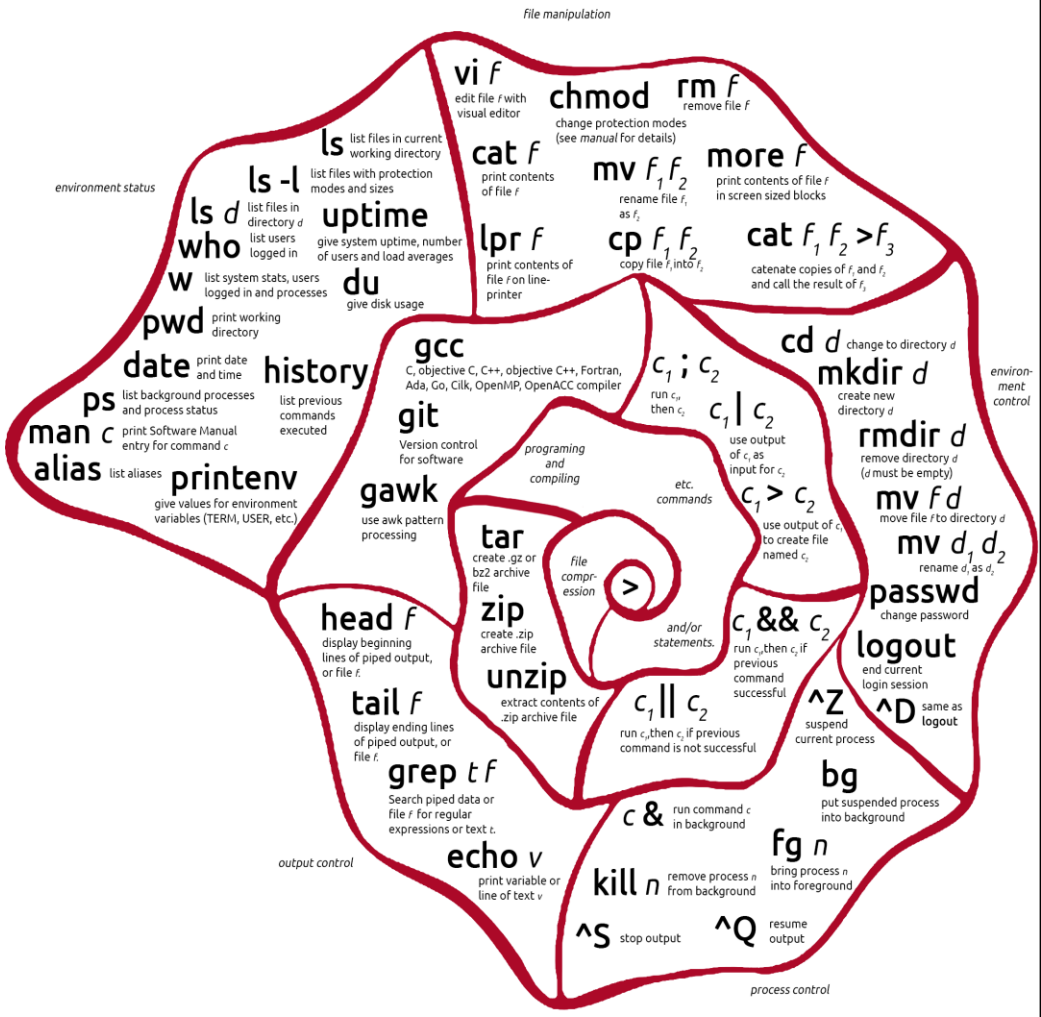
SHELL PROGRAMMING - 2



LINUX SHELL COMMANDS

Another use of brackets [ ]
(alterative for expr)

- [ ] in place of expr
- We generally use:

  var1=`expr 5 \* 2`

- Instead now let's type:

```
var1=$[ 5 * 2 ]                          echo $var1
var2=$[ $var1 + 1 ]                       echo $var2
var3=$[ $var1 * ($var2 - $var1) ]         echo $var3
```

Syntax: **$[operation]**

An example for **File** operation

```
$ cat scr6_1
1 #!/bin/bash
2 if [ -f letter1 ]          #letter1 is the filename
3 then
4       echo "We have found the evidence, here it is!"
5       cat letter1
6 else
7       echo "Keep looking!"
8 fi

$ ./scr6_1
We have found the evidence!
How much would it cost to buy an Apple Computer?
Best,
Bill
```

And, Or, Not

You can **combine and negate expressions** with:

-a	And
-o	Or
!	Not

```
$ cat test10
1 #!/bin/bash
2 if [ `who | grep Mark | wc -l` -ge 1 -a `whoami` != “Bill” ]
3 then
4    echo “Who else but Mark is loading down the machine again!“
5 else
6   echo "All is well!“
7 fi
```

```
$ ./test10
Who else but Mark is loading down
the machine again!
```

