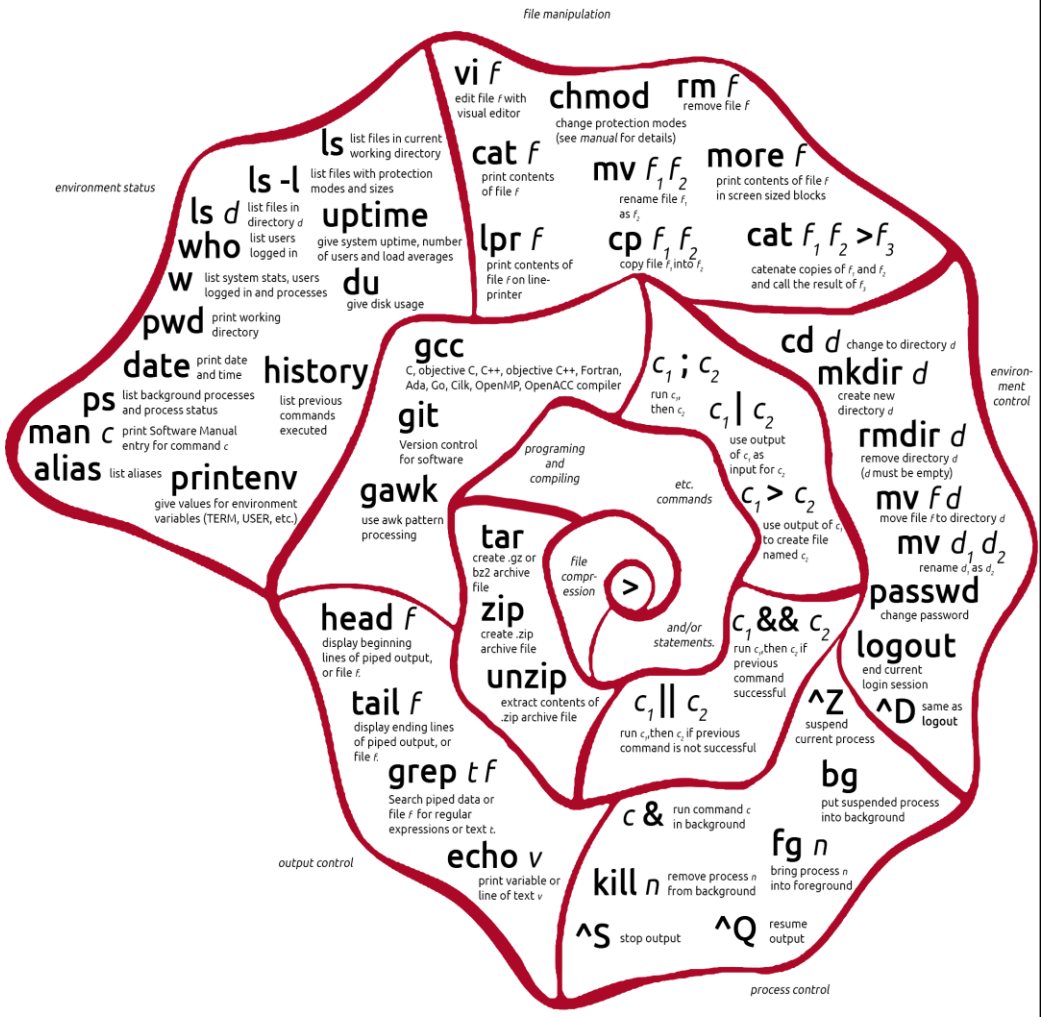
SHELL PROGRAMMING - 1



LINUX SHELL COMMANDS

Shells

- A shell can be used in one of two ways:
  - A *command interpreter*, used interactively
  - A *programming language*, to write shell scripts (your own custom commands)

Shell Scripts

shebang: `#!/bin/bash`
? Use file command

- A shell script is just a file containing shell commands, and **one extra line**
- **First line of a shell script** should be a comment of the following:

```
#!/bin/bash      --absolute path to bash interpreter
```

  – A shell script must be readable and executable.

```
chmod u+rx scriptname
```

Shell Script Example

- Here is a “hello world” shell script:

