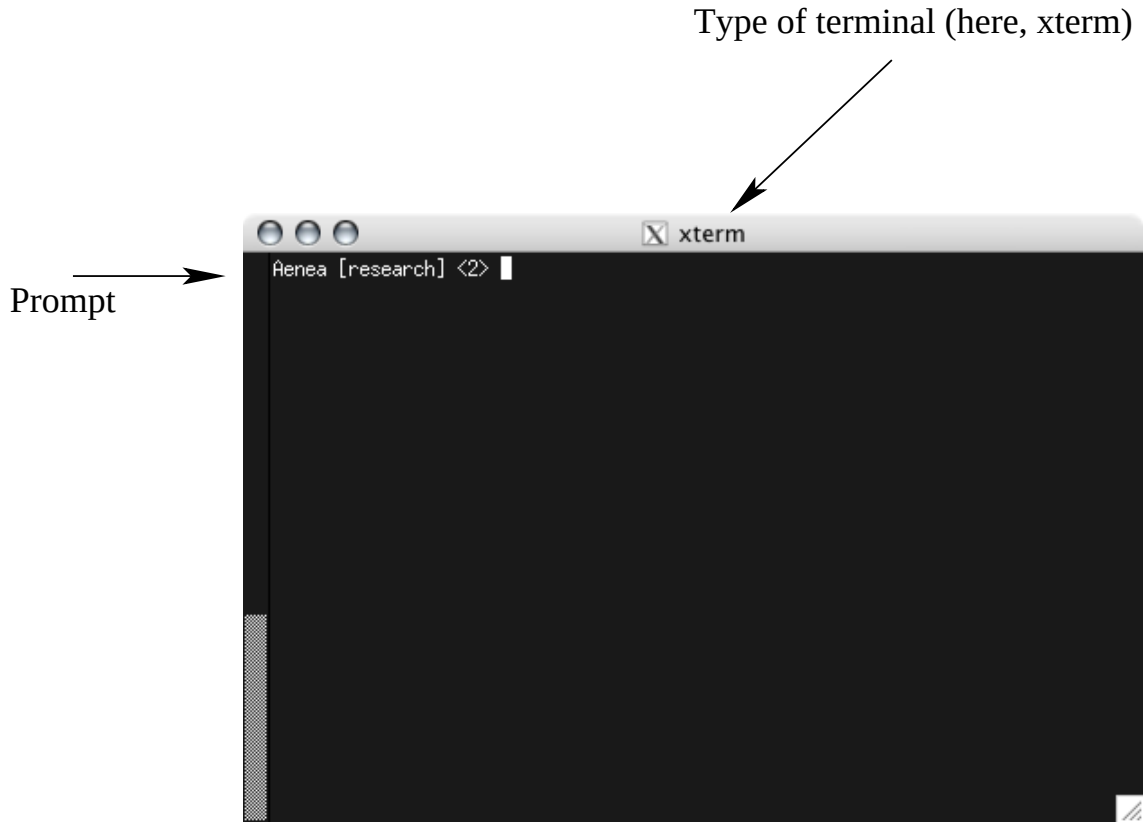
Working in the Command-Line Environment

After we acquire images of astronomical sources (using the Macalester Observatory, or any other telescope) we want to be able to do sensible things with these data. We interact with the computer (which can handle the millions and millions of computations needed to manipulate digital images) in the most straightforward manner as possible. This involves using *command line* computing; most people are used to interacting with the computer by clicking on buttons. Command line interaction accomplishes the same goals but allows for a much higher degree of freedom with your files.

In this course we will use two *terminals* to talk to our computers. The first is an *xterm*; an example *xterm* is shown in the following image:



Note that the terminal type is displayed at the top. The computer's *command line prompt* is also noted in this figure (here, it says "Aenea [research] <3>"). I have configured our computers such that the prompt line shows "PHYS440" and then a running number showing how many commands have been issued in this terminal window (so far, 2 in this example).

