Mixing Languages

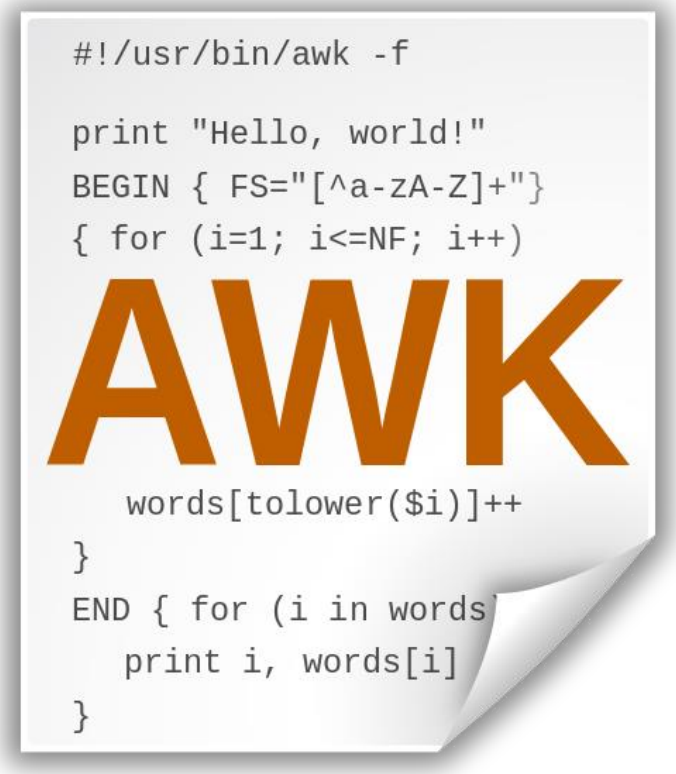
Benjamin Brewster

Scripting Languages

- The bash shell scripting language is not the only standard UNIX scripting language
- We can mix all of these languages and programs together!
- The only other always-built-in scripting language for a UNIX system is `awk`

awk

- awk was invented by
  - Alfred Aho
  - Peter Weinberger
  - Brian Kernighan
- It is commonly used for writing one-line programs on UNIX systems
- Popular early on because it adds computational ability to the command line
  - But now-a-days, we can do this directly in bash with e.g. $(( ))



awk example

- Hello world in awk

```
#!/usr/bin/awk -f
BEGIN { print "Hello, world!"; exit }
```

Associative Arrays

- `awk` features a kind of array called an associative array
- A normal array maps numbers to arbitrary objects (i.e., whatever you pick)
- Here are examples of a normal array mapping integer indexes to strings:
  - 6 maps to "jones"
  - 2 maps to "Nahasapeemapetilon"

Associative Arrays

- An associate array maps arbitrary objects to arbitrary objects
- Here is an example of mapping strings to other strings:
  - "Nahasapeemapetilon" maps to "Apu"
  - "Eat more beef" maps to "Kick less cats“
- Here, an object called MyObject maps to integers:
  - myObj1 maps to 6
  - myObj2 maps to 7

Associative Arrays

- Awk associative array example:

```
myarray[0] = "dog"
myarray["cat"] = "feline"
myarray[3] = 6
```

- This is a sparse array, because there are breaks in the integer numbering from 0 to 3
- An associative arrays is also called:
  - Map, hash, lookup table

Perl

- Perl is a general-purpose programming language
  - Practical Extraction and Report Language
- Written by Larry Wall, released in 1987
- Borrows features from C, shell scripting, awk, sed, Lisp, and others
- Designed to be easy to use, not necessarily elegant

Perl

```
#!/usr/bin/perl
# The traditional first program.

# Strict and warnings are recommended.
use strict;
use warnings;

# Print a message.
print "Hello, World!\n";
```

