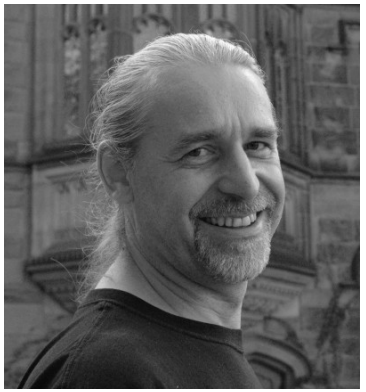
drPACS:
A Unix Execution Pipeline



Peter TEUBEN

Astronomy Department
University of Maryland

Abstract

We describe a very simple yet flexible and effective pipeliner for Unix commands. It uses a Makefile (behind the scenes) to define a serial set of commands for your choice of the pipeline. The pipeline commands share a common set of parameters by which they can communicate. Pipeline parameters can optionally be made persistent across multiple runs of the pipeline. Commands must follow a simple convention to retrieve and store parameters. Tools were added to simplify running a large series of pipelines, which can then also be run in parallel.
An updated version of a GUI script extender, tkrun, is also described.

INTRODUCTION

Imagine a script or procedure that runs a simulation or reduces data. If you have made it flexible, it will have parameters to control this process.

But now imagine you have to run many many (100s, 1000s) of these scripts, and potentially have to re-run them when something in your pipeline has changed (e.g. one of the programs in your pipeline has a bug). Or make the pipeline longer. Or insert steps in the pipeline. Etc.etc.

I'll describe a generic Unix (so it works on your Mac as well) solution to this dilemma, to make this process easy.

We have used this pipeline in CARMA for PACS data reduction, as well as converted an existing CARMA data reduction pipeline for the STING project.

Theory

Unix pipe:

```
% grep ^tshirt= mdjo10 | sort | uniq -c

% grep ^lunch= mdjo10 | awk -F= '{sum+=$2} END {print sum}'
```

PYTHON

```
>>> f = os.popen("grep ^lunch= mdjo10","r")
>>> lines=f.readlines()
>>> sum=0
>>> for line in lines:
>>>    sum = sum + int(line.split("=")[1])
>>> print sum
```

Theory (2)

```
% P1   p1_key1=val   p1_key2=val ...
% P2   p2_key1=val   ...
...
```

- Unix programs “P1”, “P2”, ... form the pipeline
- Each program has their own set of keywords
- Programs can pass information along in the pipe

$$(o_{11},o_{12},\ldots) = f_1(p_{11},p_{12},\ldots)$$
$$p_{21} = g(o_{11},o_{12},\ldots)$$
$$(o_{21},o_{22},\ldots) = f_2(p_{21},p_{22},\ldots)$$

drPACS

- d=dalton r=roger PACS=phase correction scheme
- Code in CVS (“**cvs checkout drpacs**”)
- Uses “configure” to install
  - **configure ; make ; make install**
  - **source drpacs_start.csh**
- Minimal dependencies (csh, python, sh)

Example: disk_*

(needs NEMO)

- disk_init — Initialize disk for simulation
  - nbody=250
  - rcut=1.25
  - Qtoomre=1
- disk_run — Run simulation
  - tstop=10
- disk_check — Check some results

Simple “pipe” commands

- **pipeline** : create a Pipefile (a Makefile)
- **pipepar** : get and set pipeline parameters
- **pipe** : run the pipeline
- **piperun** : run the pineline in set of directories
- **pipesave** : save pipeline pars
- **pipesetup** : grab previously save pipeline pars

Typical Session

Step 1: generate the control Pipefile (hint: it's a Makefile)

```
pipeline $DRPACS/cat/pipeline.001 > Pipefile
pipeline 4 step1 step2 step3 step4 > Pipefile
```

Step 2: set some parameters

```
pipepar -c
pipepar -c project=c0184.3B_108PG2130.13 carmaRefant=2
pipepar interval=5 carmaRefant=3
```

Step 3: run the pipe

```
pipe all
pipe step2
pipe all
pipe clean
```

Running a pipeline in a set of directories

Method 1: old-style c-shell method

```
foreach dir (`cat dirs.txt`)
  pushd $dir
  pipepar foo=3.3
  pipe step3 all
  popd
end
```

Method 2: using piperun

```
piperun dirs.txt 'pipepar foo=3.3 ;  pipe step3 all'
```

Examples

Scatterplot from a set of directories

```
piperun dirs.txt pipepar -v foo -v bar > foo_bar.tab
tabplot foo_bar.tab
```

Parallel pipe

```
piperun -n 4 dirs.txt pipe step2 all
```

PACS, in parallel

```
piperun -n 4 -c -o pipe.log dirs.txt 'pipepar -c project=%s showPlots=False;
            pipe clean all'
```

Persistent Pipes

```
pipesetup a=1 b=2 project=test
pipe all
pipesave
```

csh example

```
#! /bin/csh -f
#
#   (1) define default values in case not given
set a=1
set b=2
#  (2) pipeline interface to grab old defaults
pipepar -s csh > tmp$$.par;  source tmp$$.par;  rm tmp$$.par
#  (3) poor man's command line processor to override parameters
foreach _arg ($*)
 set $_arg
end
#  (4) The Actual Code where the work can be done
echo A=$a   B=$b
#  (5) write pipeline parameters back
pipepar a=$a b=$b c=3
```

python example

```
#! /bin/env python
#
import parfile, sys
a=1
b=2
if __name__ == "__main__":
  p = parfile.Parfile('drpacs.def')
  p.argv(sys.argv)
  p.set('a',a)
  if p.has('b'):
     b = p.get('b')
  else:
     p.set('b',b)
  p.set('sum',a+b)
  p.save()
```