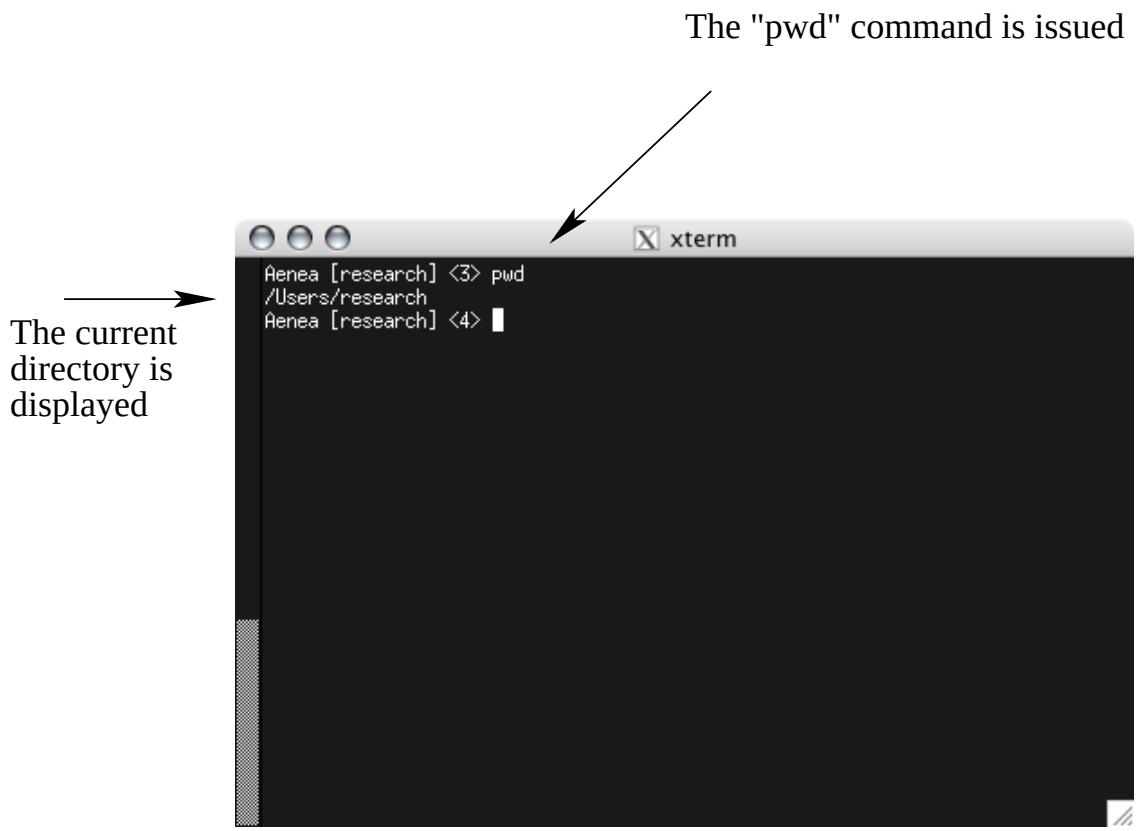
Like the folders structure of most computers you are familiar with, the research computers we will be using operate on a structure of *directories*. Each user has a *home area*; this is the location that you start out in. You can then navigate to any given directory with very simple commands that I detail below.

We will issue a very small number of commands in the *xterm*; I list explicitly each of these common commands and what each does below.

```
pwd:   returns current directory name
 ls:   lists contents of current directory
 cd:   change directories
 cp:   copy a file
 rm:   remove a file
 mv:   move a file to a new location or rename the file
```

By way of examples, let us use each command and see what it does:

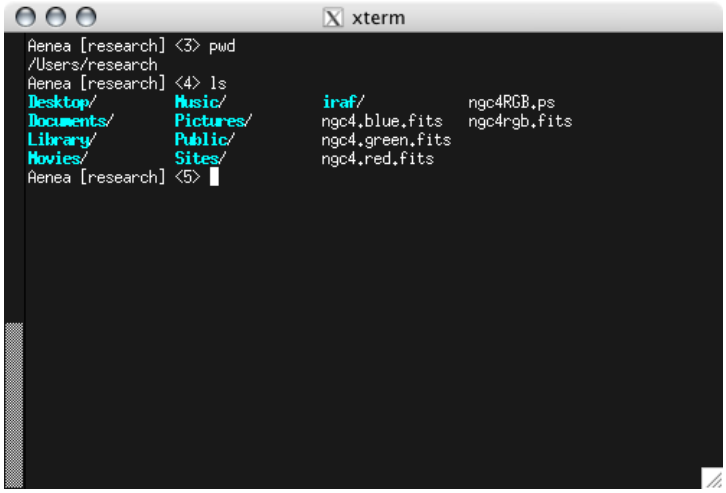
The "pwd" command:



The "pwd" command lists the current directory location (here, /Users/research).

The "ls" command:

The contents of the current directory are displayed

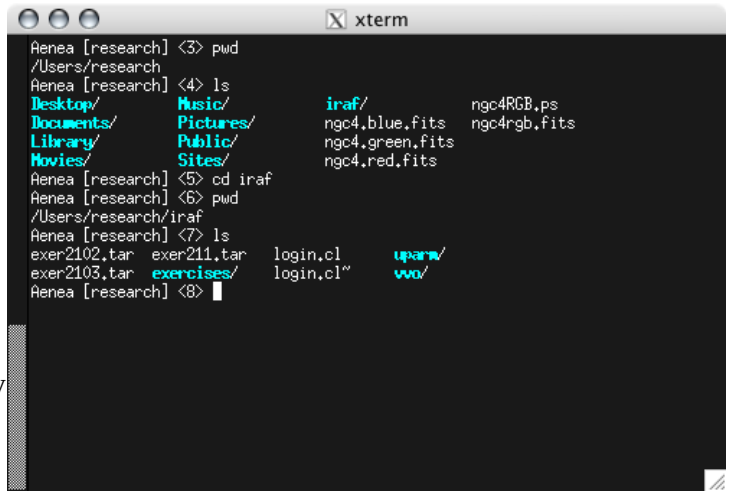


The "ls" command lists the contents of the current directory; note that subdirectories of the current directory are shown in a cyan color, while individual files are shown as normal text. The file extension (e.g., .ps, .fits), is indicative of the type of file it is.

Note in the above example that any subdirectories are shown in a cyan color, while individual files are shown as text. Here, we could change directories into any of the directories shown; we move into the "iraf" directory in the next step. The file *extension* (e.g., .ps, .fits, is indicative of the type of file it is. So, a ".ps" file is a postscript file, a ".fits" file is a FITS file, and so on.