Perl Camel

- http://perl.postbit.com/photos/other/perl-camel-source-code.html

- Image comes from here, one of the classic O'Reilly books:


```
#!/usr/bin/perl -w
# camel code
                                        $_='ev
                                      al("seek\040D
          ATA,0,                    0;");foreach(1..3)
       {<DATA>}my                 @camel1hump;my$camel;
  my$Camel  ;while(               <DATA>){$_=sprintf("%-6
9s",$_);my@dromedary            1=split(//);if(defined($
_=<DATA>)){@camel1hum          p=split(//);}while(@dromeda
 ry1){my$camel1hump=0         ;my$CAMEL=3;if(defined($_=shif
        t(@dromedary1     ))&&/\S/){$camel1hump+=1<<$CAMEL;}
       $CAMEL--;if(d    efined($_=shift(@dromedary1))&&/\S/){
      $camel1hump+=1  <<$CAMEL;}$CAMEL--;if(defined($_=shift(
    @camel1hump))&&/\S/){$camel1hump+=1<<$CAMEL;}$CAMEL--;if(
    defined($_=shift(@camel1hump))&&/\S/){$camel1hump+=1<<$CAME
    L;;}$camel.=(split(//,"\040..m`{/J\047\134}L^7FX"))[$camel1h
     ump];}$camel.="\n";}@camel1hump=split(/\n/,$camel);foreach(@
     camel1hump){chomp;$Camel=$_;y/LJF7\173\175`\047/\061\062\063\
     064\065\066\067\070/;y/12345678/JL7F\175\173\047`/;$_=reverse;
      print"$_\040$Camel\n";}foreach(@camel1hump){chomp;$Camel=$_;y
       /LJF7\173\175`\047/12345678/;y/12345678/JL7F\175\173\0 47`/;
        $_=reverse;print"\040$_$Camel\n";}';;s/\s*//g;;eval;   eval
          ("seek\040DATA,0,0;");undef$/;$_=<DATA>;s/\s*//g;(   );;s
            ;^.*_;;;map{eval"print\"$_\"";}/.{4}/g; __DATA__   \124
              \1   50\145\040\165\163\145\040\157\1 46\040\1  41\0
                   40\143\141  \155\145\1 54\040\1   51\155\  141
                   \147\145\0  40\151\156 \040\141    \163\16 3\
                    157\143\   151\141\16  4\151\1     57\156
                    \040\167  \151\164\1   50\040\      120\1
                    45\162\   154\040\15    1\163\      040\14
                    1\040\1   64\162\1       41\144       \145\
                    155\14    1\162\        153\04        0\157
                     \146\     040\11      7\047\         122\1
                     45\15      1\154\1  54\171           \040
                     \046\        012\101\16             3\16
                     3\15           7\143\15             1\14
                     1\16            4\145\163           \054
                    \040             \111\156\14         3\056
                   \040\           125\163\145\14        4\040\
                   167\1           51\164\1  50\0        40\160\
                  145\162                               \155\151
                 \163\163                                \151\1
                57\156\056
```

Perl Camel

- http://perl.postbit.com/photos/other/perl-camel-source-code.html

Python

- Similar philosophies as Perl, but now far more widespread than Perl
- In active development and usage
- Python is faster, with better support for Object Oriented Programming

Perl & Python

- Perl & Python are interpreted languages
- When you want to run code you've written, it is first read by an interpreter, slightly optimized ("compiled"), and then executed.
- Perl can only be interpreted by `perl` (the Perl interpreter), Python is interpreted by `python`

Python – Example Scripts

```
#!/usr/bin/python
print "Hello World!";
```

```
#!/usr/bin/python
# Create a file for writing
file = open("myfile.dat", "w+")
file.write("STUFF N JUNK”)
```

Python – Running an Example Script

```
$ cat pythontest
#!/usr/bin/python
print "Hello, World!";
$ chmod +x pythontest
$ pythontest
Hello, World!
```

