Getting started

In order to get familiar with the **FITSH** package, we recommend to start your work somehow similar to the following.

Recommended auxiliary software packages

Indeed, **FITSH** is intended to process the images themselves, however, the package does not contain any utilities to display them or any additional tools for plotting or visualization. Hence, before and/or after installing **FITSH**, it is advised to get familiar with the following additional packages as well. Of course, you may choose any of your favorite packages or programming environments (for instance, Python or IDL has many of these additional features), but the examples featured in this website mainly expects that these ones are also available in your computer system. In addition these packages are not only free and open source packages but available for almost all of Linux, Mac/OSX or UNIX-like systems as well as on MS/Windows.

- bash [1]. The GNU Bourne-Again SHell, known as bash is one of the most well-known shells found on Linux or UNIX-like operating systems, and it is a de facto standard among the various kind of system shells. Although the **FITSH** tasks can be executed from any kind of environment, these tasks were designed to be run from bash. Many features of this shell is known by all of Linux or UNIX users, however, we recommend to get familiar with other features, such as conditional or iterative loops, variable definitions, arrays and function declarations as well. Please note that on certain systems, the default shell, `/bin/sh` not evidently refers the bash shell and hence some features (loops, arrays) work differently. In order to ensure the usage of bash, just start your executable scripts with `#!/bin/bash` instead of `#!/bin/sh` (see also the examples section here).
- Gnuplot [2]. Gnuplot is a portable command-line driven plotting utility and can be used to easily visualize the numerical data outputs of the **FITSH** tasks.
- DS9 [3]. The SAOImage/DS9 is the most widely used utility to display and analyze astronomical images that are stored in FITS format. Since the individual tasks of the **FITSH** package handles this format, we recommend this program to display images.
- XPA [4]. The X Public Access Mechanism (or XPA Messaging System) is the protocol via which external programs can communicate with DS9. If these utilities (mainly, the programs named `xpaset` and `xpaget`) are installed to the system, one can easily superimpose **FITSH** output to the FITS images directly. See e.g. the examples tvmark.sh or imexam.sh that also feature these.
- CDSclient [5]. The Centre de Données astronomiques de Strasbourg (CDS [6]) maintains the most widely used online astronomical catalogue service, the VizieR [7] service. Involving the utilities found in the CDSclient package, we can retrieve these catalogues directly to our system using command line tools.
- wget [8]. Most of the essential features of the CDSclient package can be implemented using purely the `wget` utility and some bash scripting. In the examples found in this webpage, we present scriptlets that access CDS/VizieR services using purely `wget` - so, strictly speaking, it is not necessary to install CDSclient -- but it is highly recommended since there are many nice features there which can be useful in your research.
- AWK [9]. The bash scripts presented among the **FITSH** examples frequently involves AWK scriptlets for various purposes.
- GNU/Coreutils [10]. These examples also uses some tasks from the GNU/Coreutils package, namely programs like `sort` or `uniq`.
- pexec [11]. Normally, each **FITSH** task exploits only a single processor core. Therefore, on multi-processor or multi-core architectures, it is rather effective to do computationally expensive tasks -- image registration, source extraction, image convolution or differential photometry -- in parallel. The utility `pexec` is a good choice to easily replece shell iterators and loops (`for ... in ...; do ...; done` or `while ...; do ...; done`) into a parallel form.

From the above list, bash, gnuplot, wget, (g)awk, GNU/coreutils and pexec are available also as parts of recent Linux distributions and also packaged for other unices (e.g. Mac/OSX, NetBSD) as well. Just use your favorite package manager to install them. The other tools, DS9, XPA, and the CDSclient package can easily be downloaded from their websites. DS9 is available as pre-compiled binary executable, while for the other ones, we can retrieve the source code (which must be compiled and installed thereafter).

Shell tricks

Before proceeding to more complex examples, we recommend to get a bit familiar with the following shell features:

- Input redirection, output redirection and pipeline:
  - `command ... > command.out`,
  - `command ... < command.in`,
  - `command1 ... | command2 ...`
- Shell loops:
  - Iterators: `for ... in ...; do ...; done`
  - conditional: `while ...; do ...; done`
- Shell arrays: `arr=(first second last); echo ${arr[0]}, ${arr[1]}, ...`

Here are some links to introduce bash programming:

- BASH Programming - Introduction HOW-TO [12] (by Mike G.)
- Advanced Bash-Scripting Guide [13] (by Mendel Cooper)

A few initial commands

To test that the **FITSH** package has been installed successfully and/or make the ``first steps *and get familiar with the **FITSH** concept, try the following commands. Just type (or copy/paste) them directly to the command prompt of a `bash` shell.*

