Bash Scripting
Lesson 3

Working with substitution operator

- A substitution operator (also known as string operator) allows you to manipulate values of variables in an easy way
  - Ensure that variables exist
  - Set default values for variables
  - Catch errors that result from variables that don't exist
  - Remove portions of variable values

For egs:

- **${VAR:-word}**: if $VAR exists, use its value, if not, return the value "word". This does NOT set the variable.
- **${VAR:=word}**: if $VAR exists, use its value, if not, set the default value to "word."
- **${VAR:?message}**: if $VAR exists, show its value. If not, display VAR followed by message. If message is omitted, the message VAR: parameter null or not set will be shown.
- **${VAR:*offset*:*length*}**: if $VAR exists, show the substring of $VAR, starting at *offset* with a length of *length*.

- ```
  # DATE=
  # echo DATE

  # echo ${DATE:-today}
  today
  # echo $DATE
  ```

- ```
  # echo ${DATE:=today}
  today
  # echo $DATE
  today
  ```

- ```
  # DATE=
  # echo ${DATE:?variable not set}
  -bash: DATE: variable not set
  # echo $DATE
  ```

- ```
  # DATE=$(date +%d-%m-%y)
  # echo the day is ${DATE:0:2}
  the day is 05
  ```

Using pattern matching operator

- Pattern Matching is used to *remove* patterns from a variable
- It's an excellent way to clean up variables that have too much information
  - For example, if $DATE contains 05-01-15 and you just need today's year
  - Or if a file has the extension *.doc and you want to rename it to use the extension *.txt
- ${VAR#pattern}: Search for pattern from the beginning of variable's value, delete the shortest part that matches, and return the rest
  - ```
    FILENAME=/usr/bin/blah
    echo ${FILENAME#*/}
    usr/bin/blah
    ```
- ${VAR##pattern}: Search for pattern from the beginning of variable's value, delete the longest part that matches, and return the rest
  - ```
    FILENAME=/usr/bin/blah
    echo ${FILENAME##*/}
    blah
    ```
- ${VAR%pattern}: If pattern matches the end of the variable's value, delete the shortest part that matches, and return the rest
  - ```
    # FILENAME=/usr/bin/blah
    # echo ${FILENAME%/*}
    /usr/bin
    ```
- ${VAR%%pattern}: If pattern matches the end of the variable's value, delete the longest part that matches, and return the rest
  - # FILENAME=/usr/bin/blah
    # echo ${FILENAME%%/*}
    ```

For eg:

```
#!/bin/bash
BLAH=rababarabarabarara

echo BLAH is $BLAH
echo 'The result of ##*ba is' ${BLAH##*ba}
echo 'The result of #*ba is' ${BLAH#*ba}
echo 'The result of %%ba* is' ${BLAH%%ba*}
echo 'The result of %ba* is' ${BLAH%ba*}
```

Output:

```
[root@server1 bin]# vim patterns
[root@server1 bin]# vim patterns
[root@server1 bin]# chmod +x patterns
[root@server1 bin]# ./patterns
BLAH is rababarabarabarara
The result of ##*ba is rara
The result of #*ba is barabarabarara
The result of %%ba* is ra
The result of %ba* is rababarabara
[root@server1 bin]#
```

Understanding Regular Expression

- Regular expressions are search patterns that can be used by *some* utilities (**grep** and other text processing utilities, **awk**, **sed**)
- Regular expressions are NOT the same as shell wildcards
- When using regular expressions, put them between strong quotes so that the shell won't interpret them

Greneric regular expression parsing == grep

Regular expression	Use
^text	Line starts with text
text$	Line ends with text
.	Wildcard (Matches any single character)
[abc], [a-c]	Matches a,b or c
*	Matches 0 to an infinite number of the previous character
\{2\}	Matches exactly 2 of the previous character
\{1,3\}	Match a minimum of 1 and a maximum of 3 of the previous character
colou?r	Match 0 or 1 of the previous character (which makes the previous character optional)

Calculating

- Bash offers different ways to calculate in a script
- Internal calculation: **$((1 + 1))**
- External calculation with **let**:

```
#!/bin/bash
# $1 is the first number
# $2 is the operator
# $3 is the second number
let x="$1 $2 $3"
echo $x
```

- External calculation with **bc**

```
[root@server1 bin]# vim lets
[root@server1 bin]# vim lets
[root@server1 bin]# chmod +x lets
[root@server1 bin]# lets 1 + 2
3
[root@server1 bin]# lets 1 + 20000
20001
[root@server1 bin]# lets 1 - 20000
-19999
[root@server1 bin]# lets  6 / 2
3
[root@server1 bin]# lets  7 / 2
3
[root@server1 bin]#
```

BC:

- **bc** is developed as a calculator with its own shell interface
- It can deal with more than just integers
- Use **bc** in non-interactive mode:
  - echo "scale=9; 10/3"| bc
- Or in a variable:
  - VAR=$(echo "scale=9; 10/3" | bc)

```
[root@server1 bin]# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
10/3
3
scale=9
10/3
3.333333333
^C
(interrupt) Exiting bc.
[root@server1 bin]# echo "scale=9; 10/3" | bc
3.333333333
[root@server1 bin]# NUMBER=$(echo "scale=9 ; 10/3" | bc)
[root@server1 bin]# echo $NUMBER
3.333333333
[root@server1 bin]#
```

Exercise 3

- Write a script that puts the result of the command **date +%d-%m-%y** in a variable. Use Pattern Matching on this variable to show 3 lines, displaying the date, month and year. So the result should look as follows:

```
The day is 05
The month is 01
The year is 15
```

Exercise 3 solution

```
#!/bin/bash
DATE=$(date +%d-%m-%y)
echo the day is ${DATE%%-*}
MONTH=${DATE%-*}
echo the month is ${MONTH#*-}
echo the year is ${YEAR##*-}
```