Lecture 6

Introduction to Shell Scripts

What they are, creating them, running them, constructs, variables.

COP 3353 Introduction to UNIX

*

What is a shell script?

- A text file made executable! Contains:
- Unix shell commands
- Programming control constructs (if, then, while, until, case, for, break, continue, while, …)
- basic programming capabilities (assignments, variables, arguments, expressions, …)
- The file entries are the *script*
- The file is **interpreted** rather than compiled and executed
- The first line of the script indicates which shell is used to interpret the script

Simple script (myscript.sh)

```
#!/bin/sh
#this is the script in file myscript.sh
cal
date
ls > file.txt
exit
```

- The "#!" is used to indicate that what follows is the shell used to interpret the script
- The "exit" command immediately quits the shell script (by default it will also quit at the end of the file, even if there is no "exit" command present. Not required, but handy in certain places.)
- Anything following a lone # character is a comment.

Executing shell scripts - 2 ways!

- sh myscript #uses Bourne shell
- bash myscript #uses Bourne again shell

Note that the above explicitly invoke the appropriate shell with the file containing the commands as a parameter. The file does not need to be executable when script is run in this manner..

- Method #2: You can also make the file executable and then simply run as you would a command

> chmod u+x myscript.sh

> myscript.sh

Shell scripts

Advantages

- Can quickly setup a sequence of commands to avoid a repetitive task
- Can make several programs work together

Disadvantages

- Little support for large and complicated programming semantics
- Shell scripts need to be interpreted hence are slower programs

Which shell to use?

- csh shell and tcsh shell are recommended for use at the command line
- sh (Bourne) shell and bash shell are recommended for writing shell scripts
- Examples will generally use the Bourne shell

Printing a line to standard output

- Use the echo command to print a line to stdout
- Form of command:

echo <zero or more values>

- Examples

```
echo “Hello World”
echo “hello” “world” #two values
echo hello  #need not always use quotes
echo “please enter your name”
- Quotes helpful when typing special
characters like ‘ within string to print
```

Shell arguments

- Arguments on the command line can be passed to a shell script, just as you can pass command line arguments to a program
- $1, $2, …, $9 are used to refer to up to nine command line arguments (similar to C's argv[1], argv[2], …, argv[9]).
- Note that $0 contains the name of the script (argv [0])
- Example:

```
shprog.sh john 40
shprog.sh bob 45 "new york"
```

Example using shell arguments

- Script:

```
#!/bin/sh
#script name is greet.sh
#friendly display of today's date
echo "Hello" $1 $2 "pleased to meet you"
echo "The date is"
date
exit
```

- Usage:

```
greet.sh john smith
```

Shell Environment Variables

- These are variables provided as part of the shell's operational environment. Type command env to see all of the Environment variables.
- They exist at startup but can be changed
- Examples are: USER, HOME, PATH, SHELL, HOSTNAME
- The "setenv" command (in tcsh) is used to set these, for example, by:

```
setenv PATH $PATH:/home/here/bin
```

(this sets the PATH variable so that it's current value is appended by :/home/here/bin)

- Note that setenv is how tcsh sets the environment variables

Shell variables (Bourne shell) - User Created

- Note that for all shells, variables need not be declared explicitly, but simply used
- For the Bourne shell, the use is as follows (note that there should be no blanks before and after the equals sign and no need for the set command.
- Form:

<name>=<value>

- Example

```
alpha="hello world"
beta=45
echo $alpha $beta "third argument"
```

- Note that $alpha is the value of the variable alpha

Shell Variables in TCSH - User Created

- In tcsh, the "set" command is used to set a variable to a string value
- Form: set <name> = <value>
- Examples:

```
set alpha = "any string"
set beta = 3
set mypath = /home/special/public_html
```

- Once a variable has been defined, it's value can be used by "dereferencing" it with $. `ls -al $mypath`
- Note that using setenv or set without any parameters simply displays the current settings

Reading values into shell variables

- The read statement is used to read a line of standard input, split the line into fields of one or more strings, and assign those strings to shell variables. Any strings not assigned are assigned to the last variable.
- Form:

read <var1> <var2> … <varn>

- Examples

```
read num
read field1 field2 rest
read field1 field2 < ifile.txt
```