1. Create an image

Let's create a small (128×128) image stamp, just with pure noise. The noise of the pixels is Gaussian, with the mean of 200 and standard deviation of 10. This is very similar to ``bias *images (images taken with zero exposure time). We use the task firandom for this purpose and the newly created image is then displayed with DS9:*

```
#!/bin/bash
firandom -s 128,128 -m 200 -d 10 -o noise.fits
ds9 -zscale noise.fits &
```

Many **FITSH** task options have a ``shorter *and `longer* forms. If we want to preserve somehow our scripting, it is advised to use the longer forms of command-line options:

```
firandom --size 128,128 --sky 200 --sky-noise 10 --output noise.fits
```

2. Drawing stars

Now let's create the same image with a point-like source, similar to a stellar image somewhere at the center of the image:

```
#!/bin/bash
echo 63.2 65.7 10000 3 0 0 | firandom -L - -s 128,128 -m 200 -d 10 -o star.fits
```

Here, the list of sources to be implanted to the image is read from the *standard input* of firandom since it is generated on-the-fly using the `echo` command. By default, this list should contain 6 columns at least: the centroid coordinates (here `x=63.2` and `y=65.7`), the total flux of the source (here `flux=100000`) and the shape parameters: FWHM (full width at half magnitude), the ellipticity of the profile and the elongation of profile ellipse. Here, the FWHM is 3.0 and the profile is circular (i.e. zero ellipticity).

3. Detecting point sources

The task fistar can be used to extract point-like sources from an image. Our star.fits image contains only one star which is (as one can see with, e.g. DS9) is a very prominent feature, so if this task is invoked without any tweaking, the source will safely be detected:

```
$ fistar star.fits
     1   64   66   63.152   65.771  1143.32   63.226   65.710   199.69   985.44 2.996 0.015 -84.8  0.618  0.009  0.002   10019.90
```

In order to make the output more verbose (i.e. see what is the meaning of the respective columns), one can turn on the ``output comment *option by -C or --comment:*

```
$ fistar -C star.fits
# Created by fistar 1.0rc5 (fi: 0.9.0)
# Invoked command: fistar -C star.fits
# Ident   IX   IY      C.X      C.Y    C.Max        X        Y       Bg      Amp  FWHM Ellip  P.A.      S      D      K       Flux
#  [ 1] [ 2] [ 3]     [ 4]     [ 5]     [ 6]     [ 7]     [ 8]     [ 9]     [10]  [11]  [12]  [13]   [14]   [15]   [16]       [17]
     1   64   66   63.152   65.771  1143.32   63.226   65.710   199.69   985.44 2.996 0.015 -84.8  0.618  0.009  0.002   10019.90
```

In addition, the format of the output can also be altered in order to list only the quantities in which we are really interested:

```
$ fistar -C -F x,y,flux star.fits
# Created by fistar 1.0rc5 (fi: 0.9.0)
# Invoked command: fistar -C -F x,y,flux star.fits
#        X         Y       Flux
#     [ 1]      [ 2]       [ 3]
   63.226   65.710   10019.90
```

By altering the star detection threshold, we can focus on only brighter sources. However, if the threshold is too low, some correlated but still pure noise pixels can also be characterized as stars:

```
$ fistar -F x,y,flux -f 5000 star.fits
   63.226   65.710   10019.90
$ fistar -F x,y,flux -f 20000 star.fits
$ fistar -F x,y,flux -f 300 star.fits
   87.337   26.238     293.14
   63.226   65.710   10019.90
    3.263  115.818     646.94
```

Using the tvmark script, one could directly use the output of the fistar task to mark the detected source(s) on the image opened previously by DS9:

```
$ ds9 -zscale star.fits &
$ fistar -F x,y,flux -f 5000 star.fits | tvmark.sh -c red -r 10
```

Note that in the above fistar-related examples, the output (i.e. the formatted list of detected sources) are not saved to any file: either displayed on the terminal screen (known as *standard output*) or fed to another program (in the last example with tvmark.sh). If the output should be saved (which is needed in most of the real scientific data processing), one can either redirect the output to a file (using bash redirection) or use the appropriate command line option (`-o` or `--output` for all of the **FITSH** tasks):

$ fistar -F x,y,flux -f 5000 star.fits > star.list $ fistar -F x,y,flux -f 5000 star.fits -o star.list

References

[1] http://www.gnu.org/software/bash/
[2] http://www.gnuplot.info/
[3] http://hea-www.harvard.edu/RD/ds9/site/Home.html
[4] http://hea-www.harvard.edu/saord/xpa/
[5] http://cdsarc.u-strasbg.fr/doc/cdsclient.html
[6] http://cdsweb.u-strasbg.fr/
[7] http://vizier.u-strasbg.fr/viz-bin/VizieR
[8] http://www.gnu.org/software/wget/
[9] http://www.gnu.org/software/gawk/
[10] http://www.gnu.org/software/coreutils/
[11] http://www.gnu.org/software/pexec/
[12] http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
[13] http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/