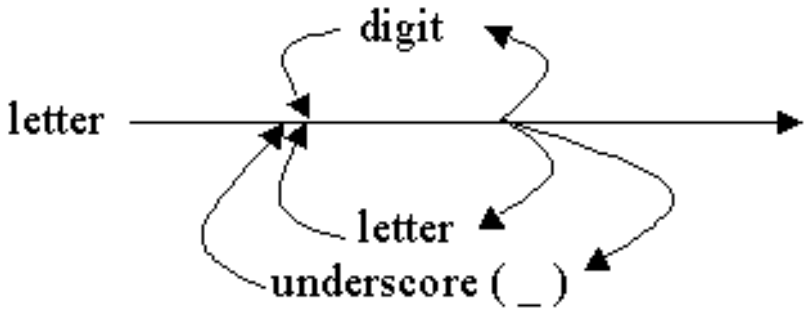
```
$ ls -l
-rwxr-xr-x  1 john john  48 Feb 19 11:50 hello*
$ cat hello
1 #!/bin/bash
2  # comment lines start with the # character
3   echo "Hello world"

$ ./hello
  Hello world
  $
```

  - The **`echo`** command functions like a print command in shell scripts.

Shell Variables

- The user variable name can be any sequence of letters, digits, and the underscore character, but the first cha[illegible]erscore



- To assign a value to a variable:

```
number=25
name="Charles Xavier"
```

- There **cannot be any space** before or after the "="
- Internally, all values are stored as strings.

Shell Variables

- To use a variable, precede the name with a “$”:

```
$ cat test1

1 #!/bin/bash
2 number=25
3 name=“Jeff Bezos"
4 echo "$number $name"

$ ./test1
25 Jeff Bezos
$
```

User Input

- Use the `read` command to get and store input from the user.

```
$ cat test2
1       #!/bin/bash
2       echo "Enter name: "
3       read name
4       echo "How many friends do you have? "
5       read number
6       echo "$name has $number friends!"
```

```
$ test2
Enter name:
Norman Bates
How many friends do you have?
nil
Norman Bates has nil friends!
```

So how do you print $?

- Use ***a backslash before $*** if you really want to print the dollar sign:

```
$ cat test4
1 #!/bin/bash
2 echo "Enter amount: "
3 read cost
4 echo "The total is: \$$cost"
$ ./test4
Enter amount:
18.50
The total is $18.50
```

Use of quotes with $

- You can also use single quotes for printing dollar signs.
- Single quotes turn off the special meaning of all enclosed dollar signs:

```
$ cat test5
1#!/bin/bash
2 echo "Enter amount: "
3 read cost
4 echo ‘The total is: $’ "$cost"

$ ./test5
Enter amount:
18.50
The total is $ 18.50
```

When do I use expr ?

- Shell programming is not good at numerical computation, it is good at text processing.
- However, the `expr` command allows simple integer calculations.
- Here is an interactive bash shell example:

```
$ i=1
$ expr $i + 1
2
```

- To **assign the result** of an `expr` command to another shell variable, surround it with back-quotes:

```
$ i=1
$ i=`expr $i + 1`
$ echo "$i"
2
```

When do I use expr ?

- The * character normally means “all the files in the current directory”, so you need a “\” to use it for multiplication:

```
$ i=2
$ i=`expr $i \* 3`
$ echo $i
6
```

- `expr` also allows you to group expressions, but the “(“ and “)” characters also need to be preceded by backslashes:

```
$ i=2
$ echo `expr 5 + \( $i \* 3 \)`
11
```

`expr` Example

```
$ cat test6
1 #!/bin/bash
2 echo "Enter height of rectangle: "
3 read height
4 echo "Enter width of rectangle: "
5 read width
6 area=`expr $height \* $width`
7 echo "The area of the rectangle is $area"

$ test6
Enter height of rectangle:
10
Enter width of rectangle:
5
The area of the ractangle is 50
$ test6
Enter height of rectangle:
10.1
Enter width of rectangle:
5.1
expr: non-numeric argument
```

Does not work for floats!

When to use grave accent? Command Substitution

- A command or pipeline surrounded by backquotes causes the shell to:
  - Run the command/pipeline
  - Substitute the output of the command/pipeline for everything inside the quotes
- You can use backquotes anywhere:

```
$ whoami
john

$ cat test7
1#!/bin/bash
2 user=`whoami`
3 numusers=`who | wc -l`
4 echo "Hi $user! There are $numusers users logged on."
$ test7
Hi peter! There are 6 users logged on.
```

Back tick
Back quote

User input at prompt?