And, or and not

You can combine **two conditional checks** with:

&&	**And**
||	**Or**
!	**Not (!= not equal)**

```
$ cat test10
1 #!/bin/bash
2 n1=4 n2=4 n3=4
3 if       [ $n1 -eq $n2 ] && [ $n1 -eq $n3 ]; then
4   echo “Equals rule”
5 else
6 echo “Un-equals do not have much to say!”
7 fi
```

```
$ ./test10
Equals rule
```

The *while* loop

The **`while`** statement loops indefinitely, while the condition is true, such as a user-controlled condition.

Syntax:

```
while [ condition ]
do
        #body of loop
done
```

The **while** loop

```
$ cat test11
1 #!/bin/bash
2 response="no"          #variable creation and assignment
3 while [ $response != "yes" ]
4 do
5         echo "Wakeup [yes/no]?"
6         read response
7 done
```

```
$ ./test11
Wakeup [yes/no]?
no
Wakeup [yes/no]?
y
Wakeup [yes/no]?
yes
$
```

The *while* loop as **normal incrementing loops** ?

```
$ cat factorial
1 #!/bin/bash
2 # Back to factorials!
3 read -p "Enter number: " number
4 fact=1
5 i=1          #control variable
6 while [ $i -le $number ]
7 do
8      fact=`expr $fact \* $i`
9      i=`expr $i + 1`
10 done
11 echo "The factorial of $n is $fact"
```

Is there an alternate to using expr?

```
$ factorial
Enter number:
5
The factorial of 5 is 120
```

Using **break** statement

- The **break** command works like in C++, breaking out of the innermost loop :

```
$ cat test12
1 #!/bin/bash
2 while [ 1 ]
3         do
4    echo "Wakeup [yes/no]?“
5    read resp
6    if [ $resp = "yes" ]
7    then
8         break
9    fi
10  Done
11  echo “Out of while loop”
12  echo “Terminating shell script”
```

True Loop

```
$ ./test12
  Wakeup [yes/no]? no
  Wakeup [yes/no]? Y
  Wakeup [yes/no]?Yes
  Out of while loop
  Terminating shell script
  $
```

The ***date*** command

Wed Feb 23 09:01:09 IST 2022

date **+%z**	Displays numeric time zone (+0530)	
date **+%Z**	Displays alpha time zone (IST or EST)	
date **+%a**	Displays Weekday name in short (like Mon, Tue, Wed)	Wed
date **+%A**	Displays Weekday name in full short (like Monday, Tuesday)	Wednesday
date **+%b**	Displays Month name in short (like Jan, Feb, Mar)	Feb
date **+%B**	Displays Month name in full short (like January, February)	February
date **+%d**	Displays Day of month (e.g., 01)	23

The *date* command

Wed Feb 23 09:02:29 IST 2022

date **+%D**	Displays Current Date; shown in MM/DD/YY	01/23/22
date **+%d**	Displays Current Date; shown in MM/DD/YY	23
date **+%F**	Displays Date; shown in YYYY-MM-DD	2020-01-23
date **+%m**	Displays month (01..12)	02
date **+%Y**	Displays full year i.e. YYYY	2022
date **+%u**	Displays day of week (1..7); 1 is Monday	3
date **+%U**	Displays week number of year, with Sunday as first day of week (00..53)	08
date **+%j**	Displays day of year (001..366)	054

The *date* command with **time**

Wed Feb 24 09:10:18 IST 2022

date **+%T**	Displays time; shown as HH:MM:SS (Hours in 24 Format)	09:10:18
date **+%H**	Displays hour in (00..23) format	09
date **+%I**	Displays hour (01..12) format	09
date **+%M**	Displays minute (00..59)	10
date **+%S**	Displays second (00..60)	18

Now, try to display the date in dd/mm/yyyy format: 23/02/2022

```
$ date +%d\/%m\/%Y
```

Now, try to display the date in dd-mm-yyyy format: 23-02-2022

```
$ date +%d\-%m\-%Y
```

The **exit** command vs. break

- The `exit` command works like in C++, breaking out of the program:

```
$ cat test13
#!/bin/bash
while [ 1 ]
      do
            echo "Wakeup [yes/no]?"
                  read resp
                  if [ $resp = "yes" ]
                  then
                        exit
                  fi
       done
 echo “Out of while loop”
 echo “Terminating shell script”
```

True Loop

```
$ ./test13
     Wakeup [yes/no]?        no
     Wakeup [yes/no]?        y
     Wakeup [yes/no]?        yes
     $
```

Try these shell scripts

1. Write a shell script to find the largest of three numbers
2. Write script to print numbers as 5,4,3,2,1 using while loop
3. Write script to print given number in reverse order
4. Write script to print the sum of digits of a given number
5. Write script to print contents of file from given line number to next given number of lines. For e.g. If we called this script as file4 and run as Print contents of file4 from line number 5 to next 5 line of that file.