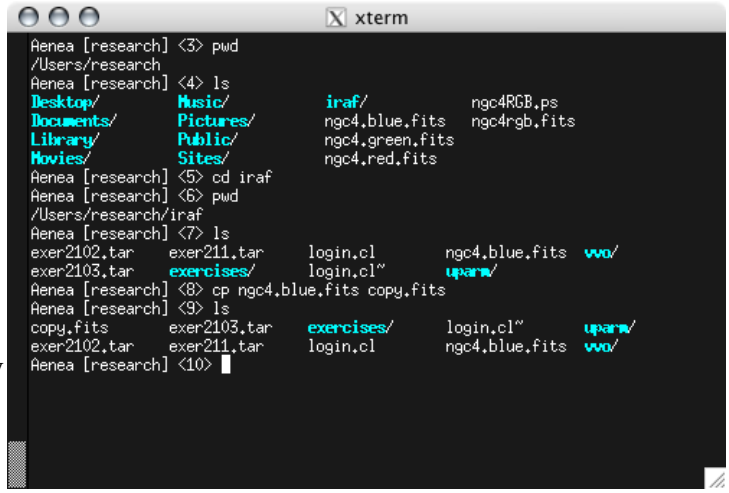
The "cd" command: Let's move into the "iraf" directory, check our current directory location with "pwd", and list the contents of the directory with "ls".

We changed directories, checked our current directory location, and listed its contents.



We changed directories, checked our current directory location, and listed its contents.

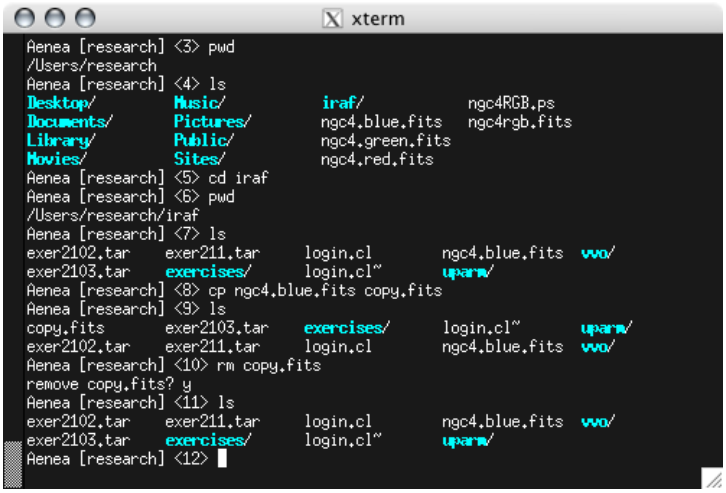
The "cp" command: To copy a file, we use the "cp" command and give a second value that is the name of the new (copied) file.



File copied; listing directory contents now shows this file.

We copied the file "rgb4.blue.fits" to a new file called "copy.fits".

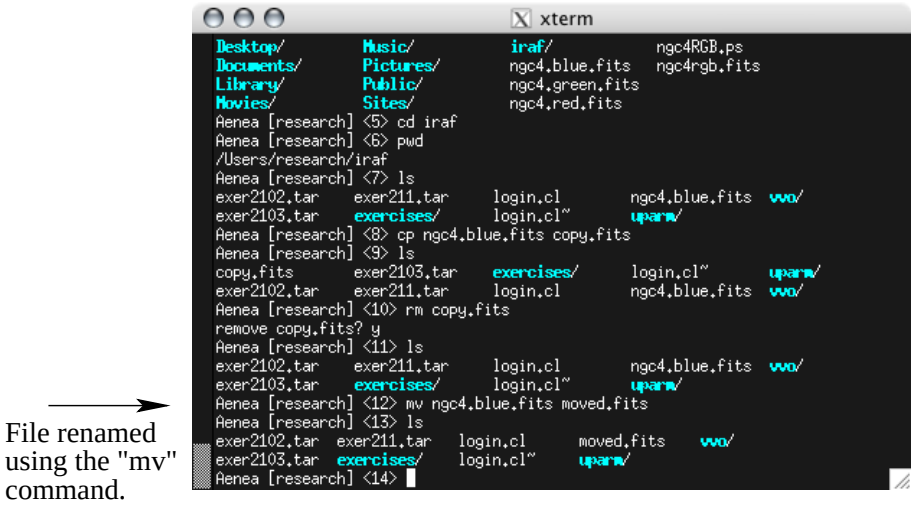
The "rm" command: To remove a file (permanently, so be careful!), we do the following:



File removed; listing directory contents now shows this file is removed.

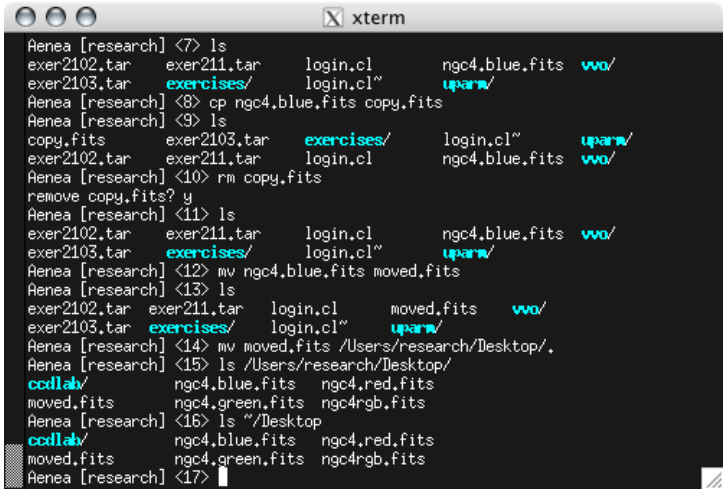
We removed a file; note that the system will prompt you to answer yes or no (answer "y" or "s") that you wish to actually delete something. Listing the directory contents allows us to see that the file is indeed gone.

