Scripting for Crystallography and Automation

Paul Emsley
LMB MRC
Aug 2013

Automation

- Is the use of control systems to optimize productivity in the production of goods and services
  - We can take that to mean plugging the munged output of one program into the the input of another
  - Do so a number of times and you've made a “pipe-line”

My Programming Background

- Early 80s
  - C-Shell script for automating CCP4 Amore
- Late 90s I started programming CHART
  - *(SOLVE* using CCP4 Software)
    - was not popular but I gained experience
- Mid 2000s:
  - HAPPy – “CCP4-blessed” re-write in Python
    - Not released, but I gained some experience
- *Coot*
  - Molecular graphics program designed to work with CCP4 (and other) software

Coot Introduction

- GNU GPL v3
- OpenGL (3D graphics) package for macromolecular model-building
  - *i.e.* making, adjusting and validating models of proteins
- 617k LoC
  - Mostly C++
    - heavy use of STL
  - scheme, python
- Most highly cited Free Software

Computation in Shell Scripts?

- I often use awk for “one-liners” in shell scripts
- If the awk program gets to be longer than straightforward, it occurs to me that *I'm doing it wrong* and will turn to rewrite in scsh or python

Coot Interfaces

- SHELX Interface
  - ins files
- Refmac Interface
  - raw parsing of log file
- PISA Interface
  - XML file parser
- PRODRG Interface
  - mdl files
- Phenix Interface
  - xmlrpc
- Mogul Interface
  - csv parser
- Wikipedia, Drugbank
  - synchronous XML
- PDBe
  - asynchronous web documents
    - including JSON

Parsing

- Very little to choose between mmCIF (PDBx) and XML.
  - XML advantage is that it can be done with scripting
  - Parsing of log files for (typically) single values is easier from log files than XML
    - (except for the “PEAKSEARCH” problem)

Inputs and Outputs

- standard in
- standard out
  - these can be typically redirected
- use pexpect where needed to automate interactive programs
  - a work-alike of Don Libes “expect”

Inputs and Outputs

```
program < input-file > log-file 2> errors-file
Data lines
<< eof-marker
```

or:

```
program << eof-marker > log-file 2> errors-file
Data lines
<< eof-marker
```

Bash Variable Substitution

```
hklin=data.mtz
FP=FP_native
PHI=PHWT

fft HKLIN $hklin << ! > fft.log
LABIN F1=$FP PHI=$PHI
!
```

Passing Parameters

```
$ bash fft.sh refined.mtz FWT PHWT
```

Bash Variable Substitution

```
hklin=data.mtz
FP=FP_native
PHI=PHWT

fft HKLIN $hklin << ! > fft.log
LABIN F1=$FP PHI=$PHI
!
```

Variable Substitution

```
hklin=$1
FP=$2
PHI=$3

fft HKLIN $hklin << ! > fft.log
LABIN F1=$FP PHI=$PHI
!
```

Conditions

- What happens if I run the previous script without specifying the phases?

```
$ bash fft.sh refined.mtz FWT
```

- Then `fft` gets run with this input:

```
LABIN F1=$FP PHI=
```

- Bad News. Let's test the number of arguments beforehand

Conditionals

```
if [ $# -ne 3 ] ; then
    exit
fi

hklin=”$1”
FP=”$2”
PHI=”$3”

fft HKLIN $hklin << ! > fft.log
LABIN F1=$FP PHI=$PHI
!
```

Executable Shell Scripts?

- No.
  - It just encourages:
  - Long winded command-line usage
  - a mismatch between command scripts and log files
  - So...
  - Use a script submitter to keep them consistent

```
iput=$1
oput=$2

if [ ! -e “$iput” ] ; then
   if [ ! -e $iput.com ] ; then
      echo sub.sh: The command file $iput does not exist.
      exit
   else
      iput=$iput.com
   fi
fi

if [ -z "$oput" ] ; then
   t1=$(basename "$iput")
   t2="${t1%.*}"
   oput=$t2.log
fi
(time bash $iput > $oput); stat=$?; echo $iput "has finished
  with status $stat" &
```

But but but...