Building a String in Python

```
$ cat pythonstring
#!/usr/bin/python
# comments!
mystring = "SO"
mystring += " MUCH "
mystring += "EASIER"
print "THIS IS " + mystring + " " + str(9) + " TIMES";
```

Converts the number into a string

```
$ pythonstring
THIS IS SO MUCH EASIER 9 TIMES
```

Python Math (Python 3)

```
$ cat pythonmath
#!/usr/bin/python
six = 6;
seven = 7;
thirteen = six + seven;
print("How much: {0}".format(str(thirteen)));
$ pythonmath
How much: 13
```

`printf()`-like functionality

Python versus C – A Vast Difference in Speed

C

```
$ cat c-billion.c
void main()
{
        long i, j;
        for (i = 0; i <= 1000000000; i++)
                j = i * i;
}
$ gcc -o c-billion c-billion.c
$ /usr/bin/time --format='%C took %e seconds' c-billion
c-billion took 2.78 seconds
```

- Both of these programs count to a billion

Python

```
$ cat python-billion
#!/usr/bin/python
for i in range(0, 1000000000):
        j = i * i;
$ /usr/bin/time --format='%C took %e seconds' python-billion
python-billion took 160.15 seconds
```

Mixing Languages

- Scripting languages and compiled languages can call each other

- This allows us to combine the best parts of one with the other, e.g.:
  - Speed == C
  - Short and easy to program == Python

Mixing C into Python

- This Python program calls a C program (it could have been a binary from any language) which *doesn't* return any results back to the Python script:

```
$ gcc -o c-billion c-billion.c
$ cat python-billion-fast
#!/usr/bin/python
from subprocess import call
call("./c-billion")
$ /usr/bin/time --format='%C took %e seconds' python-billion-fast
python-billion-fast took 2.91 seconds
```

Mixing C into Python

- Ways to get data back into Python:
  - Have the C program write a datafile, which is read by Python
  - Create a UNIX pipe, from which both Python and C can read and write
  - Create a C function inside the Python program with the “instant” module
  - Several other complex ways involving the Python C API, ctypes, SWIG, Boost Python API, etc., all of which use additional wrappers or APIs to manipulate and transmit data

- By the end of the course, you should be able to do the first two
- The others are non-trivial but are effective

Mixing Python into C

- You can write a C program that calls Python and returns the value back to C

- But why? Because many file and string handling tasks, especially extensive ones, are easier in Python

- Official example of this:

  https://docs.python.org/release/2.6.5/extending/embedding.html#pure-embedding

Mixing C into bash Shell Scripting

```
$ cat addsix-c.c
#include <stdio.h>
int main(int argc, char* argv[])
{
        printf("%d", atoi(argv[1]) + 6);
        return 0;
}
$ gcc -o addsix-c addsix-c.c
$ cat addsix-bash
#!/bin/bash
value=4
printf "value: %d\n" $value
i=$(./addsix-c 4)
#printf "i: %d\n" $i
printf "value + addsix: %d\n" $value
$ chmod +x addsix-bash
$ addsix-bash
value: 4
value + addsix: 10
```

This could be any pre-compiled binary, not just a C binary

It could even be another shell script

Mixing C into bash Shell Scripting

```
$ cat addsix-c.c
#include <stdio.h>
int main(int argc, char* argv[])
{
        printf("%d", atoi(argv[1]) + 6);
        return 0;
}
$ gcc -o addsix-c addsix-c.c
$ cat addsix-bash
#!/bin/bash
value=4
printf "value: %d\n" $value
i=$(./addsix-c 4)
#printf "i: %d\n" $i
printf "value + addsix: %d\n" $((value + i))
$ chmod +x addsix-bash
$ addsix-bash
value: 4
value + addsix: 10
```

Note: if this line is instead:

```
i=$("./addsix-c 4")
```

Then this line fails with an error because `"./addsix-c 4"` is not the name of a program; program names don't have spaces