The ***until*** loop

- The **until loop** continues running commands as long as condition is true. Once condition is false, the loop is exited.
- *Syntax:*

```
until [ condition ]
do
        //body of loop
done
```

Difference between **while** & **until** loops?

- The ***while*** loop versus the ***until*** loop:
  - The until loop executes until a nonzero status is returned.
  - The while command executes until a zero status is returned.
  - The until loop always executes at least once.

Advanced if-then features

- Double parentheses for mathematical expressions
- Double square brackets for String handling functions

The Double Parentheses Command Symbols

Symbol	Description
val++	post-increment
val--	post-decrement
++*val*	pre-increment
--*val*	pre-decrement
!	logical negation
~	bitwise negation
**	exponentiation

Double parentheses

- Syntax: ((expression))
- **Good news:**
  - a ++ and ++ a valid
  - a - - and - - a valid
  - ! a Valid
  - Logical and && is valid
  - Logical or || is valid

(Script 18)

Double square brackets

- Syntax: [[expression]]

Examples

```
#!/bin/bash
clear
val1=10
if (( val1 ** 2 > 90 ))  #exponentiation
then
        echo "inside if"
        (( val2 = val1 ** 2 ))
        echo $val2
else
        (( val2 = val1 ** 2 ))
        echo "The square of $val1 is $val2"
fi
((val3 = val2 * 4))
echo "The final solution is $val3"
```

Examples

```
# for advanced string manipulation (pattern matching)
#!/bin/bash
#echo $USER
name="andrea"
if [[ name == p?? ]]
then
   echo "Hello $USER"
else
  echo "System detects a new user $name"
fi
```

case – esac construct

- To perform a specific set of instructions depending on a value of a variable:

  General syntax:

```
case $variable-name in
    value1|value2) command ;;
    value3) command ;;
    *) command;;
    esac
```

case – esac construct

```
$ cat test15
1 #!/bin/bash
2 read –n “Enter value of variable:” x
3 case $x in
4 dozen) echo “12”;;
5 score) echo “20”
6 esac
```

$ ./test15
Enter value of variable: dozen 12
Enter value of variable: score 20
Enter value of variable: 34

case – esac construct

```
$ cat test15
1 #!/bin/bash
2 read –n “Enter value of variable:” x
3 case $x in
4 dozen) echo “12”;;
5 score) echo “20”;;
6 *) echo “ Invalid choice!”
7 esac
```

$ ./test15
Enter value of variable: dozen 12
Enter value of variable: score 20
Enter value of variable: 34 Invalid choice

Try these shell scripts now

1. Write a shell script called **listing** that includes the following commands:
   a. Display long listing of files
   b. Display long listing of files including hidden files
   c. Delete files from directory
   d. Exit from shell script only when user enters ‘y’ to “do you wish to quit?”
2. Write a shell script will:
   a. Ask the user to enter a filename and read the file name
   b. Change the FAP for user to executable for the given filename

Try these shell scripts now

1. Write a shell script which accepts two strings and compares the two strings for equality. Display appropriate messages
   - Check if the given strings are empty or not.
   - If empty, display appropriate messages
2. Write a shell script to accept many filenames through command line. Do the following for each filename
   - If it is an ordinary file, display its content and also check whether it has execute permission.
   - If it is directory, display the number of files in it.
   - If the file/directory does not exist, display a message

For loop

```
#!/bin/bash
# another example
for var in Paris "New York" London "New Delhi" Tokyo
do
echo "Now going to $var"
done
```

For loop

```
#!/bin/bash
# iterate through all the files in a directory
for file in ./*  #files in the current directory
do
if [ -d "$file" ]
then
   echo "$file is a directory"
     elif [ -f "$file" ]
     then
          echo "$file is a file"
fi
done
```

For loop

```
#!/bin/bash
#Breaking out of an outer loop
#break n
#n indicates the level of the loop to break out of.
#By default, n is one, indicating to break
# out of the current loop.
clear
for (( a = 1; a < 4; a++ )) #outer for loop a
do
        echo "Outer loop: $a"
                for (( b = 1; b < 100; b++ )) #inner loop b
                do
                        if [ $b -gt 4 ]
                        then
                                break 5
                        fi
                echo " Inner loop: $b"
                done
done
```