PYTHON

PIPELINES

HI, I'M ANDY.

» ~20 years of coding

» I code for:

  » fun

  » profit

  » non-profit

  » you

TYPE LESS.
DO MORE.

TODAY:

» Make commands

» Connect them

» Make a library

» Use a library

Intros to Unix & Python see:
http://jura/bio/education/hot_topics

SIMPLE EXAMPLE

```
samtools view -bS test1.sam > test1.bam
samtools sort test1.bam test1.sorted
samtools index test1.sorted
```

REPEATED

```
samtools view -bS test1.sam > test1.bam
samtools sort test1.bam test1.sorted
samtools index test1.sorted

samtools view -bS test2.sam > test2.bam
samtools sort test2.bam test2.sorted
samtools index test2.sorted

samtools view -bS test3.sam > test3.bam
samtools sort test3.bam test3.sorted
samtools index test3.sorted
```

A LOT

```
samtools view -bS test1.sam > test1.bam
samtools sort test1.bam test1.sorted
samtools index test1.sorted

samtools view -bS test2.sam > test2.bam
samtools sort test2.bam test2.sorted
samtools index test2.sorted

samtools view -bS test3.sam > test3.bam
samtools sort test3.bam test3.sorted
samtools index test3.sorted

samtools view -bS test4.sam > test4.bam
samtools sort test4.bam test4.sorted
samtools index test4.sorted

samtools view -bS test5.sam > test5.bam
samtools sort test5.bam test5.sorted
samtools index test5.sorted

samtools view -bS test6.sam > test6.bam
samtools sort test6.bam test6.sorted
samtools index test6.sorted

samtools view -bS test7.sam > test7.bam
samtools sort test7.bam test7.sorted
samtools index test7.sorted

samtools view -bS test8.sam > test8.bam
samtools sort test8.bam test8.sorted
samtools index test8.sorted

samtools view -bS test9.sam > test9.bam
samtools sort test9.bam test9.sorted
samtools index test9.sorted
```

MAKE THE COMPUTER DO THE REPETATIVE STUFF

CUSTOM COMMANDS

```
sam2bam.sh test1.bam
```

Turning your workflow into a command

* extract parameters
* parse arguments
* feedback
* error checking

PARAMETERS

test1 x 5

```
samtools view -bS test1.sam > test1.bam
samtools sort test1.bam test1.sorted
samtools index test1.sorted
```

PARAMETERS!

```
name = "test1"

samtools view -bS $name.sam > $name.bam
samtools sort $name.bam $name.sorted
samtools index $name.sorted
```

ARGUMENTS

```
name = "test1" # <-- take from commandline

samtools view -bS $name.sam > $name.bam
samtools sort $name.bam $name.sorted
samtools index $name.sorted
```

ARGUMENTS!

```
filename = $1 # bash first argument
name = `basename $filename .bam` # strip extension

samtools view -bS $name.sam > $name.bam
samtools sort $name.bam $name.sorted
samtools index $name.sorted.bam
```

CLEANUP & FEEDBACK

```
filename = $1 # bash first argument
name = `basename $filename .bam` # strip extension

samtools view -bS $name.sam > $name.bam
samtools sort $name.bam $name.sorted
samtools index $name.sorted.bam
rm $name.sam $name.bam
echo "Created $name.sorted"
```

SAFETY

```
filename = $1 # bash first argument
name = `basename $filename .bam` # strip extension

# Only continue if successful. (&&)
samtools view -bS $filename > $name.bam &&
samtools sort $name.bam $name.sorted &&
samtools index $name.sorted.bam &&
rm $filename $name.bam &&
echo "Created $name.sorted"
```

HANDLE MULTIPLE FILENAMES

```
for filename in "$@"; do # <!-- loops over arguments
  name = `basename $filename .bam` # strip extension

  # Only continue if successful. (&&)
  samtools view -bS $filename > $name.bam &&
  samtools sort $name.bam $name.sorted &&
  samtools index $name.sorted.bam &&
  rm $filename $name.bam &&
  echo "Created $name.sorted"
done
```

GOOD ENOUGH?

```
for filename in "$@"; do # <!-- loops over arguments
  name = `basename $filename .bam` # strip extension

  # Only continue if successful. (&&)
  samtools view -bS $filename > $name.bam &&
  samtools sort $name.bam $name.sorted &&
  samtools index $name.sorted.bam &&
  rm $filename $name.bam &&
  echo "Created $name.sorted"
done
```

GOOD ENOUGH?

```
#> sam2bam.sh test1.sam
#> sam2bam.sh a.sam b.sam c.sam
#> sam2bam.sh *.sam
```

Probably.