- Shell scripting like this is often not what you want...
  - (not what I want)
- I want to combine process execution with computation and “non-trivial” judgement
- And this is better done with a general purpose language that can also control subprocesses
  - (albeit somewhat less elegantly)
- Python

But but but...

- Shell scripting is what people did before there was Python

Python (Shell) Scripting

```
from subprocess import call

def run_mogul(sdf_file_name, mogul_ins_file_name,
   mogul_out_file_name):

    f = make_mogul_ins_file(mogul_ins_file_name,
       mogul_out_file_name, sdf_file_name)
    if f:
       call(['mogul', '-ins', mogul_ins_file_name])
```

Python (Shell) Scripting

```
def make_mogul_ins_file(mogul_ins_file_name,
   mogul_out_file_name, sdf_file_name):
    f = open(mogul_ins_file_name, 'w')
    if f:
      f.write('mogul molecule file ')
      f.write(sdf_file_name)
      f.write('\n')
      f.write('mogul output   file ')
      f.write(mogul_out_file_name)
      f.write('\n')
      f.write('mogul output distribution all on\n')
      f.write('bond all\n')
      f.write('angle all\n')
      f.write('torsion all\n')
      f.write('ring all\n')
    return f
```

Network Communication

- PDB validation services will be available as XML files
  - interpretation is being built into Coot now
  - using python's built-in xml.etree
- Asynchronous communication
  - the outstandingly difficult task that I have tackled
    - why is it useful?
    - why is it hard?
  - Don't do it (unless you have to)

Tutorial Info

- We will be using python to get information from PDBe server in JavaScript Object Notation

Recommendations

- For Shell scripting
  - Scsh is the best (by far :)
  - Use sh (bash) not csh
    - bash is “Unix” POSIX standard
    - no functions
    - can't redirect standard error
    - can't read from redirected stdin
    - also quoting, signals, parsing, evaluation inelegances
  - Python for scripts that are more than “just” running processes

Recommendations

- Get to know your editor
  - to love it, even?
  - customize it
  - if it doesn't do paren matching and language-dependent commenting, choose something else...

Recommendations

- Do **not** write your own PDB parser
- Do **not** write your own crystallographic library
  - cctbx and mmdb/clipper are superb
  - (and it will take you ~5 years to begin to match the work therein)
- cctbx:
  - much functionality, and available for scripting
- mmdb/clipper:
  - easy to install, not pythonic (yet)

Software Recommendations

- For the unaffiliated
  - Core in C++
  - Scripting: python *via* boost.python
  - numpy for numerical library
  - Doxygen for documentation
  - GUI in Qt.

- *Coot*'s architecture (GNU heritage)
  - Core in C++
  - Scripting in scheme via SWIG
  - GNU Scientific Library of numerical libs
  - Texinfo for documentation
  - GUI in GTK+

Recommendations

- Use Revision Control
  - Subversion is the safe option
    - consider also bazaar, mercurial, git
    - Distributed VCS allow local commits
      - why is that good?
- If you have a GUI, you should routinely watch people using it
  - CSHL students since 2007

Take Home Message

- Rapid deployment
  - Often gets fixes out to those requesting them on the same day
  - requires:
    - revision control
    - automated testing
    - automated builds

Web Sites

- For programming queries:
  `stackoverflow.com`

“Release Early, Release Often”

- This is ridiculous
  - (in our field)
- Should be:
  - “Release when it's 'done done', release often”

