Lec 03 - Bash - Working With Scripts

CS 1XA3

Jan. 23, 2018

Your First Bash Script!

There comes a point in every young coder's life when typing in the same commands for the same operation over and over again just isn't satisfying anymore

- Create a new file with the extension .sh

  ```
  touch test_script.sh
  ```

- Put the following at the top of the file, to signify what shell to use **(bash)**, and add an echo for some content

  ```
  #!/bin/bash

  echo "Hello World"
  ```

Your First Bash Script!

There are generally two ways to go about executing a bash script

- Use the sh command

  `sh test_script.sh`

- Or make the file an executable with chmod

  `chmod a+x test_script.sh`

- Then execute with

  `./test_script.sh`

Roles and Permissions - chown

Roles

a: all

u: user

g: group

o: others

Permissions

r: read

w: write

x: execute

chmod can add / remove permissions to different roles on a file like in a variety of ways using the following syntax

```
chmod a-wx file.txt
# remove write / execute permissions from everyone
chmod u+x file.txt
# give executable permissions to the user
```

Bash Variables

- Assign a variable

```
myvar=Hello
```

- Extract a variables value using $

```
echo $myvar
```

- Beware: of variable expansion with white-spaces, to be safe, wrap your variable expansions in quotes

```
echo "$myvar"
```

Command Line Arguments

- Command line arguments are accessed through **$1**, **$2**, etc
- One can remake the cp command like so

```
#!/bin/bash

cp "$1" "$2"

echo "Copied $1 to $2"
```

- Another built-in variable **($#)**, gives the number of command line arguments supplied

  Note: quotations were used in this example **just to be safe**

Parameter Expansion (Curly Braces)

- Use curly braces to avoid ambiguities

```
${var} # same thing as
$var
```

- Why do this? Consider

```
var = Hello

echo "$varGoodBye"    # unexpected
echo "${var}GoodBye" # that's better
```

Note: there's more functionality to curly brace expansion, but we'll ignore it in this course

Conditionals (If Statements)

Syntax

```
if [ cond ]
then
    commands
elif [ cond ]
then
    commands
else
    commands
fi
```

Note: **elif** and **else** are optional. See next slide for what qualifies as **cond**

Conditionals (Test)

Anything inside of square brackets is actually a reference to the command test. The following operators are proper flags to test

```
!Expression          # Not Expression
-n STRING            # Length STRING > 0
STRING1 = STRING2    # STRING Equality
STRING1 != STRING2   # STRING Inequality
INT -eq INT          # Integer Equality
INT -gt INT          # Integer >
INT -lt INT          # Integer <
    -d FILE          # File is directory
    -e FILE          # File exists
```

For Loops (Over Integers)

Two choices of syntax for iterating over Integers

- Curly Brace Syntax

```
for i in {1..5}
do
    echo $i
done
```

- Range Syntax

```
for ((i=1;i<=5;i+=1))
do
    echo $1
done
```

For Loop (Over Files)

```
shopt -s nullglob
for f in ~/*.tmp
do
        echo "Iterating over tmp file - $f"
done
```

Note: the **shopt** handles the situation where there are no pdf's in the directory, and bash otherwise attempts to iterate over ***.pdf** literally

Arithmetic Expansion

Arithmetic Expansion is performed through double parenthesis ((expr)). Consider the following example that uses the $RANDOM environment variable

```
#!/bin/bash

var=$(($RANDOM % 10))

for i in {1..5}
do
  echo "Current Value of Var:${var}"
  var=$(($var + $i))
  let var+=1 # use let for assignment
done
```

Note: arithmetic operators like %, +, −,etc are only available inside arithmetic parenthesis