Shell Scripting

Intermediate Systems Administration DeCal
Lecture #6
George Wu
Slides prepared by Joshua Kwan

Today

- Unlock the raw power of the shell as a programming language!
- How to use variables + special variables
- How to create functions
- Control structures: *for, while, if, case*
- Input functionality with *read, shift*
- Shell builtins: *echo, printf*, *let, test* (aka *[ ]*)

Shell scripts

- At their simplest, just a list of commands executed in order as if you had typed them into the shell.
- Anything you can do at the shell, you can do in a shell script, and vice versa.
- Like DOS batch scripts, but *way better*.
- Run them by putting “#!/bin/sh” at the top and using chmod +x to make executable.

Variables

- Assignment: *FOO="Test 1 2 3"*
- Reference: *echo $FOO* or *echo "$FOO"*
  (What's the difference?)
- Want to set a variable to the output of a command? Input substitution!
  FOO=$(ls pictures)

Special Shell Variables

- *$1, $2, $3*.. - arguments passed in on command line.
- *$@* - all arguments as a big string.
- *$#* - number of arguments passed in
- *$?* - exit code of last program; you knew this already
- *$$* - your process ID
- *$!* - process ID of last program started w/ ‘&’

Functions

- When you make a shell script, lines of code are executed top-to-bottom
- If you make functions, they won't be run though, just *declared*. You can use them as if they were separate programs.
- Learn by example! We know enough to write a simple program now.

Example 1

```
#!/bin/sh

confuciusprint() {
     echo "Confucius say: \"$@\""
}

confuciusprint "Baseball wrong. Man with four balls cannot walk."

echo "OK, now it's your turn! Here's your quote:"

confuciusprint "$@"

echo "What if it were only the first word you said?"

confuciusprint "$1"
```

Control structures

- For loops set a variable based on the contets of a list (like python, unlike C):

  for *x* **in** *$(seq 1 9);* **do** *touch $x;*
  done

- While loops test a condition and exit when the condition is 1. You can also run a program...

  while *! try_to_connect;* **do** *echo "Trying to connect...";* **done**

- and if statements behave the same way (use a conditional or a program), but they don't loop

  if *[$SUM -eq 0];* **then** *echo Zero;*

Control structures

- You can have many conditional branches with if: *if ...; then ...;* **elif** *...; then ...;* **else** *...; fi*
- case statements; like switch in C, for many nested ifs:

```
case "$x" in
[aA]) echo "a for anteater!" ;;
b|c) echo "you typed in b or c" ;;
*) echo "who knows what you typed";;
esac
```

Conditionals

- In a previous example we did this:

```
if [ $SUM -eq 0 ]; then
    echo Zero
fi
```

- This is a conditional, however it’s implemented using a program called [that evaluates the condition and returns 0 or 1.
- test is the same thing, but it doesn’t require a closing bracket. Personal taste.

```
if test $SUM -eq 0; then ...
```

Conditionals

- `[ -n "$var" ]`: returns true if $var is non-blank (opposite: `-z`)
- `[ "$var" -eq 1 ]`: returns true if $var is a number and is 1. (opposite: `-ne`)
- Ditto for `-ge` (greater/equal), `-gt` (greater than), `-le` (less/equal), `-lt` (less than)
- `[ "$var" = foo ]`: returns true if $var equals “foo” by string comparison. (opposite: !=)
- `[ -f "file.txt" ]`: returns true if file.txt exists and is a file. No opposite; negate it, e.g.

Input processing

- Want to use standard input? *read var* will read one line of standard input into *$var*. A typical construct:

```
while read line; do
    do stuff with $line
done
```

Input processing

- You can also parse your command line arguments one by one.

```
while [ $# -gt 0 ]; do
    echo "$1"
    shift
done
```

- shift will delete $1, and shift everything else down. ($2 becomes $1, $3 becomes $2). Then it decrements the value of $#.

Useful builtins

- The shell has several built-in programs for very common tasks.
- *echo*: prints a line to the screen, you knew this already.
- *printf*: does printf(3) style formatting on text, e.g. `printf '%02d' "$tracknumber"`
- *let*: changes variables, e.g. `let "x=x+1"` changes $x

Image resizing example

```
#!/bin/sh

FILE="$1"

if [ ! -f "$FILE" ]; then
  exit 1
fi

ID=$(identify "$FILE" | cut -d' ' -f3)
WIDTH=$(echo "$ID" | cut -dx -f1)
HEIGHT=$(echo "$ID" | cut -dx -f2)
let RATIO=”(WIDTH*100)/HEIGHT”

if [ "$RATIO" -eq 133 ]; then # landscape
  mogrify -scale $2x$3 "$FILE"
elif [ "$RATIO" -eq 75 ]; then # portrait
  mogrify -scale $3x$2 "$FILE"
fi
```