Hands-on MacLabs
Session LK:
UNIX Scripting for Mac OS X

Wayne E. Beech
University of Kentucky

Macworld Conference & Expo

http://macpres09.shownets.net

Legal Stuff

University of Kentucky, UK, and the University of Kentucky logo are trademarks of the University of Kentucky.

Apple and Mac OS X are trademarks of Apple Computer, Inc. and are registered in the U.S. and other countries.

UNIX is a registered trademark of The Open Group

Course Description

Submitted Information

Description:

This session will introduce the concepts necessary to start writing scripts in a UNIX environment. We will touch on a variety of scripting languages that are often used in a UNIX / Mac OS X environment.

Who Should Attend:

System Administrators who are unfamiliar with scripting languages. General users who are interested in learning about scripting languages.

Course Description

Final Information

Description:

Looking to develop some UNIX scripting skills? Look no further. The mysteries of UNIX, such as Regular Expressions, shell control structures and loops, and more are revealed in this hands-on session. A variety of common and not-so-common commands are illustrated, and a brief survey of scripting tools and languages available on Mac OS X and Mac OS X Server are presented, all with examples. Since the best way to to learn is to jump right in and just start doing, you'll then have an opportunity to try your newly learned skills in the safety of the classroom.

Quick Survey

- Who uses the UNIX command line on a regular basis?
- Who understands the concept in UNIX that everything is a file?
- Who has a favorite shell and has changed their account so that Terminal uses that shell?
- Who understands I/O redirection?
- Who understands UNIX permissions?
- Who was written a shell scipt before?
- Who was written a Perl script before?

Basic

Concepts

I/O Redirection and Piping

Key Terminology/Concepts:

STDIN - Where input is coming from. The default is the keyboard.

STDOUT - Where output is going. The default is the users tty.

STDERR - Where error/warning/information messages are going. The default is the users tty.

I/O redirection - The process of changing (redirecting) STDIN, STDOUT and/or STDERR from the defaults mentioned above.

I/O Redirection and Piping

Simple Examples of Basic I/O redirection:

```
prompt% programname > outputfile

prompt% programname < inputdata

prompt% programname < inputdata > outputfile
```

I/O Redirection and Piping

Piping:

Piping is a special case of I/O redirection.

Placing a pipe symbol, |, on the UNIX command line connects the STDOUT of the command to the left of the | to the STDIN of the command on the right of the |.

You can string together as many commands and |s as you need to solve the problem you are working on.

You can still redirect the STDIN and/or STDOUT of the pipeline if desired.

I/O Redirection and Piping

Example of Building a Pipleline:

Assume we have these two files in our current directory:

input1

```
input1 data1
input1 data2
input1 data3
input1 data3
```

input2

```
input2 data3
input2 data2
input2 data1
input2 data3
```

I/O Redirection and Piping

If we what to issue the UNIX command to concatenate these two files and display the result:

```
prompt% cat input1 input2
```

```
input1 data1
input1 data2
input1 data3
input1 data3
input2 data3
input2 data2
input2 data1
input2 data3
```

I/O Redirection and Piping

Let’s assume we want to get rid of the input filename in the first field.

```
prompt% cat input1 input2 | cut -d ‘ ‘ -f 2
```

```
data1
data2
data3
data3
data3
data2
data1
data3
```

I/O Redirection and Piping

Now let’s assume we would like our data sorted

```
prompt% cat input1 input2 | cut -d ‘ ‘ -f 2 | sort
```

```
data1
data1
data2
data2
data3
data3
data3
data3
```

I/O Redirection and Piping

Now let’s assume we would like only one copy of each of the data items.

```
prompt% cat input1 input2 | cut -d ‘ ‘ -f 2 | sort | uniq
```

```
data1
data2
data3
```

I/O Redirection and Piping

Finally, we would like to have the output of this series of commands go into a file.

```
prompt% cat input1 input2 | cut -d ‘ ‘ -f 2 | sort | uniq > outfile
```

When this command is issued, you would simply get the command prompt back. However, a file named outfile would be created in your current directory that would contain the output of the sequence of commands issued.

