CSE 374 Lecture 4

Shell Variables and More Scripting

Today

1. Shell Variables
   a. Defining your own
   b. Built-ins
   c. Exporting
2. Arithmetic
3. Logic and loops
4. Things to watch out for

Variables

Shell has a state, which includes shell variables

All variables are strings (but can do math, later)

Create: myVar= or myVar=value

Set: myVar=value

Use: $myVar

Remove: unset $myVar

List variables (use 'set)

Special Variables

Common variables which set shell state:

$HOME - sets home directory. $HOME=~/CSE374 would reset your home directory to always be CSE374

$PS1 - sets prompt

$PATH - tells shell where to look for things. Often extended:
$PATH=$PATH:~/CSE374

Show current state: `printenv`

Variables useful in a script

$# stores number of parameters (strings) entered

$0 first string entered - the command name

$N returns the Nth argument

$? Returns state of last exit

$* returns all the arguments

$@ returns a space separated string with each argument

    (* returns one word with spaces, @ returns a list of words)

Quoting Variables

In order to retain the literal value of something use 'single quotes'

In order to retain all but $, `, \ use "double quotes"

Put $* and $@ in quotes to correctly interpret strings with spaces in them.

Export Variables

Use: export myVar

To make variable available in the
initial shell environment.

If a program changes the value of
an exported variable it does not
change the value outside of the
program

: export -n remove export property

Variables act as though passed by
value

Arithmetic

Variables hold strings, so we need a way to tell the shell to evaluate them numerically:

K=$i+$j does not add the numbers

Use the shell function ((

k=$(($i+$j))

Or let k="$i+$j"

The shell will automatically convert the strings to the numbers

Functions and local variables

Yes, possible

Generally, a script's variables are global

```
name () compound-command [ redirections ]
or
function name [()] compound-command [ redirections ]

Ex:

func1()
{
    local var='func1 local'
    func2
}
```

Stuff to watch out for

White space: spacing of words and symbols matters

Assign WITHOUT spaces around the equal, brackets are WITH SPACES

Typo on left creates new variable, typo on right returns empty string.

Reusing variable name replaces the old value

Must put quotes around values with spaces in them

Non number converted to number produces '0'

Conditionals

Binary operators: -eq -ne -lt -le -gt -ge

Can use the [[shell command to use < , > , ==

Syntax is a little different, but commands works as expected

Shell-scripting Notes

Bash Scripting

Java Programming

Interpreted

Compiled

Esoteric variable access

Highly structured, Strongly typed

Everything is a string

Strings have library processing

Easy access to files and program

Data structures and libraries

Good for quick & interactive programs

Good for large complex programs