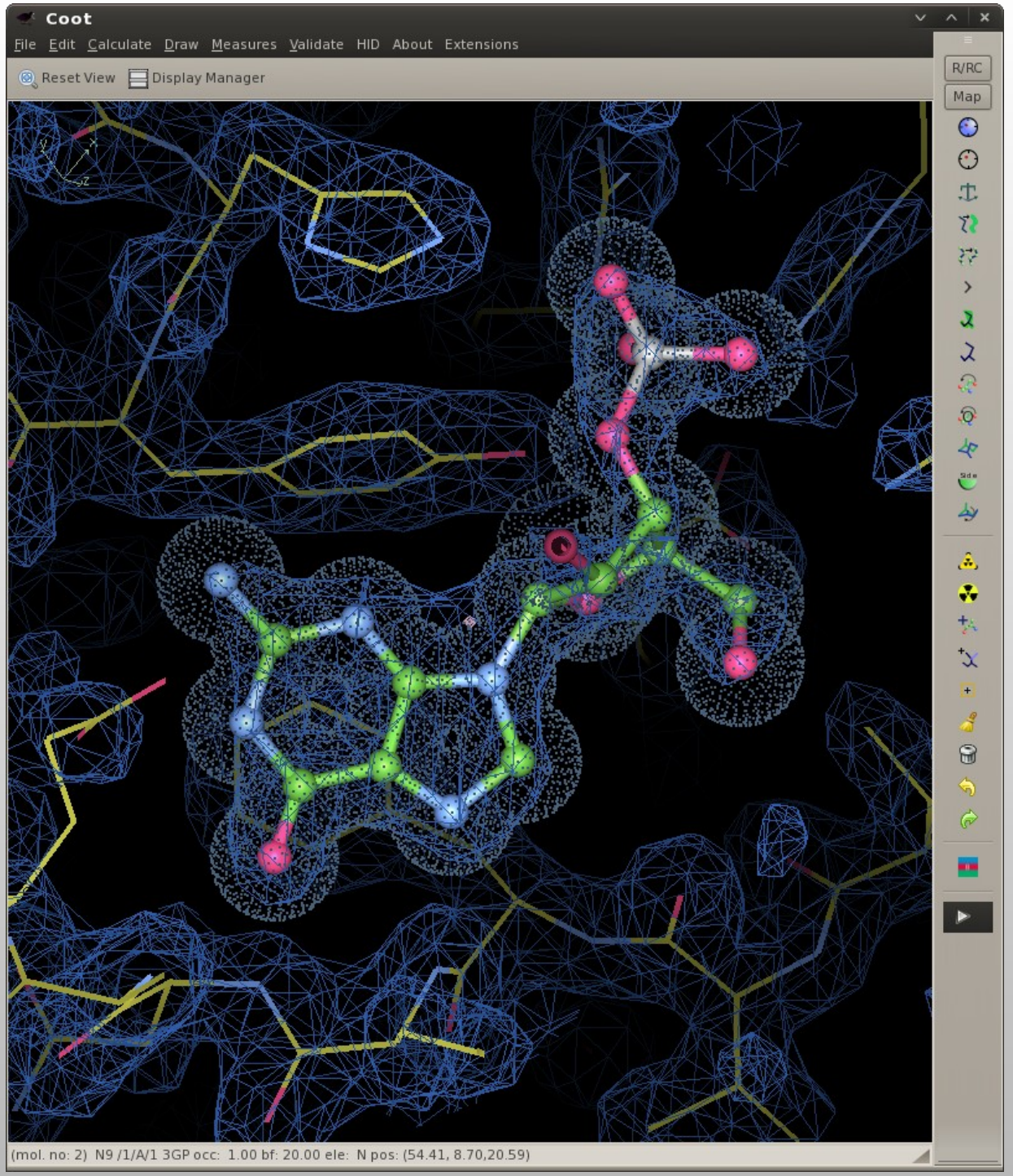
Thank you

Parallelization Considerations

- On multi-cored hardware
  - multiple thread, one process
  - paralellize the very inner level
- On cluster
  - (multiple processes, single-threaded)
  - parallelize the very outer layer
    - the batch submission layer

Scripting for Clusters

- It is convenient to have executables, data, input and output files for cluster jobs in “the same” location available from a server
- However, this can cause file-server bottlenecks
- Steps need to be taken to reduce this (reducing convenience)
  - cloning the database
  - cloning the software installation
  - random delay in execution
  - use the local file system and copy results back


Coot
File Edit Calculate Draw Measures Validate HID About Extensions
Reset View
Display Manager
R/RC
Map
(mol. no: 2) N9 /1/A/1 3GP occ: 1.00 bf: 20.00 ele: N pos: (54.41, 8.70,20.59)