I/O Redirection and Piping

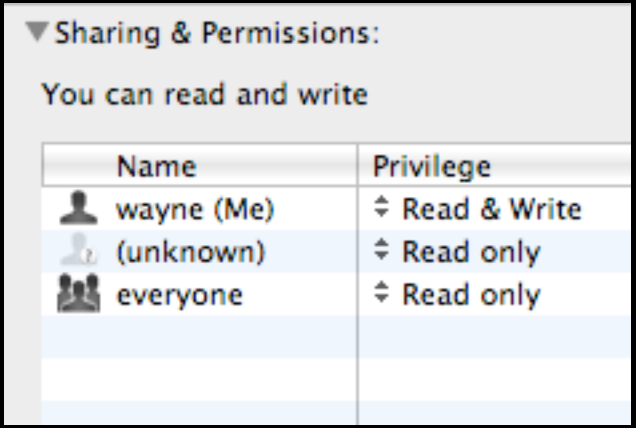
Example system administration task using piping

Mac OS X (and other versions of UNIX) have a killall command. You can do the same thing, and more complex things if needed with:

```
prompt% ps wwaux | grep wayne | grep -v grep |
awk ‘{print “kill “$2}’ | /bin/sh
```

UNIX Permissions

rwxr-xr-x



Sharing & Permissions:

You can read and write

Name	Privilege
wayne (Me)	Read & Write
(unknown)	Read only
everyone	Read only

UNIX Permissions

rwxrwxrwx

rwx	rwx	rwx
user	group	other

- A character (most commonly r, w or x) displayed means that the permission is granted to the appropriate person/group. A - in a character location means that the permission is not granted.

UNIX Permissions

Why are we talking about permissions?

Because for a UNIX script to behave as if it was a UNIX executible, its execute bit has to be set.

Example:

```
prompt% chmod +x scriptname.pl
```

Everyone start Terminal...

Applications → Utilities → Terminal

UNIX Permissions

Issue the command

```
ls -l
```

Scripting Basics

Scripting Basics

Why scripting languages?

- Originally UNIX scripting languages were written to allow you to program at your shell prompt or to place frequently used series of commands into a file to be run.
- I often do something like this:

```
stargate# while(1)
while? ./watchfolding.pl
while? sleep 3000
while? end
```

Scripting Basics

`#!/usr/bin/perl` What is this all about??

- The first UNIX shell was the Thompson shell (1969 - 1974), later replaced with the Bourne shell (1974 - present) (typically /bin/sh)

- Allowed you to put series of commands in a script file but also gave you traditional programmatic features

Scripting Basics

/bin/sh Example:

```
#!/bin/sh

echo Enter a value
read val
if [ $val -lt 0  -o  $val -gt 100 ]; then
    echo Value \"$val\" out of range
    exit 127
else
    echo Value \"$val\" is in range
fi
```

Scripting Basics

issue command: `ls -l sh1`

issue command: `chmod +x sh1`

issue command: `ls -l sh1`

issue command: `sh1`

issue command: `export PATH=$PATH:.`

issue command: `sh1`

Scripting Basics

What is we need information on all the various things we can do with sh?

issue command: man sh

Scripting Basics

What shells does Mac OS X support?

```
wayne% cat /etc/shells
/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
```

Scripting Basics

What shell is currently the default for Mac OS X?

/bin/sh for root, /bin/bash for users, but...

```
[sabine:~] wayne% sh -version
GNU bash, version 3.2.17(1)-release (i386-apple-darwin9.0)
Copyright (C) 2005 Free Software Foundation, Inc.
[sabine:~] wayne% bash -version
GNU bash, version 3.2.17(1)-release (i386-apple-darwin9.0)
Copyright (C) 2005 Free Software Foundation, Inc.
```

Scripting Languages

Scripting Languages

So far we have just talked about shell scripting. What other scripting languages do we have with Mac OS X?

http://developer.apple.com/internet/opensource/opensourcescripting.html

Perl, Python, php, ruby, AppleScript

Apple Script

AppleScript

- Released in 1989
- Its purpose is to allow users to automate task.
- Uses “Apple Events” to control applications
- Applications -> AppleScript -> Script Editor
- Folder Actions
- http://developer.apple.com/referencelibrary/GettingStarted/GS_AppleScript/index.html

