CSNB113: System Administration

-

8th Topic: Shell Scripts – Programming I

The prompt is a prompt is a prompt ... and a program input

It is possible to **write** and **run** a program **directly** on the command prompt!

```
$ date
Mon Jan 17 10:19:28 MYT 2011
$ echo "I am fine on"
I am fine on
$ echo "I am fine on: "
I am fine on:
$ echo "I am fine on: "`date`
I am fine on: Mon Jan 17 10:22:53 MYT 2011
$
```

'echo' is a command and instructs the shell to print whatever follows the 'echo'
The backward apostrophe (`) executes (runs) a(nother) command

Sequencing of commands

it is possible to run a multitude of commands sequentially. This is done by using a semicolon between the commands:

```
$ echo "I am fine on: "`date`; ls *.pdf; whoami
I am fine on: Mon Jan 17 10:31:02 MYT 2011
System_Admini_Lab1.pdf  System_Admini_Lab3.pdf  System_Admini_Lab5.pdf
System_Admini_Lab2.pdf  System_Admini_Lab4.pdf  System_Admini_Lab6.pdf
udippel
$
```

This can be beautified:

```
$ echo "I am fine on: "`date`; echo; ls *.pdf; echo; echo -n "I am "; whoami
I am fine on: Mon Jan 17 10:37:15 MYT 2011

System_Admini_Lab1.pdf  System_Admini_Lab3.pdf  System_Admini_Lab5.pdf
System_Admini_Lab2.pdf  System_Admini_Lab4.pdf  System_Admini_Lab6.pdf

I am udippel
$
```

Beautified??

```
$ echo "I am fine on: "`date`; echo; ls *.pdf; echo; echo -n "I am "; whoami
```

The **output** was more beautiful (and readable), but the **input** is not.
Plus, if one wants to run it again, it needs to be retyped.
It makes a lot of sense, to put
1 - the lines into a **file**
2 - make the file **executable** (the 'x' at permissions!)
3 - **run** the file (using ./ - dotslash)

```
$ vi my_first_program
```

```
#!/bin/bash
echo "I am fine on: "`date`
echo
ls *.pdf
echo
echo -n "I am "; whoami
```

```
$ chmod 755 my_first_program
$ ./ my_first_program
```

Beautified!

```
#!/bin/bash
# this is my first program
# written in January 2011

echo "I am fine on: "`date`
echo    # creates a new line
ls *.pdf        # list all PDFs
echo    # bla-bla-bla-lah!
echo -n "I am "; whoami
# done!
```

The lines starting with '#' are **comments**

It is also possible to add a comment at the **end** of a line

Only in the first line, the '#' is not really a comment. It is the indication, which **command interpreter** should be used for this program. In this case, it is Bash (and found in /bin/bash on the system.)

Available!?

From now on, this file is available as executable program in your system.
Is it?

```
$ ./my_first_program
I am fine on: Mon Jan 17 11:44:53 SGT 2011

yahoo.pdf

I am udippel
$ mkdir newdir
$ cd newdir/

$ ./my_first_program
bash: ./my_first_program: No such file or directory
$ ls -l
$
```

Why??

Availability

```
$ ./my_first_program
bash: ./my_first_program: No such file or directory
$ whoami
udippel
$

$ which whoami
/usr/bin/whoami
$ whoami| grep di
udippel
$ which grep
/bin/grep
$ which my_first_program
$
```

How does it know some, but the other not? That has to make with the environment; the 'path':

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
$
```

Availability

```
$ ./my_first_program
bash: ./my_first_program: No such file or directory
$ whoami
udippel
$

$ which whoami
/usr/bin/whoami
$ whoami| grep di
udippel
$ which grep
/bin/grep
$ which my_first_program
$
```

How does it know some, but the other not? That has to make with the environment; the 'path':

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
$
```

Available!

Make your new program available system-wide:

```
$ sudo cp my_first_program /usr/local/bin/
$

$ which my_first_program
/usr/local/bin/my_first_program
$ my_first_program
I am fine on: Mon Jan 17 12:26:22 SGT 2011

yahoo.pdf

I am udippel
$
```

This example shows the relevancy of the 'path'. The **path** is an ***environment variable***, that the system administrator can **set** or change.

Shell Variables

It is good convention and practice, but not compulsory, to use ALL UPPERCASE shell variables.

The user can easily define these variables on the command prompt.
To ***substitute*** a variable, it must be **preceded** by a '**$**' (Dollar sign)

```
$ SLOGAN="Uniten generates professionals"
$ echo $SLOGAN
Uniten generates professionals
$ echo SLOGAN
SLOGAN
$
```

Though

```
$ SLOGAN=Uniten generates professionals
```

might also work, it is good practice to enclose ***strings*** with (double) quotes, as above

A shell variable **must not** have a **blank** in its name. Use underscore ('_') instead:

```
$ UNITEN_SLOGAN="Uniten generates professionals"
```

Loops

Loops can be written with 'for' or 'while'.
Here are two examples:

1. **for** – loop:

```
#!/bin/bash
for i in 1 2 3 4 5
do
  echo "This is round $i"
done
```

2. **while** – loop:

```
#!/bin/bash
TEST="no"
while [ "$TEST" != "yes" ];
do
        read TEST
done
```

Decision

Decisions ('selection') is usually implemented with if ... fi.
Here are two examples:

1. **for** – loop:

```
#!/bin/bash
TEST=4
if [ $TEST -lt 5 ]; then
   echo "less than five"
fi
```

```
#!/bin/bash
TEST=6
if [ $TEST -lt 5 ]; then
   echo "less than five"
else
   echo "five or more"
fi
```

Parameter

It is possible to ***parse arguments*** when ***calling*** a shell program.
Uuuh? What's that?

Look at this example. The name of the program is **`just_an_echo`**:

```
#/bin/ksh
echo "arguments for $0 was $1 $2 $3 $4"
```

Shell variable **$0** is the **program name**
Shell variables **$1 to $9** are the **parameters** following the program call:

```
$ ./just_an_echo A B C D
arguments for just_an_echo was A B C D

$ ./just_an_echo A B C D E F
arguments for just_an_echo was A B C D
$ cp just_an_echo another_name
$ ./another_name A B C D E F
arguments for another_name was A B C D
$
```

Parameter

It is possible to ***pass arguments*** when ***calling*** a shell program.
Uuuh? What's that?

Look at this example. The name of the program is **just_an_echo**:

```
#/bin/ksh
echo "arguments for $0 was $1 $2 $3 $4"
```

Shell variable **$0** is the **program name**
Shell variables **$1 to $9** are the **parameters** following the program call:

```
$ ./just_an_echo A B C D
arguments for just_an_echo was A B C D

$ ./just_an_echo A B C D E F
arguments for just_an_echo was A B C D
$ cp just_an_echo another_name
$ ./another_name A B C D E F
arguments for another_name was A B C D
$
```

References

- http://www.google.com.my/
  search for 'shell script'
-