CS 246 Fall 2016 - Tutorial 2

September 21, 2016

1 Summary

- Bash Variables
- Program Exit Codes
- Bash Loops and If Statements
- Bash Scripting
- Testing Exercise

2 Tip of the Week: More Vi Commands

- `dd`: Delete the current line.
- `C`: Delete contents from current character to the end of the line and enters insert mode.
- `D`: Deletes contents from current character to the end of the line.
- `cw`: Deletes all characters from the current character to the next whitespace and enters insert mode.
- `dw`: Deletes all characters from the current character to the next whitespace.
- `r`: Replace the current character with the next character typed.
- `R`: Enter repalce mode. This is like insert mode except characters will be replaced when you type.
- `s`: Delete the current character and enter insert mode.
- `S`: Deletes contents of the current line and enters insert mode.

3 Bash Variables

- In bash, a variable is declared as follows: `var=42`. Note. There cannot be spaces on either side of the equals symbol.
- All variables are stored as strings.
- Unlike C variables, bash variables persist outside of the scope of if statements, loops, and scripts.
- Accessing the value in a variable: `$var`
- `${var%<end>}` removes `<end>` from the from the string stored in var. If `<end>` is not at the end of var, the string is unchanged.
- We can store a command in a variable and call it later.

4 Embedded Commands

- We can use a subshell to embed commands as command line arguments to scripts.
- `egrep $(cat file) myfile.txt` could allow us to run egrep with the contents of a file being the regular expression.
- Note the difference between `x="echo cat"` and `x=$(echo cat)`. In the first expression, when `x` is evaluated, the output is the word "cat". In the second expression, echo returns cat which is then run by the shell

Question: What does the following sequence of commands output?

```
x=echo
$x $($x $x)
```

1. `echo echo echo`
2. `echo echo`
3. `echo`
4. nothing

5 Program Exit Codes

- When a program completes, it always returns a status code to signify if the program was a success.
- This is true of any C program you have written before now. The exit code is the value returned from main, hence the contract `int main();`. In C and C++, if you do not return from main, the exit code is 0.
- In bash, if a program is successful, the exit code is 0. Otherwise, the exit code is non-zero. The exit code is stored in the variable `?`.

6 Bash Loops and If Statements

- For the condition in both if statements and while loops checks the result, go into the body of the if statement or while loop if it's true.
- Form of an if statement in bash:

```
if [ cond1 ]; then
   ...
elif [ cond2 ]; then
   ...

...
else

fi
```

- Form of a while loop in bash:

```
while [ cond ]; do
   ...
done
```

- Form of a for loop in bash:

```
for var in words; do
   ...
done
```

- "words" is a list of whitespace separated strings. The loop runs once for each string in "words".
- Side note: '[cond]' can be replaced by any command and the exit code will be checked.

6.1 Test

- Test is a bash command. The program is '[' and is called in the form `[ cond ]` whose exit code is 0 if `cond` is true and 1 if `cond` is false. It may be useful to review the man page for test.
- A short non-exhaustive list of conditions for test:

```
num1 -gt num2: num1 > num2
num1 -lt num2: num1 < num2
num1 -eq num2: num1 == num2
num1 -neq num2: num1 != num2
str1 = str2: str1 == str2
-e file: file exists
```

7 Bash Scripting

- A bash script is a series of commands saved in a file so that we can accomplish the same task without having to manually type all the commands
- The first line of every shell script is the 'shebang line' - #!/bin/bash. This line is telling the shell what langauge the file is using.
- To call a bash script, give the file excutable permission using chmod and call the file by giving either an absolute path or a relative path.
- Command line arguments are $1, $2, ect. The number of commands line arguments is stored in $#.

7.1 Subroutines in Bash Scripts

- Format:

```
subroutine(){
...
}
```

- A subroutine is a series of commands which can be called at any time in a bash script.
- They can be given command line arguments the same way a program would be given command line arguments. A subroutine cannot access the command line arguments to the script. All other variables can be accessed.
- Write a bash script which takes in two arguments, ext1 and ext2. For each file (not directory) in current directory which ends with an .ext1, rename the file to end with .ext2.

8 Testing Exercise

We have a compiled program which takes in numbers and tests if they statisfy a property. For each test, the program will return 'true' for each line of numbers which satifies the property and 'false' for all other input.

First input: 2 4 6 results in true

Additional Exercise: Consider how you would test the bash script we wrote in this tutorial.

9 Tip of the Week: Bash Script Debugging

- A debugging mode can be activated when running a bash script by placing '-x' at the end of the shebang line.
- When running the script, each command, with variables being expanded, is printed to the screen.
- If a script is not doing what you expect it to do, using this debugging mode can be an easy way to see what is happening in the script.
- Note: you must call the script by making it executable and providing the path to the script to use this debugging mode.