AppleScript

Simple Example:

```
#!/usr/bin/osascript
say "I am going to launch TextEdit"
delay 2
tell application "TextEdit"
  launch
  set newwin to make new document
  activate
  tell newwin
    set its text to (date string of (current date) as text)
    set font of paragraph 1 to "CourierNew-Bold"
    set size of paragraph 1 to 35
  end
end
```

AppleScript

More Involved Example:

```
set fn to "Powerbook:Users:wayne:cis245students"
set urlpath to the text returned of ?
   (display dialog "Enter URL" default answer
    "http://sweb.uky.edu/~STU/cit245/pgmx.html")
set fileref to (open for access file fn)
set fileContents to (read fileref as text using
delimiter (ASCII character 10))
repeat with theItem in fileContents

   set userid to theItem as text

   set currenturl to searchReplace(urlpath, "STU",
userid)
```

AppleScript

```
tell application "Safari"
    if not (exists (document 1)) then
        --
        -- Safari is *Not* Opened, Call Open New Window Function, Go To Preferred Page
        -- and Activate Safari to Enable Scripting
        --
        tell application "Safari" to activate
        my open_new_window()
        --
        -- Set Safari Location to Preferred Page
        --
        set the URL of document 1 to currenturl
    else
```

AppleScript

```
        --
        -- Safari is Opened, Bring it To Front and
Open New Tab w/Preferred Page
        --
        if (exists (URL of document 1)) then
          my open_new_tab()
        end if
        --
        -- and Activate Safari to Enable Scripting
        --
        tell application "Safari" to activate
        --
        -- Set Safari Location to Preferred Page
        --
        set the URL of document 1 to currenturl
      end if
    end tell
```

AppleScript

```
    -- display dialog "next?" giving up after 1800
    delay 1

end repeat
```

AppleScript

```
on searchReplace(theText, SearchString,
ReplaceString)
   set OldDelims to AppleScript's text item
delimiters
   set AppleScript's text item delimiters to
SearchString
   set newText to text items of theText
   set AppleScript's text item delimiters to
ReplaceString
   set newText to newText as text
   set AppleScript's text item delimiters to
OldDelims
   return newText
end searchReplace
```

AppleScript

```
--
-- Open New Window Function
--
on open_new_window()
   tell application "Safari" to activate
   tell application "System Events"
     tell process "Safari"
       click menu item "New Window" of menu
"File" of menu bar 1
     end tell
   end tell
end open_new_window
```

AppleScript

```
--
-- Open New Tab Function
--
on open_new_tab()
   tell application "Safari" to activate
   tell application "System Events"
      tell process "Safari"
         click menu item "New Tab" of menu "File" of menu bar 1
      end tell
   end tell
end open_new_tab
```

php

php

- Released in 1995
- Primary used as a server-side HTML-embedded scripting language
- http://www.php.net

php

Simple Example:

```
#!/usr/bin/php
text is just displayed...
<?php phpinfo(); ?>
```

ruby

ruby

- Released in 1993

- The author,Yukihiro Matsumoto,wanted to create a fresh programming language that could provide a balance between functional programming and imperative programming.

- http://www.ruby-lang.org

ruby

Simple Example:

```
#!/usr/bin/ruby

1.upto(100) do |percent|
  sleep 1
  puts "<xgrid>"
  puts "{control = statusUpdate; percentDone = %d.
0; }" % percent
  puts "</xgrid>"
  $stdout.flush
end
```

ruby

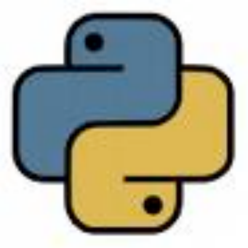
Another Simple Example:

```
#!/usr/bin/ruby

def fact(n)
    if n == 0
        1
    else
        n * fact(n-1)
    end
end

puts fact(ARGV[0].to_i)
```