The "mv" command: To rename a file (or move it to a new directory location), use the "mv" command. If moving to a new directory, append the directory name to the output name.



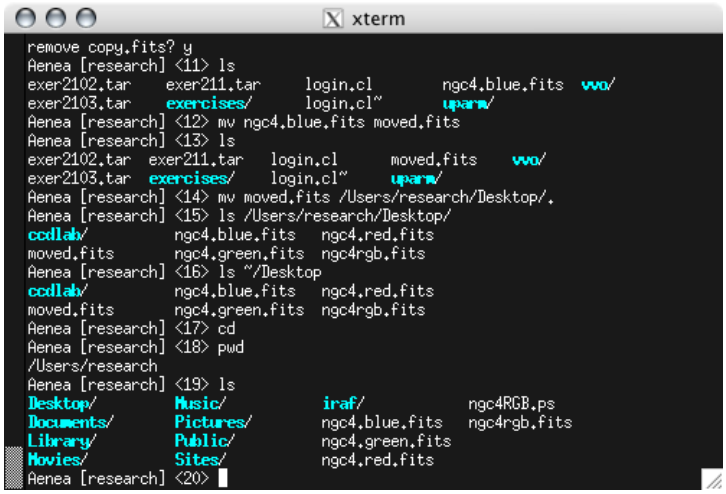
We renamed "ngc4.blue.fits" to "moved.fits" via the "mv" command.

With any command, you can specify a file at any location in the directory tree. By way of example, let's move this new image ("moved.fits") to the Desktop. Then, let's list the contents of that directory without moving there:



We moved the file "moved.fits" to the Desktop and listed the contents of that directory without moving there. Note that, since the home directory is /Users/research/, we can abbreviate this using the ~ command.

Finally, let's return to our home directory by issuing a "cd" command without any arguments.



We moved to our home directory by using the "cd" command without any arguments.

The FITS Image Format

— Some information drawn from the NASA/HEASARC website at http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html

FITS stands for 'Flexible Image Transport System' and is the standard astronomical data format endorsed by both NASA and the International Astronomical Union. FITS is much more than an image format (such as JPG or GIF) and is primarily designed to store scientific data sets consisting of multi-dimensional arrays (1-D spectra, 2-D images or 3-D data cubes) and 2-dimensional tables containing rows and columns of data.

A FITS file consists of one or more Header + Data Units (HDUs), where the first HDU is called the 'Primary HDU', or 'Primary Array'. The primary array contains an N-dimensional array of pixels, such as a 1-D spectrum, a 2-D image, or a 3-D data cube. Five different primary data types are supported: unsigned 8-bit bytes, 16 and 32-bit signed integers, and 32 and 64-bit single or double precision floating point reals. FITS can also store 16 and 32-bit unsigned integers.

Any number of additional HDUs may follow the primary array; these additional HDUs are called FITS 'extensions'. There are currently 3 types of extensions defined by the FITS Standard:

```
Image Extension - a N-dimensional array of pixels, like in a primary array
ASCII Table Extension - rows and columns of data in ASCII character format
Binary Table Extension - rows and columns of data in binary representation
```

Every HDU consists of an ASCII formated 'Header Unit' followed by an optional 'Data Unit'. For historical reasons, each header or data unit must be an exact multiple of 2880 bytes long. Any unused space at the end of the header or data unit is padded with fill characters (ASCII blanks or NULs depending on the type of unit).

Each header unit consists of any number of 80-character keyword records which have the general form:

```
KEYNAME = value / comment string
```

The keyword names may be up to 8 characters long and can only contain uppercase letters, the digits 0-9, the hyphen, and the underscore character. The keyword name is (usually) followed by an equals sign and a space character (=) in columns 9 - 10 of the record, followed by the value of the keyword which may be either an integer, a floating point number, a character string (enclosed in single quotes), or a boolian value (the letter T or F).

The last keyword in the header is always the 'END' keyword which has no value or comment fields. There are many rules governing the exact format of a keyword record (see the FITS Standard for details) so it is generally better to rely on standard interface software to correctly

construct or parse the keyword records rather than directly reading or writing the raw FITS file.

Each header unit begins with a series of required keywords that specify the size and format of the following data unit. A 2-dimensional image primary array header, for example, begins with the following keywords:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  440 / length of data axis 1
NAXIS2  =                  300 / length of data axis 2
```

The required keywords may be followed by other optional keywords to describe various aspects of the data, such as the date and time of the observation. Other COMMENT or HISTORY keywords are also frequently added to further document the contents of the data file.

The data unit, if present, immediately follows the last 2880-byte block in the header unit. Note that some HDUs do not have a data unit and only consist of the header unit.

The images we will work with in this course are 2-dimensional imaging arrays with header information. Our CCD camera acquires images with 1530×1020 pixels; an example header file from one such image is as follows:

```
aug16_12.fits[1530,1020][ushort]: Uranus
No bad pixels, min=0., max=65535.
Line storage mode, physdim [1530,1020], length of user area 1418 s.u.
Created Thu 21:01:00 16-Aug-2007, Last modified Fri 00:56:03 17-Aug-2007
Pixel file "aug16_12.fits" [ok]
OBJECT  = 'Uranus  '           /
TELESCOP= 'DFM CCT-16 16" Cassegrain'
INSTRUME= 'SBIG ST-8'
OBSERVER= '        '
DATE-OBS= '2007-08-17T05:54:45.000' / GMT START OF EXPOSURE
BZERO   = +3.276800000000E+004 /
BSCALE  = +1.000000000000E+000 /
EXPTIME = +1.000000000000E+001 / EXPOSURE IN SECONDS
CCD-TEMP= -1.693909657372E+001 / CCD TEMP IN DEGREES C
XPIXSZ  = +9.000000000000E+000 / PIXEL WIDTH IN MICRONS
YPIXSZ  = +9.000000000000E+000 / PIXEL HEIGHT IN MICRONS
XBINNING=                    1 / HORIZONTAL BINNING FACTOR
YBINNING=                    1 / VERTICAL BINNING FACTOR
XORGSUBF=                    0 / SUB_FRAME ORIGIN X POS
YORGSUBF=                    0 / SUB-FRAME ORIGIN Y POS
EGAIN   = +2.450000000000E+000 / ELECTRONS PER ADU
```

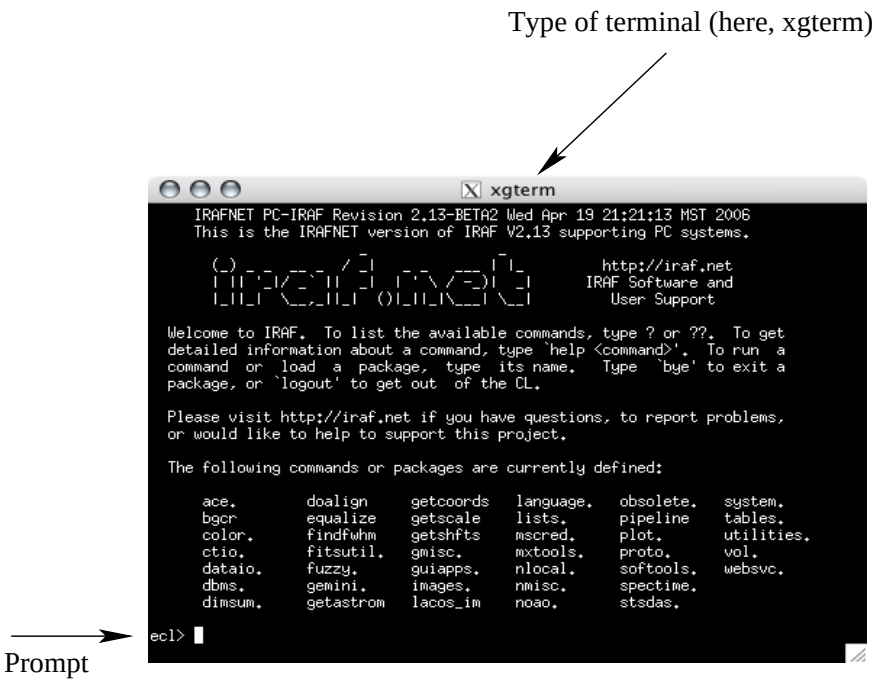
```
FOCALLEN= +3.251200000000E+003 / FOCAL LENGTH IN MM
APTDIA  = +4.064000000000E+002 / APERTURE DIAMETER MM
APTAREA = +1.114756480160E+005 / APERTURE AREA IN SQ-MM
CBLACK  =                  120 / BLACK ADU FOR DISPLAY
CWHITE  =                  201 / WHITE ADU FOR DISPLAY
PEDESTAL=                 -100 / ADD TO ADU FOR 0-BASE
DATAMAX =                65535 / SATURATION LEVEL
SBSTDVER= 'SBFITSEXT Version 1.0' / SBIG FITS EXTENSIONS VER
SWACQUIR= 'WinOPS Ver 5.44-NT' / DATA ACQ SOFTWARE
SWCREATE= 'SBIG Win CCDOPS Version 5.44-NT'
SWMODIFY= 'WinOPS Ver 5.44-NT'
HISTORY Auto Dark Subtraction
FILTER  = 'R       '           / OPTICAL FILTER NAME
SNAPSHOT=                    1 / NUMBER IMAGES COADDED
DATE    = '2007-08-17'         / GMT DATE WHEN THIS FILE CREATED
RESMODE =                    0 / RESOLUTION MODE
EXPSTATE= '125     '           / EXPOSURE STATE (HEX)
RESPONSE= +3.000000000000E+003 / CCD RESPONSE FACTOR
NOTE    = 'Local time:8/17/2007 at 0:54:45'
```

Much information about this particular image is contained in this header: target, exposure time, date and time of observation, etc., just to name a few. The actual image is contained in the FITS image primary array, which in our case is 1530×1020 pixels.

Basic Operations in IRAF

IRAF is a collection of computer routines that perform basic analysis operations on FITS images. At first use it can seem a bit finicky, but it allows astronomers to quickly and efficiently perform various *tasks* that would otherwise take hours and hours when done manually. We thus wish to exploit these capabilities.

Instructions for starting *IRAF* are given in the CCD imaging lab. Assuming you have started *IRAF* from the correct location, you will begin with a screen that appears as follows:



Your prompt is different in *IRAF* compared to in the *xterm*; it has changed to

```
ecl>
```

You can use the "pwd", "cd", and "ls" commands you learned above in *IRAF*. This allows you to see what images you will be able to work with. When you start *IRAF* you begin in the *IRAF home directory*, which is different than the home directory for your *xterm*. The *IRAF* home directory and its contents are shown with the now-familiar commands:

```
xgterm
 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.

