Linux Basics II

Python by Example

Why Python?

- As an interpreted language with an extensive standard library makes it ideal for scripting
- Indented code is mandatory (makes for more readable code by default)
- Clean and concise syntax makes for improved readability
- Well suited for larger projects > 100 lines of code

Objective

Write a Python script to read a file, extract only the data from that file and store it another file. Write a bash script to use the Python script to extract the data for many files in a directory.

Python Scripting Language

Python is an easy to learn and a powerful object-oriented high-level programming language.

To meet our objective, we will use just a few of Python's features:

- Modules and File I/O
- Loops and control structures
- Regular expressions for pattern matching

Python Key Features

- Indenting is required
- Good support for arrays
- Different programming styles are supported
  - object-oriented, procedural, functional
- Large standard library

Python Help

Python man pages give you command line options

```
$ man python
```

Using python's internal help utility for any module, keyword or topic

```
$ python
Python 2.6.6 (r266:84292, Jun 18 2012, 09:57:52)
[GCC 4.4.6 20110731 (Red Hat 4.4.6-3)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> help()
....
help>
```

Python Libraries: Import

import modules from a Python library

For example:

```
import re, sys, os
```

re — Regular expression operations
os — Miscellaneous operating system interfaces
sys — System-specific parameters and functions

Python I/O: Open and Read

open() returns a file object, and is most commonly used with two arguments:

```
fileobj = open(filename, mode)
```

mode = "r" for reading, "w" for writing, "a" for appending
if omitted, default if "r"

Reading lines by looping over the file object is memory efficient, fast, and leads to simpler code. For example:

```
f = open("datafile","r")
for line in f:
    print line
```

Python I/O: Write and Close

f.write(*string*) writes the contents of *string* to the file object, **f**, if it was opened for writing. For example:

```
fout = open("datafile.new", "w")
fout.write("New data")
```

f.close() will close the file opened as file object, **f**, and free up any system resources taken up by the open file. Any attempts to use the file object will automatically fail after it has been closed. For example:

```
fout.close()
```

Python Project

Data files and scripts

Get Exercises (Mills account)

1. If you have an account on the Mills cluster, use SSH to connect

   ```
   ssh -Y username@mills.hpc.udel.edu
   ```

2. Copy the exercise directory `mlbII-python` into your home directory and change to it.

   ```
   cp -r ~trainf/mlbII-python $HOME
   ```

Get Exercises (wget)

1. If you do not have an account on the Mills cluster, then download the exercise file `mlbII-python.tar.gz` using `wget` into your home directory.*

```
cd $HOME
wget http://www.udel.edu/it/research/files/cluster/workshop/mlbII-python.tar.gz
```

2. Untar and uncompress the exercise file to create the `mlbII` directory and change to it.

```
tar -zxvf mlbII-python.tar.gz
cd ~/mlbII-python
```

* Note `wget` is available on most Gnu/Linux distributions.

Sample data file

```
# SAVED BY:
# DATE:
# TIME:
…
…
# COMMENTS:
# DATA:
0.0000E+0 0.0000E+0
… …
```

Python script: data_extract.py

```
#!/usr/bin/python
#
# usage:
# python data_extract.py source
#

import re, sys, os
datdest=sys.argv[1] + '.dat'

# open data source
fsrc=open(sys.argv[1], "r");

# tmp file
fdat=open(datdest, "w");
```

data_extract.py (cont'd)

```
begin_extraction=0
for line in fsrc: # read through each line in fsrc
     if begin_extraction==1:
         # begin writing data to destination
         fdat.write(line)
     elif re.match("# DATA", line):
         # match the beginning pattern of data section
         begin_extraction=1
     else:
         pass

fsrc.close() # Don't forget to close files
fdat.close()
```

Testing data_extract.py

```
$ cat datafile
      ... Display contents of datafile file ...
$ python data_extract.py datafile datafile

or

$ chmod +x data_extract.py
$ ./data_extract.py datafile datafile
$ ls
      ... Display list of files  ...
```

Bash script: extract_all_data

```
#!/bin/bash

for file in ./many_datafiles/*
do
   # If data_extract.py is executable otherwise use
   # python data_extract.py ${file}
   ./data_extract.py ${file}
done
```

Testing extract_all_data

```
$ ls -la many_datafiles
      ... List of files in many_datafiles directory ...
$ source extract_all_data
$ ls -la many_datafiles
      ... List of files in many_datafiles directory ...
      ... Note the new .dat files
      ... Note .datafile does not have .datafile.dat
```

Exercise

Python Tutorial

1. The Python Tutorial

   This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style.

- http://docs.python.org/tutorial/index.html