Shell Scripting

AFNOG X
Cairo, Egypt
May 2009

Why

- Scheduled Tasks
- Repetitive sequences
- Boot scripts

When not to use scripting

- Resource-intensive tasks, especially where speed is a factor
- Complex applications, where structured programming is a necessity
- Need direct access to system hardware
- Proprietary, closed-source applications

Sample repetitive tasks

- Cleanup
- Run as root, of course.

```
# cd /var/log

# cat /dev/null > messages

# cat /dev/null > wtmp

# echo "Logs cleaned up.”
```

- You can put these commands in a file and say bash filename

she-bang

- #! and the shell (first line only)
- chmod a+x (remember the permissions)
- Example: put the following text in hello.sh

```
#!/bin/bash
echo Hello World
```

- chmod a+x hello.sh
- ./hello.sh (remember $PATH)

variables

- Variable is a “container” of data. Some variables already exist in your “environment” like $PATH and $PROMPT
- Shell substitutes any token that starts with $ with the contents of the variable of that name
- Variable can be created using VAR=something – some shells require the keyword “set” to make it persist, others need “export”

Sample special variables

$ echo $PATH

the shell searches PATH for programs if you do not type them with an absolute path

$ echo pwd
$ echo $(pwd)

the shell runs the command in between "$(" and ")" and puts the result on the command line

$?

When a process ends, it can leave an "exit code" which is an integer which you can check. If the exit code is zero then usually it exited successfully. Non zero usually indicates an error.

sample repetitive tasks revisited

```
#!/usr/local/bin/bash # Proper header for a Bash script.


# Cleanup, version 2


# Run as root, of course.
# Insert code here to print error message and exit if not root.


LOG_DIR=/var/log# Variables are better than hard-coded values.


cd $LOG_DIR
cat /dev/null > messages
cat /dev/null > wtmp


echo "Logs cleaned up.”
```

Conditionals

- ***if expression then statement***
- ***if expression then statement1 else statement2.***
- ***if expression1 then statement1 else if expression2 then statement2 else statement3***

Bash conditional syntax

```
#!/usr/local/bin/bash
if [ "foo" = "foo" ]; then
        echo expression evaluated as true
fi
```

```
#!/usr/local/bin/bash
if [ "foo" = "foo" ]; then
    echo expression evaluated as true
else
    echo expression evaluated as false
fi
```

Loops

- **for** loop lets you iterate over a series of 'words' within a string.
- **while** executes a piece of code if the control expression is true, and only stops when it is false
- **until** loop is almost equal to the while loop, except that the code is executed while the control expression evaluates to false.

Sample syntax

```
#!/usr/local/bin/bash
for i in $( ls ); do
    echo item: $i
done
```

```
#!/usr/local/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
             echo The counter is $COUNTER
let COUNTER=COUNTER+1
done
```

```
#!/usr/local/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
    echo COUNTER $COUNTER
    let COUNTER-=1
done
```

Practice

- Write a shell script to print the disk usage every 5 seconds.
- Hint: sleep N is a command which will basically put the prompt/program to sleep for N seconds
- Hint2: in any conditional, you can say “true” or “false” to force it to always evaluate like that.

Extra

- Programming (say in C) builds on similar concepts.
- Source text is COMPILED into binary machine code. Why?

hello world (c style)

- Edit hello.c and put the following text

```
#include <stdio.h>
int main(){
  printf("Hello World\n");
  return 0;
}
```

- Type gcc -o hello hello.c
- Type ./hello ; echo $?
- Change the return 0 to return 42
- Compile it again,
- Run ./hello; echo $?