Shell Scripting

Class Meetings 6-7

* Notes adapted by Doug Bowman from previous work by other members of the CS faculty at Virginia Tech

Shell Script (Program)

- What is a shell script?
  - Shell commands in a text file that is invoked as its own command
- Commands include . . .
  - anything you can type on the command line
  - shell variables
  - control statements (`if`, `while`, `for`, . . .)

Script Execution

- Provide script as an argument to a shell command — `bash my_script`
- Or specify shell on the first line of the script
  - First line of script is `#!/bin/bash`
  - Make sure that the script is executable
  - Run `my_script` directly from the command line
- No compilation; interpreted by shell

Simple Script

```
#!/bin/bash
echo "Hello, World!"
system=$(uname -a)
echo $system
```

Result:

```
Hello, World!
Linux cashew.cslab 2.6.3-14mdk #1
  Fri Aug 13 11:14:25 EDT 2004 i686
  unknown unknown GNU/Linux
```


Shell Variables

- Numeric
- Strings
- Arrays
- `var` refers to the name of the variable, `$var` refers to the value
  - `var=100 # sets the value to 100`
  - `(( var2=1+$var ))`
- Variable names begin with a letter and can include letters, digits, and underscores

Numeric Variables

- Integer variables are the only pure numeric variables that can be used in `bash`
- Declaration and setting value
  - `declare -i var=100`
- Numeric expressions are enclosed in double parentheses
  - `(( var+=1 ))`
- Operators are the same as in C/C++
  - `+, -, *, /, %, &, |, <, >, <=, >=, ==, !=, &&, ||`

String Variables

- Unless explicitly declared as another type, variables are strings
- `var=100` makes the `var` the string `"100"`
- However, placing the variable within double parentheses will treat it as an integer
  - `(( var2=1+$var ))`

String Variables (continued)

- Using substrings
  - `${string:n}`
    - `${string:5} # first five chars`
    - `${string:-2} # last two chars`
  - `${string:n:m}`
    - `${string:0:4} # first to fifth`
    - `${string:1:3} # second to fourth`
  - `${#string} # length of string`
- Concatenating strings
  - `string_var1="$string_var1 $string_var2"`

Array Variables

- Array is a list of values — do not have to declare size
- Reference a value by `${a[index]}`
  - `${a[3]} # value in fourth position`
  - `$a # same as ${a[0]}`
- Use the `declare -a` command to declare an array
  - `declare -a sports`
  - `sports=(basketball football soccer)`
  - `sports[3]=hockey`

Array Variables (cont)

- Array initialization
  - `sports=(football basketball)`
  - `moresports=($sports tennis)`
- `${array[@]}` or `${array[*]}` refers to the entire contents of the array
  - `echo ${moresports[*]}`
  - Output: `football basketball tennis`

Command Line Arguments

- If arguments are passed to a script, they can be referenced by `$1`, `$2`, `$3`, etc.
- `$0` is the name of the script
- `$*` is a string of all of the arguments separated by spaces
- `$@` is an array of all of the arguments
- `$#` is the number of arguments

Output and Quoting

- `echo a message # print to stdout`
- `echo -n "yes/no? " # a prompt`
- Shell interprets `$` and `` ` ` `` within double quotes
  - `$` — variable substitution
  - `` ` `` — command substitution
  - `echo "`date +%D`" # 09/27/04`
- Shell does not interpret special characters within single quotes
  - `echo '`date +%D`' # `date +%D``

Redirecting Output to File

- Redirecting output to a file is the same as on the command line
- Examples
  - `echo "$var" > $OUTFILE  # overwrite`
  - `echo "$var" >> $OUTFILE # append`
- String expansion
  - `echo $'\n\n\n'  # 3 blank lines`
  - `echo $'\t' indent  # indented text`

Control

- Testing conditions
- `if` — Conditional execution
- `for` — Loop over the elements of an array
- `while` — Loop while a condition is true
- `case` — Multiway conditional on variable value
- Define and invoke functions

Conditions

- If using integers: `(( condition ))`
- If using strings: `[[ condition ]]`
- Examples
  - `(( a == 10 ))`
  - `(( b >= 3 ))`
  - `[[ $1 = -n ]]`
  - `[[ ($v != $un) && ($v != games) ]]`

Conditions (cont)

- `[[ -e $file ]]` – File exists?
- `[[ -f $file ]]` – Regular file?
- `[[ -d $file ]]` – Directory?
- `[[ -L $file ]]` – Symbolic link?
- `[[ -r $file ]]` – Read permission?
- `[[ -w $file ]]` – Write permission?
- `[[ -x $file ]]` – Execute permission?

If Statement

- Syntax

```
if condition1
then
  statements
elif condition2
then
  statements
else
  statements
fi
```

optional

If Statement — Example

```
file=readings.php
if [[ -r $file ]]
then
  echo "$file is readable"
elif [[ (-w $file) && (-x $file) ]]
then
  echo "$file is writable and executable"
fi
```

Result:

```
readings.php is readable
```