinamos os bias e flat fields para ter-se **o** bias e **o** flat-field, o programa "combine" faz uma estatística robusta e rejeita os pontos muito fora da média. É uma prevenção contra raios cósmicos. No caso o "low" e "high" rejects são valores multiplicados ao "σ" para definir os valores aceitáveis. A região de confidência de "3σ" representa 99% dos pontos.

Por fim, podemos tratar nossos dados. Vamos começar por obter o nosso "bias" e em seguida o nosso "flat". Aqueles que usam flats de vários filtros não terão dificuldade em extrapolar os procedimentos daquilo que dissemos até aqui, definindo de forma conveniente os "subsets".

```
cl> zerocombine @zero.lis
```

"zerocombine" e "flatcombine" são "scripts" especialmente escritos para se produzir os resultados desejados usando os parâmetros corretos para rodar a task "combine". Uma vez terminados teremos no nosso diretório os arquivos "Zero.imh" e "Flat.imh" que são resultados das combinações dos arquivos bias e flat-field. Teoricamente são arquivos "limpos" de todos os desvios que costumam acontecer normalmente nesses tipos de frames. Mas cuidado, se um ou mais arquivos originais não for "legal", isto é, não estiverem dentro de uma norma de um "bom" frame, a combinação estará completamente falseada. Dê uma olhada nesses arquivos antes da operação e não tenha compaixão de jogar fora os "replicantes" (ver filme "Blade Runner").

O próximo passo é lançar a correção de bias e flat field nos nossos arquivos dos objetos:

```
cl> ccdproc *.??h zerocor+ zero=Zero flatcor-
```

O primeiro comando diz para pegar todos os arquivos iniciando com "obj" e admintindo apenas aqueles cuja extensão tenha somente 3 caracteres e terminem com 'h'. Essa é uma forma de fazer reconhecer os nomes dos arquivos que tenham ou ".imh" ou ".hhh" em sua extensão. A primeira forma é a do NOAO e a segunda é do "STScI". O sinal '*' indica "qualquer coisa com qualquer quantidade de letras", enquanto o "?" indica "qualquer coisa com apenas 1 letra". Como disse anteriormente, mudo o parâmetro "zerocor" fazendo-o verdadeiro enquanto imponho "flatcor" como sendo falso para forçar que seja feita apenas correção de "bias" num primeiro momento. Indico com o parâmetro "zero=Zero" que o bias seja procurado no arquivo chamado "Zero.imh". Não esqueço de fazer a correção do bias também nos objetos de calibração e, importante, também no flat field.

Agora é fazer a correção de flat field:

```
cl> flatcombine @flat.lis
cl> ccdproc *.??h flatcor+ flat=Flat zerocor-
```

Terminamos aqui a redução da imagem CCD. Chegamos a 2 caminhos. Aqueles que vão trabalhar com espectros devem escolher os packages "twodspec" e posteriormente "onedspec". Os "fotometristas" partirão pelas estradas dos "focas", "digiphot" ou mesmo "stsdas". Leia os manuais e tente. Dê uma olhada nos "helps", "lpar" e "demos" se existirem. Com essa leitura você aprendeu como lidar com as diferentes facilidades do IRAF, suas dicas, manhas e forma de interação. Existe muito mais a descobrir. O que vimos foi apenas um começo.

6 Performance

Trabalhei numa Sparc II do ON-DAF. Precisei consultar algumas páginas do "conjunto azul", *helps on line* e algumas tentativas com os programas indicados. Desconta-se a parte de aprendizado que é manifestamente a mais longa. Uma vez apreendido, contei o tempo de modificações nos "headers" e edições dos parâmetros dos diferents programas, com o tempo de verificação. Somando com o tempo de execução das tasks, sem me apressar ou tentar bater *records* de tempo, cheguei a uma marca de 2 horas de tempo de interação com a máquina (sem contar as paradas para o café, um cigarro fumado pelos corredores e papos furados). Credite um dia para completar tudo.