 Please visit http://iraf.net if you have questions, to report problems,
 or would like to help to support this project.

 The following commands or packages are currently defined:

     ace.        doalign     getcoords   language.   obsolete.   system.
     bgcr        equalize    getscale    lists.      pipeline    tables.
     color.      findfwhm    getshfts    mscred.     plot.       utilities.
     ctio.       fitsutil.   gmisc.      mxtools.    proto.      vol.
     dataio.     fuzzy.      guiapps.    nlocal.     softtools.  websvc.
     dbms.       gemini.     images.     nmisc.      spectime.
     dimsum.     getastrom   lacos_im    noao.       stsdas.

ecl> pwd
/Users/cannon/iraf
ecl> ls
exer2102.tar     exercises/       login.cl~
exer2103.tar     fintro0001.fits  uparm/
exer211.tar      login.cl         vvo/
ecl>
```

Note that the *IRAF* home directory is different than the *xterm* home directory. Your *IRAF* home directory will be /Users/research/iraf/.

IRAF tasks are grouped into *packages* according to their function. In the above screen, there are various *packages* listed (these have periods at the ends of their names; e.g., ace., color., ctio., etc.); there are also a few *tasks* that are not associated with a *package* and these are listed without a period following their name (e.g., bgcr, doalign, etc.).

You move into a given *package* by issuing the name of that package. For example, to look at the *tasks* grouped into the *images* package, simply issue the command "images":

```
xgterm
      (_) _ _  __ _  / _|   _ _   ___ | |_        http://iraf.net
      | || '_|/ _` ||  _|  | ' \ / -_)|  _|      IRAF Software and
      |_||_|  \__,_||_|  ()|_||_|\___| \__|        User Support

 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.

 Please visit http://iraf.net if you have questions, to report problems,
 or would like to help to support this project.

 The following commands or packages are currently defined:

     ace.        doalign     getcoords   language.   obsolete.   system.
     bgcr        equalize    getscale    lists.      pipeline    tables.
     color.      findfwhm    getshfts    mscred.     plot.       utilities.
     ctio.       fitsutil.   gmisc.      mxtools.    proto.      vol.
     dataio.     fuzzy.      guiapps.    nlocal.     softtools.  websvc.
     dbms.       gemini.     images.     nmisc.      spectime.
     dimsum.     getastrom   lacos_im    noao.       stsdas.

ecl> images
     imcoords.   imfit.      immatch.    tv.
     imfilter.   imgeom.     imutil.

images>
```

A new set of *packages* is now displayed; let's select the "imutil" package:

```
xgterm
 package, or `logout' to get out  of the CL.

 Please visit http://iraf.net if you have questions, to report problems,
 or would like to help to support this project.

 The following commands or packages are currently defined:

     ace.        doalign     getcoords   language.   obsolete.   system.
     bgcr        equalize    getscale    lists.      pipeline    tables.
     color.      findfwhm    getshfts    mscred.     plot.       utilities.
     ctio.       fitsutil.   gmisc.      mxtools.    proto.      vol.
     dataio.     fuzzy.      guiapps.    nlocal.     softools.   websvc.
     dbms.       gemini.     images.     nmisc.      spectime.
     dimsum.     getastrom   lacos_im    noao.       stsdas.

ecl> images
     imcoords.   imfit.      immatch.    tv.
     imfilter.   imgeom.     imutil.

images> imut
     chpixtype      imdelete       imheader       imslice        listpixels
     hedit          imdivide       imhistogram    imstack        minmax
     hselect        imexpr         imjoin         imstatistics   sections
     imarith        imfunction     imrename       imsum
     imcopy         imgets         imreplace      imtile

imutil>
```

Note that *IRAF* accepts least-descriptive matches to call *tasks* and *packages*. So, in the above example, "imut" was sufficient to tell *IRAF* that we want the "imutil" package. "imu" would have been sufficient also; note, however, that "im" would not - this describes other available *packages* as well, and *IRAF* would be unable to discern which you are interested without further information (it would tell you this).

Now we see a number of *tasks*; we know these are not further levels of *packages* because there is no period after their names. Let's begin manipulating one task, "imstatistics". *Tasks* are controlled by a collection of *parameters*; each *task* has a specific set of *parameters*. We can *list* the parameters for a given *task* with the "lpar" command:

```
xgterm
     dbms.       gemini.     images.     nmisc.      spectime.
     dimsum.     getastrom   lacos_im    noao.       stsdas.

ecl> images
     imcoords.   imfit.      immatch.    tv.
     imfilter.   imgeom.     imutil.