python



python

- Released in 1991
- python is a general-purpose, high-level programming language which emphasizes programmer productivity and code readabiltiy
- http://www.python.org

python

Simple Example
(from http://www.amk.ca/python/simple/):

```
#!/usr/bin/python
import string, sys
if len(sys.argv)==1:
    print 'Usage: celsius temp1 temp2 ...'
    sys.exit(0)
# Loop over the arguments
for i in sys.argv[1:]:
    try:
        fahrenheit=float(string.atoi(i))
    except string.atoi_error:
  print repr(i), "not a numeric value"
    else:
  celsius=(fahrenheit-32)*5.0/9.0
  print '%i F = %i C' % (int(fahrenheit),
int(celsius+.5))
```

Perl

Perl

- Released in 1987
- Was created by a UNIX systems administratory, Larry Wall, as a programming language to make reporting and systems administration task easier
- Took what he liked from C, sed, awk and the Bourne shell
- Became popular for writing server-side CGI scripts
- http://www.perl.com
- http://www.perl.org
- http://www.cpan.org

Perl

Simple Example (Xgrid example from before):

```
#!/usr/bin/perl

$| = 1;

for($i=0;$i<=100;$i++)
    {
    sleep 1;
    printf "<xgrid>{control = statusUpdate;
percentDone = %d.0; }</xgrid>"
            ,$i;
    }
```

Scripting Gotchas

Scripting Gotchas

- Missing #! line

Cause: Either you didnt put one in your script, you have a space before it, or you have a blank line before it

Result: This may cause your script to be processed by your shell instead of the intended scripting language

The reason this may be confusing is that often times there are enough similarities that the incorrect interpreter may be able to run some of the program

Scripting Gotchas

- Execute bit not set

Cause: Either you didnt change the program’s permissions from the default or you set them incorrectly

Result: This may cause your script to give the error message “Permission denied”

The reason this may be confusing is that often times people don’t understand why a script they just created would not allow them to execute it

Scripting Gotchas

- Incorrect #! line

Cause: You have a typo or mistake in the #! line. For example, putting #!/usr/local/bin/perl when it should be #!/usr/bin/perl

Result: This may cause your script to give the error message “Command not found”

The reason this may be confusing is that often times you will look to make sure you are typing the command correctly and/or giving the correct path to the command, and you are, it is telling you that the command in the #! line can not be found.

Regular Expressions

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Regular Expressions

Definition: A regular expression is a pattern that describes a set of strings without having to list every string in the set.

Example: `[0-9]{3}-[0-9]{4}`

Any ideas??

Describes a 7 digit phone number

Regular Expressions

Regular Expressions can be used for matching and searching in conjunction with a number of UNIX commands:

grep:

```
grep ‘[0-9]\{3\}-[0-9]\{4\}’ fn
```

sed:

```
cat fn | sed ‘s/[0-9]\{3\}-[0-9]\{4\}/###-####/’
```

awk:

```
/[0-9]{3}/ {print “3”}
/[0-9]{8}/ {print “8”}
```

Regular Expressions

Regular expressions can be just a series of characters.

Example: abc

Regular expressions can have characters that represent positions with in the string:

^ represents beginning of string
$ represents end of string

Example: ^abc

Regular Expressions

A . in a regular expression represents a single character.

Example: ^a.c$

Regular Expressions

Characters in a regular expression can be followed by a quantifier. The quantifier tells how many occurences you of a character you are looking for.

The quantifier qualifies the character before it.

Some quantifiers and their meanings:

*	=> 0 or more occurrences
+	=> 1 or more occurrences
{n}	=> n occurrences
{o,p}	=> o to p occurrences
{q,}	=> q or more occurrences

Example: a.{6}b

Regular Expressions

A character class is a set of characters in square brackets. A character class designates that any of the characters in the class are acceptable at that position within the string.

Example: d[abc]ef

Character classes can be quantified as well:

Example: a[xyz]+b

Contact Information

Wayne E. Beech
Systems Labs
210 McVey Hall
University of Kentucky
Lexington, KY 40506-0045
wayne@uky.edu

Questions and Answers

http://macpres09.shownets.net