Computer Practicum 1

Introduction to Bash scripts

Vida Groznik

Source: Ryan‘s tutorials.

Boolean operations

```
#!/bin/bash
# and example

if [ -r $1 ] && [ -s $1 ]
then
    echo This file is useful.
fi
```

```
#!/bin/bash
# or example

if [ $USER == 'bob' ] || [ $USER == 'andy' ]
then
    ls -alh
else
    ls
fi
```

Sometimes we only want to do something if multiple conditions are met.

Other times we would like to perform the action if one of several condition is met.

We can accommodate these with boolean operators.

and - &&

or - ||

Case statements

```
#!/bin/bash
# case example

case $1 in
    start)
        echo starting
        ;;
    stop)
        echo stoping
        ;;
    restart)
        echo restarting
        ;;
    *)
        echo don\'t know
        ;;
esac
```

If we wish to take different paths based upon a variable matching a series of patterns, we could use a series of `if` and `elif` statements but that would soon be unclear.

To make things cleaner, we can use `case` statement.

```
case <variable> in
    <pattern 1>)
        <commands>
        ;;
    <pattern 2>)
        <other commands>
        ;;
esac
```

Case statements (2)

```
1.  #!/bin/bash
2.  # case example
3.
4.  case $1 in
5.      start)
6.          echo starting
7.          ;;
8.      stop)
9.          echo stoping
10.         ;;
11.     restart)
12.         echo restarting
13.         ;;
14.     *)
15.         echo don\'t know
16.         ;;
17. esac
```

- **Line 4** - This line begins the `case` mechanism.
- **Line 5** - If $1 is equal to 'start' then perform the subsequent actions. the `)` signifies the end of the pattern.
- **Line 7** - We identify the end of this set of statements with a double semi-colon (`;;`). Following this is the next case to consider.
- **Line 14** - Remember that the test for each case is a pattern. The `*` represents any number of any character. It is essentially a catch all if none of the other cases match. It is not necessary but is often used.
- **Line 17** - `esac` is case backwards and indicates we are at the end of the case statement. Any other statements after this will be executed normally.

Exercise

Write a script, using case statement to perform basic math operation as follows:
+   addition
-   subtraction
x   multiplication
/   division
The name of script must be 'q4' which works as follows:
`$ ./q4 20 / 3`, Also check for sufficient command line arguments

While loops

```
1. #!/bin/bash
2. # Basic while loop
3.
4. counter=1
5. while [ $counter -le 10 ]
6. do
7.     echo $counter
8.     ((counter++))
9. done
```

`While` an expression/test (the test is placed between square brackets [ ]) is **`true`**, keep executing lines of code:

```
while [ <some test> ]
do
        <commands>
done
```

Line 4 – Initialization of the variable counter with a starting value.

Line 5 – `While` the test is `true` (counter is < or = to 10) `do` the following commands.

Line 7 – Print value of the variable counter.

Line 8 – Increase the value of counter by 1.

Line 9 - We're at the bottom of the loop, go back to line 5 and perform the test again. If the test is true then execute the commands. If the test is false then continue executing any commands following done.

Until loops

```
1. #!/bin/bash
2. # Basic until loop
3.
4. counter=1
5. until [ $counter -gt 10 ]
6. do
7.     echo $counter
8.     ((counter++))
9. done
```

The `until` loop is similar to the while loop. The difference is that it will execute the commands within it **until the test becomes true**.

```
until [ <some test> ]
do
        <commands>
done
```

Line 5 – `Until` the test is `true` (counter is > 10) `do` the following commands.

Line 7 – Print value of the variable counter.

Line 8 – Increase the value of counter by 1.

Exercises

Write a script to print numbers as 5, 4, 3, 2, 1 using while loop.

Write a script to print a given number in a reverse order (eg. if a number is 123 it must print as 321).

Write a script to print a sum of all digits of a given number (eg. if a number is 123 it's sum of all digit will be 1+2+3 = 6).