- How do we get the user input on the same line as the prompt?
- Two ways:

Option 1:

```
1 echo –n “Enter your name?”
2 read name
3 echo “The name is $name!”
```

Option 2:

```
1 read  -p “ Enter your name?” name
2 echo “The name is $name!”
```

More on read?

```
1 read  –s  -p “ Enter your name?” name
2 echo “The name is $name!”
```

```
1 read  -p “ Enter your name?” fname lname
2 echo “The name is $lname $fname!”
```

bc – Bash calculator

- To overcome limitation of bash integer
- Programming language that allows entry of floating point expressions at command line, then interprets the expressions, calculates them and returns result
- Recognises
  - Numbers, variables, comments (/* */), expressions, statement constructs, functions
- Usage at command line: $ bc
- Use quit to exit

Some more
Command Substitution: with bc

```
$ cat test10
      1#!/bin/bash
      2 var1=`echo “scale=4; 3.44/5” | bc`

$ test10
  .6880
```

General form:

variable=`echo “options; expression” | bc`

? Try with variables instead of constants

Try this writing this shell script. Easy.

Write a shell script to create a menu that looks this:

1. date
2. who
3. ls
4. pwd

? Remember your vi commands. We need to start coding scripts on vi.

May be a solution?

```
#!/bin/bash
#Date of Creation: 23 Feb 2022
#Simple menu creation
clear
echo "Options"
echo "1. Today's date"
echo "2. Number of users"
echo "3. List all files"
echo "4. Display current directory"
```

Your first script Voila!

```
1 #!/bin/bash                          ---- shebang
2 # my first script                    ---- comment 1
3 d=`date`                    ---- command 1
4 echo “Today’s  date is : $d”                  ---- command 2
5 echo “ these are the active users” # who – command 3
6 who                                  ----- command 4
7 clear                                ----- command 5
8 ls                          ----- command 6
9 pwd                                  ----- command 7
10 # end of my first script    ---- comment 2
```

Add-on statements ?

Control Flow

The shell allows several control flow statements:

If construct

if-then-fi

if-;then-fi

while construct

for construct

Classes of Condition

Relational operators:

```
-eq   5 -eq 6
-ne   5 -ne 6
-gt   5 -gt 6
-ge   5 -ge 6
-lt   5 -lt 6
-le   5 -le 6
```

String operators:

```
-z string   True if the length of string is zero
-n string   True if the length of string is nonzero
s1 = s2     True if s1 and s2 are the same
s1 != s2    True if s1 and s2 are different
s1 < s2     True if s2 greater than s1
s1 > s2     True if s1 greater than s2
```

Test comparisons for strings checks all punctuation & capital

Classes of Condition

File operators:

-f file	True if file exists and is not a directory
-d file	True if file exists and is a directory
-s file	True if file exists & size > 0 (non-empty)
-w file	True if file exists and is writable
-r file	True if file exists and is readable
-x file	True if file exists and is executable

Conditions: test command & [ ]

Syntax:

test expression

Example:

```
1 if test $num1 –gt 0
2 then
3     echo “Number is positive”
4 else
5    echo “Number is negative”
6 fi
```

Syntax:

[expression **]**

Example:

```
1 if [ $num1 –gt 0 ]
2 then
3        echo “Number is positive”
4 else
5        echo “Number is negative”
6 fi
```

The **if** construct

Syntax:

```
if  command
then
        commands
fi
statement x
```

or

```
if  command;       then
        commands
fi
statement x
```

if

- The if statement works mostly as expected:

```
$ cat test6
1 #!/bin/bash
2 a=6 b=6
3 if [ $a –eq $b ]
4 then
        echo “EQUAL!”
5 fi

$ test6
EQUAL!
```

Spaces before and after the square brackets [ ] required.

if

- another `if` example using strings:

```
$ whoami
  Sawyer
$ cat test7
1 #!/bin/bash
2 user=`whoami`  #using backtick remember command substitution
3 if [ $user = “Specter” ]
4 then
5     echo “Hi Harvey!”
6 fi

$ test7
   Hi Harvey!
```

if then else

- The `if then else` statement is similar:

```
$ cat test7
1 #!/bin/bash
2 user=`whoami`
3       if [ $user = “Clinton” ]
4       then
5               echo "Hi Hillary!
6       else
7               echo "Hi $user!”
8 fi

$ test7
    Hi Trump!
```

if *elif* else

You can also handle a list of cases:

```
$ cat test8
1 #!/bin/bash
2 users=`who | wc -l`
3 if [ $users -ge 4 ]
4 then
        echo "Heavy load"
5 elif [ $users -gt 1 ]
6 then
7       echo "Medium load"
8 else
9       echo "Just me!"
10 fi

$ test8
   Heavy load!
```

Try these shell scripts

- Write a shell script to find if a given number is even or odd
- Write a shell script to accept a user name and display “Good morning” if username is Alice and “Good night” if the username is Peter
- Write a shell script to display grade of a user. If cgpa (user input) is:
  - 7.0, grade ‘C’
  - 8.0, grade ‘B’
  - 9.0 grade ‘A’
  - 10.0 grade ‘A+’, any other cgpa, grade is ‘D’