REASONS FOR PYTHON

» complex workflow
  conditional commands

» data structures & transformation
  parse & summarize output

» reusable components & libraries
  numpy, biopython, matplotlib

PYTHON

» Commands

» Strings

» Output

» Arguments

» Conditionals

RUN A COMMAND

```
#!/usr/bin/env python
from subprocess import call
call("samtools view -bS test1.sam > test1.bam")
call("samtools sort test1.bam test1.sorted")
call("samtools index test1.sorted.bam")
```

STRING INTERPOLATION

```
#!/usr/bin/env python
from subprocess import call
name="test1"
call("samtools view -bS %s.sam > %s.bam" % (name, name))
call("samtools sort %s.bam %s.sorted" % (name, name))
call("samtools index %s.sorted.bam" % (name))
```

STRING INTERPOLATION /W LOCALS

```
string1, string2 = "Hello", "World"
"%s %s" % (string1, string2)
# => "Hello World"

"%(key1)s %(key2)s" % {key1: "Learning", key2: "Python"}
# => "Learning Python"

"%(string1)s %(string2)s" % locals()
# => "Hello World"
```

STRING INTERPOLATION

```
#!/usr/bin/env python
from subprocess import call
name="test1"
call("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())
call("samtools sort %(name)s.bam %(name)s.sorted" % locals())
call("samtools index %(name)s.sorted.bam" % locals())
```

GATHERING ARGUMENTS

```
import sys
sys.argv # => ["script.py", "test1.sam", "test2.sam"]

sys.argv[0] # => ["script.py"]
sys.argv[1:] # => ["test1.sam", "test2.sam"]
```

CLEANING ARGUMENTS

```
import sys
import re

filenames = sys.argv[1:]
for filename in filenames:
  name = re.sub(".sam$","", filename)
    # "test1.sam" => "test1"
  ...
```

RUNS, BUT NOT SAFELY!

```
#!/usr/bin/env python
from subprocess import call
import sys # system libraries, like arguments (argv)
import re # regular expressions
filenames = sys.argv[1:]
for filename in filenames:
  name = re.sub(".sam$","", filename) # strip extension
  call("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())
  call("samtools sort %(name)s.bam %(name)s.sorted" % locals())
  call("samtools index %(name)s.sorted.bam" % locals())
  call("rm %(name)s.sam %(name)s.bam" % locals())
  #     ^^-- this will always run! DANGER!
```

CAPTURE EXIT CODE & OUTPUT

```
from subprocess import call
call(cmd) # no exit code, no output

from subprocess import Popen, PIPE
ps = Popen(cmd, shell=True, stdout=PIPE, stderr=PIPE)
exitcode = ps.wait()
stdout = ps.stdout.read().rstrip('\n')
stderr = ps.stderr.read().rstrip('\n')
# Ugh. So much typing.
```

LET'S MAKE A METHOD

MAKING OUR METHOD

```
def run(cmd, dieOnError=True):
  ps = Popen(cmd, shell=True,
             stdout=PIPE, stderr=PIPE)
  exitcode = ps.wait()
  stdout = ps.stdout.read().rstrip('\n')
  stderr = ps.stderr.read().rstrip('\n')
  if dieOnError && exitcode != 0:
    raise Exception("run failed." +
      " cmd:`%(cmd)s` exit: %(exitcode)s" % locals())
  return (exitcode, stdout, stderr)
```

USING OUR METHOD

```
...
run("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())
run("samtools sort %(name)s.bam %(name)s.sorted" % locals())
run("samtools index %(name)s.sorted.bam" % locals())
run("rm %(name)s.sam %(name)s.bam" % locals())
...
```

