Shell Programming

session 5

Agenda

- Loop control structures

  while, until and for loop

  break and continue statements

- Arrays and functions
- Directory stack manipulation
- Job control history and processes
- Built in functions of shell

Loop constructs

- Three types of loops
    1. while
    2. until
    3. for
- Syntaxes are some what different than we expect but working is same

while loop

- while loop syntax

```
while expression
do
    commands
done
```

- expression is evaluated at the start and after each iteration
- *expression* can be specified using **test** or **[]**
- execute commands inside loop body while *expression* **true**
- do -- start of loop body
- done -- end of loop body

Example while loop

```
#!/bin/bash
# while loop example
# Number reverser
echo -e “Enter a number : \c”
read number
let input=number
reverse=0
while [  “$input” -gt 0 ]; do
        let reverse=reverse*10
        let reverse=reverse+input%10
        let input=input/10
done
echo “Reverse of $number is $reverse”
exit 0
```

until loop

- until loop syntax

```
until *expression*
do
        commands
done
```

- In completely reverse to while loop commands in loop body are executed until *expression* becomes true.
- do -- start of loop body
- done -- end of loop body

until loop example

```
#!/bin/bash
# while loop example
# Number reverser
echo -e “Enter a number : \c”
read number
let input=number
reverse=0
until [  “$input” -eq 0 ]; do
        let reverse=reverse*10
        let reverse=reverse+input%10
        let input=input/10
done
echo “Reverse of $number is $reverse”
exit 0
```

for loop

- For loop syntaxes

1.
```
for <loop-var> in <list>
do
        commands
done
```

2.
```
for (( <loop-counter>=<init-value>;<condition>;<change-loop-counter> ))
do
        commands
done
```

for loop example

- Listing files in current directory

```
 #!/bin/bash                              ## Note : no $ symbol before variable name
for file in *
do
    echo “$file”
done
```

- Example for another variant of for loop

```
 #!/bin/bash
 for (( i=0;i<5;i++ ))                    ## Note : no $ symbol before variable name
 do
         echo “$i”
 done
```

Controlling flow of loop - break, continue

- **break**

  It is used to break out of enclosing loop

  break in shell has one additional feature of breaking out of particular nested loop

- **continue**

  Continue in for loop assign next value to the for loop variable

  In case of while and until it will start from checking expression.

break example

- break-while.sh

```
#!/bin/bash
# request break to take you out of loop

while [ : ]; do                                        ## infinite for loop
        echo “Breaking out of infinite for loop”
        break;
done
```

Functions

- Why do we need them
- is it possible to have function in shell script?
- How functions are used in shell script

functions syntax

- Definition

```
fun()
{
	commands
	## no return type and no arg list
}
```

- Function call

```
fun <arg-list>
```

Example of function

- functions.sh

```
#!/bin/bash
#function definition and call

fun()
{
	echo "I am in function"
	echo "Function Argument 1 : $1"
}

echo "Shell argument 1 : $1"
fun 1
```

Arrays in shell script

- array index start from 0 in shell script
- Ways of declaring array
  1. declare -a var
- Ways of defining array
  1. arr=(element1 element2 element3 … elementn)
  2. arr=([index1]=element1 [index2]=element2 … [indexn]=elementn)
- Reading array

```
read -a arr     # enter elements separated by space, new line will be
                # treated as end of input
```

- Accessing value at particular index of array - ${arr[<index>]}   # curly braces are mandatory
- Number of elements in array : ${#arr[@]}
- Accessing all elements of array

```
for i in ${arr[@]} ; do
        echo "$i"
done
```

Array example

- Shortest element in array

```
#!/bin/bash
# read elements of array and find smallest element
echo -e “Enter elements of array : \c”
read -a arr                    ## arr=(1 2 3 4 5 6) or arr=([0]=1 [5]=6 [2]=3 [1]=2 [3]=4 [4]=5)
let small=${arr[0]}
for i in ${arr[@]} ; do
        if [ “$small” -gt “$i” ]; then
                let small=i
        fi
done
echo “Smallest element in array : $small”
```

Jobs, background and foreground process

- Running process in background and why ?
- Use of /dev/tty in such cases
- Ctrl+Z signal SIGSTOP
- Useful commands

1. bg
2. fg
3. jobs

Directory stack manipulation

- Makes moving through directory very easier
- pushd <directory> -> push directory on directory stack
- popd -> pop top directory off the directory stack
- dirs -> print directory stack
- dirs -p -> print each directory on a line