images> imut
     chpixtype      imdelete       imheader       imslice        listpixels
     hedit          imdivide       imhistogram    imstack        minmax
     hselect        imexpr         imjoin         imstatistics   sections
     imarith        imfunction     imrename       imsum
     imcopy         imgets         imreplace      imtile

imutil> lpar imstat
       images = "flat[150:640,10:510]" List of input images
     (fields = "image,npix,mean,stddev,min,max") Fields to be printed
      (lower = INDEF)          Lower limit for pixel values
      (upper = INDEF)          Upper limit for pixel values
      (nclip = 0)              Number of clipping iterations
     (lsigma = 3.)             Lower side clipping factor in sigma
     (usigma = 3.)             Upper side clipping factor in sigma
   (binwidth = 0.1)            Bin width of histogram in sigma
     (format = yes)            Format output and print column labels ?
      (cache = no)             Cache image in memory ?
       (mode = "ql")
imutil>
```

Parameters

Values

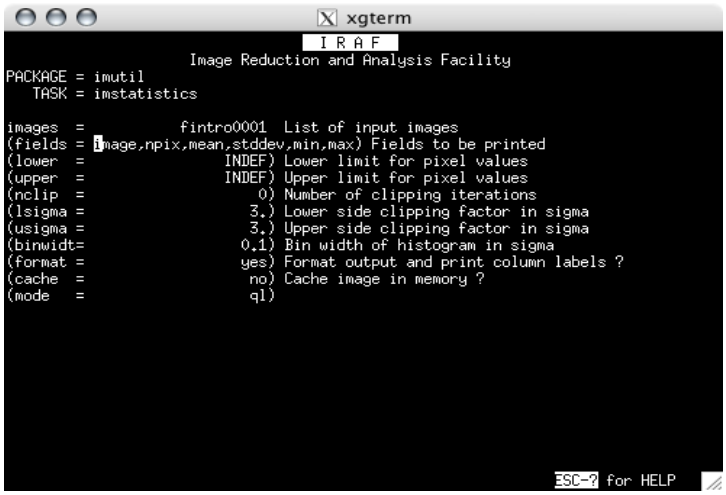
Brief Description

The "lpar" function allows us to examine the current values of parameters that a given *task* uses.

We can now set the values of these parameters as we see fit. Note from above that the image "fintro0001.fits" is in our current directory. We will input this image into the "imstat" *task* using the "edit parameters" or "epar" function, which is called as follows:

```
ecl> epar imstat
```

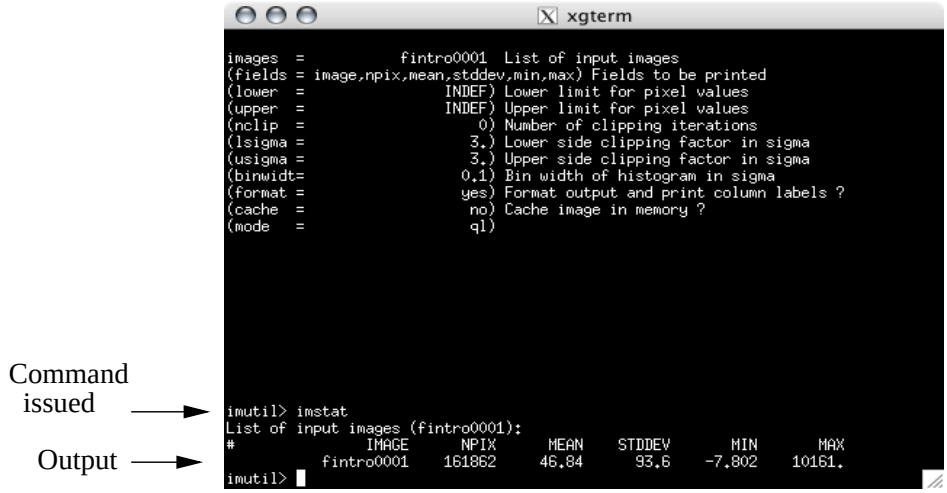
We now are presented with the *parameters* screen for the "imstat" *task*, where we can edit any field we like:



The "epar" function allows us to edit the current values of parameters that a given *task* uses. To move up and down between parameters, simply use the up and down arrow keys. To edit a field, simply move to that line and type in your new value. If you make a mistake in typing, the easiest recovery method is to move to another parameter field with the up or down arrow key, and then return to the parameter field you seek to edit and try again. Once you have set the parameters to your liking, you exit with the sequence ":q" (that is, colon, then the "q" key) and then hit return.

Here we have set the "images" *parameter* of the "imstat" *task* to be "fintro0001" (the .fits extension is not needed - *IRAF* is smart!). Note the instructions from the figure caption above: to move up and down between parameters, simply use the up and down arrow keys. To edit a field, simply move to that line and type in your new value. If you make a mistake in typing, the easiest recovery method is to move to another parameter field with the up or down arrow key, and then return to the parameter field you seek to edit and try again. Once you have set the parameters to your liking, you exit with the sequence ":q" (that is, colon, then the "q" key) and then hit return.

Once we exit the "epar" function, we return to the normal prompt. We can now *execute* this *task* by simply giving its *task* name at the command line:

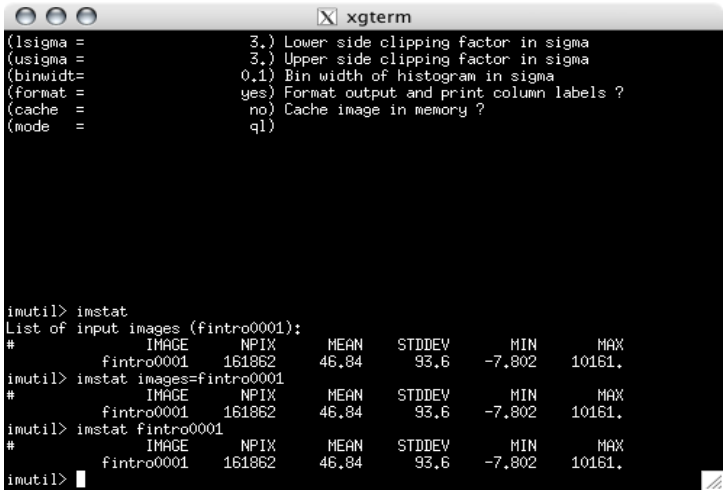


The "imstat" *task* is executed from the *IRAF* command line by simply typing the name of the *task*. The results of the *task* are printed (here, statistics on the image).

Note by comparing the above two images that *IRAF* will prompt you to verify any *parameter* value that is *not* contained in parentheses when using the "epar" function. This means that you do not need to use "epar" every time you use a *task* (though, until you have used a given *task* a number of times and are comfortable with how the various *parameters* affect the operation it is prudent to do so).

You can execute a given *task* without using the "epar" function, and without verifying any parameters at all. This is termed *scripting* and can greatly facilitate the processing of astronomical images. To use the "imstat" *task* in this mode, we simply issue the name of the *task* (exactly as above) and follow this with the values *IRAF* should use for each of the *parameters* not given in parentheses when using epar (explicitly, any *parameter* that is not *hidden*).

Let us demonstrate with an example. Examining the "epar" output for the *task* "imstat", we see that only the "images" *parameter* does not have parentheses around it. All of the other parameters are so-called *hidden* and will use a sensible default value. Thus, to run this function from the command line without using "epar", we simply supply the requisite values, as shown in the following image:



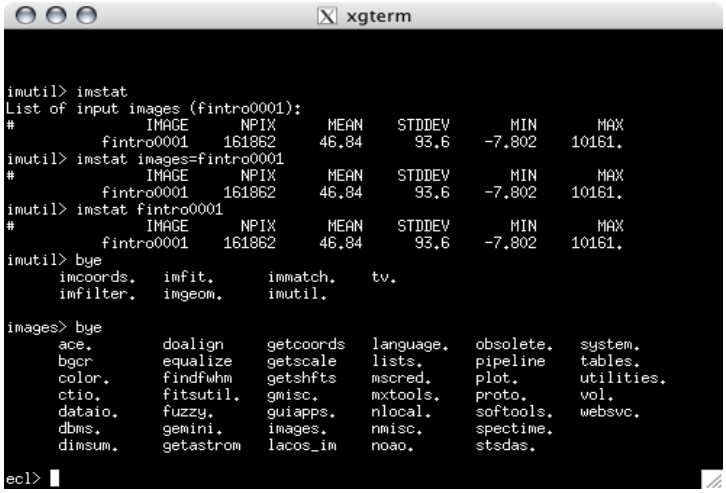
The "imstat" *task* is executed from the *IRAF* command line, without editing parameters in "epar" first. Note that the output is identical to our last application of the task. Further, note that you can do this one of two ways: one using a *parameter*=*value* nomenclature, and one using a *value*-only nomenclature.

Our outputs are exactly the same, which is the desired result. Note the instructions from the figure caption above: you can perform command-line *task* execution one of two ways: one using a *parameter*=*value* nomenclature, and one using a *value*-only nomenclature.

At times you will need to move between *packages*. To exit the current *package* (moving upward toward the top level of the *IRAF* package), simply issue the command