COMPLETE PIPELINE!

```
#!/usr/bin/env python
import subprocess
import sys # system libraries, like arguments (argv)
import re # regular expressions

def run(cmd, dieOnError=True):
  ps = Popen(cmd, shell=True,
             stdout=PIPE, stderr=PIPE)
  exitcode = ps.wait()
  stdout = ps.stdout.read().rstrip('\n')
  stderr = ps.stderr.read().rstrip('\n')
  if dieOnError && exitcode != 0:
    raise Exception("run failed." +
      " cmd:`%(cmd)s` exit: %(exitcode)s" % locals())
  return (exitcode, stdout, stderr)

filenames = sys.argv[1:]
for filename in filenames:
  name = re.sub(".sam$","", filename) # strip extension

  run("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())
  run("samtools sort %(name)s.bam %(name)s.sorted" % locals())
  run("samtools index %(name)s.sorted.bam" % locals())
  run("rm %(name)s.sam %(name)s.bam" % locals())
```

MAKE A LIBRARY!

```
#-------------------------------------
# file: cli_utils.py
import sys
import re
from subprocess import Popen, PIPE

def run(cmd):
  ps = Popen(cmd, shell=True, stdout=PIPE, stderr=PIPE)
  exitcode = ps.wait()
  stdout = ps.stdout.read().rstrip('\n')
  stderr = ps.stderr.read().rstrip('\n')
  return (exitcode, stdout, stderr)

#-------------------------------------
# file: greeter.py
from cli_utils import run
exitcode, stdout, stderr = run("echo 'hello'")
print stdout
```

COMPLETE PIPELINE
WITH OUR LIBRARY

```
#!/usr/bin/env python
from cli_utils import run # our little library
import sys # system libraries, like arguments (argv)
import re # regular expressions

filenames = sys.argv[1:]
for filename in filenames:
  name = re.sub(".sam$","", filename) # strip extension

  run("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())
  run("samtools sort %(name)s.bam %(name)s.sorted" % locals())
  run("samtools index %(name)s.sorted.bam" % locals())
  run("rm %(name)s.sam %(name)s.bam" % locals())
```

LOOK AT WHAT WE'VE DONE!

» running commands

» parameters and arguments

» methods

» our libraries

ADVANCED TOPICS

» argument parsing (argparse)

» conditions

» bio libraries

TINY EXAMPLE
BUT THE CONCEPTS SCALE!

George's RNASeq Pipeline

* For either single or paired reads
* Run quality control on short reads
* Drop low-quality reads
* Trim low-quality bases from end of reads
* After trimming, get reads that are still paired
* Map reads with tophat
* Sort and index mapped reads

ADVANCED ARGUMENT PARSING

```
...
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("-s","--sort", action="store_true")
parser.add_argument("-i","--index", help="requires sort", action="store_true")
args = parser.parse_args()

run("samtools view -bS %(name)s.sam > %(name)s.bam" % locals())

if args.sort:
  run("samtools sort %(name)s.bam %(name)s.sorted" % locals())

  if args.index:
    run("samtools index %(name)s.sorted.bam" % locals())
else:
  if args.index:
    print "WARNING! You must sort before you can index."
```

ADVANCED ARGUMENT PARSING!

```
sam2bam.py --help

optional arguments:
  -h, --help   show this help message and exit
  -s, --sort
  -i, --index  requires sort
```

LIBS: PYSAM
READ BAM FILES

```
import sys
import pysam
from Bio import SeqIO, Seq, SeqRecord

filename = sys.argv[1]
bam = pysam.Samfile(filename, 'rb')
for read in bam:
  print read.qname
```

LIBS: MORE

» Cogent - COmparative GENomics Toolkit

» SciPy - Stats, Graphs, Linear Algebra, ...

» Numpy - Matrix, Random Sampling, FFT, ...

» Biopython - PDBs, Phylogenetic trees, AlignIO, ...

BOOKS

» Learning Python - Lutz

» Managing Your Biological Data with Python - Via, Rother, Tramontano

» Python for Bioinformatics - Bassi

» Bioinformatics Programming - Model

» Python for Data Analysis - McKinney

» Bioinformatics Algorithms - Jones & Pevzner

THANK YOU.
WHAT DO YOU WANT TO BUILD?