```
ecl> bye
```

Successive "bye" commands will get you back to *IRAF*'s starting level, from which you can locate any other task:

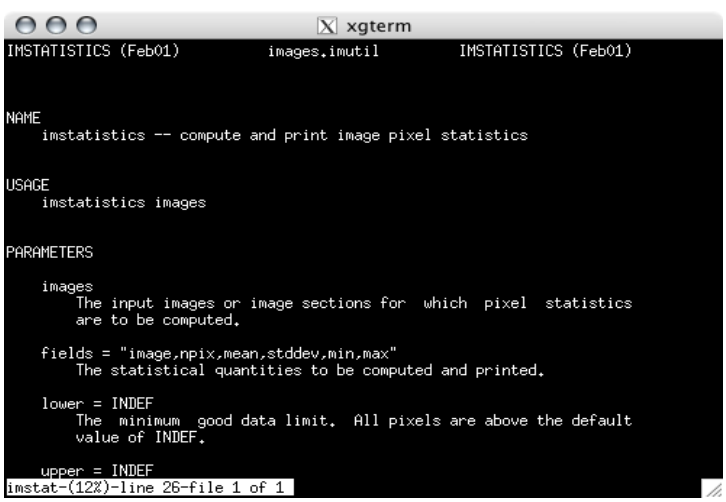


The "bye" command allows you to exit the current package, moving upward toward the top level of *IRAF*. From there you see the same screen you saw at the beginning, and you will be able to locate any other *task* you wish.

To obtain an explanation of a given task, issue the command

```
ecl> phelp taskname
```

where *taskname* is the name of the *task* you seek. For example, the help file for the "imstat" *task* begins as follows:



The beginning of the help file for the "imstat" *task*. Note from the bottom of the figure that this is only 12% of the file; you can page down with the space bar and upward with the "b" key; you exit from a help file using the "q" key. Note at the top of such a help file is the location within *IRAF* of the particular *task*; in this example, the "imstat" *task* help file states "images.imutil"; this is exactly the sequence of *packages* that we selected at the beginning in order to find the "imstat" *task*.

Finally, to exit *IRAF*, issue the simple command:

```
ecl> lo
```

Every *task* in *IRAF* can be executed using a similar approach as that used for the "imstat" *task* above.