


Shell Programming

Lec-04



Shell

- A shell or command interpreter in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output.
- It is the interface through which a user works on the programs, commands, and scripts.
- A shell is accessed by a terminal which runs it.
- When you run the terminal, the Shell issues a **command prompt (usually \$)**, where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

Types of Shell

- There are two main shells in Linux:
- 1. The **Bourne Shell**: The prompt for this shell is \$ and its derivatives are listed below:
 - Bourne shell also is known as sh
 - Korn Shell also known as ksh
 - **Bourne Again SHell** also known as bash (most popular)
- **The C shell**: The prompt for this shell is %, and its subcategories are:
 - C shell also is known as csh
 - Tops C shell also is known as tcsh
- You can check your current shell using command
 - echo \$SHELL

Shell Scripts (1)

- Basically, a shell script is a text file with Unix commands in it.
- Shell scripts usually begin with a `#!` and a shell name
 - For example: `#!/bin/sh`
 - If they do not, the user's current shell will be used
- Any Linux command can go in a shell script
 - Commands are executed in order or in the flow determined by control statements.
- Different shells have different control structures
 - The `#!` line is very important
 - We will write shell scripts with the Bourne shell (`sh`)

Shell Scripts (2)

- Why write shell scripts?
 - To avoid repetition:
 - If you do a sequence of steps with standard Unix commands over and over, why not do it all with just one command?
 - To automate difficult tasks:
 - Many commands have subtle and difficult options that you don't want to figure out or remember every time.

Assigning Command Output to a Variable

- Using backquotes, we can assign the output of a command to a variable:

```
#!/bin/sh
```

```
files=`ls`
```

```
echo $files
```

- Very useful in numerical computation:

```
#!/bin/sh
```

```
value=`expr 12345 + 54321`
```

```
echo $value
```

Please keep space in mind ☺

Using expr for Calculations

- Variables as arguments:

```
count=5
```

```
count=`expr $count + 1`
```

```
echo $count
```

```
6
```

- Variables are replaced with their values by the shell!

- expr supports the following operators:

- arithmetic operators: +, -, *, /, %

- comparison operators: <, <=, ==, !=, >=, >

- precedence is the same as C, Java

Control Statements

- Without control statements, execution within a shell script flows from one statement to the next in succession.
- Control statements control the flow of execution in a programming language
- The three most common types of control statements:
 - conditionals: if/then/else, case, ...
 - loop statements: while, for, until, do, ...

Conditionals

- Conditionals are used to “test” something.
 - In Java or C, they test whether a Boolean variable is true or false.
 - In a Bourne shell script, the only thing you can test is whether or not a command is “successful”
- Every well behaved command returns back a **return code**.
 - 0 if it was successful
 - Non-zero if it was unsuccessful (actually 1..255)

The if Statement

- Simple form:

```
if decision_command_1
then
    command_set_1
fi
```


grep returns 0 if it finds something
returns non-zero otherwise



- Example:

```
if grep unix myfile >/dev/null
then
    echo "It's there"
fi
```

redirect to /dev/null so that
"intermediate" results do not get
printed



if and else

```
if grep "UNIX" myfile >/dev/null
then
    echo  UNIX occurs in myfile
else
    echo  No!
    echo  UNIX does not occur in myfile
fi
```

if and elif

```
if grep "UNIX" myfile >/dev/null
then
    echo "UNIX occurs in file"
elif grep "DOS" myfile >/dev/null
then
    echo "Unix does not occur, but DOS does"
else
    echo "Nobody is there"
fi
```

for Loops

- for loops allow the repetition of a command for a specific set of values
- Syntax:

```
for var in value1 value2 ...  
do  
    command_set  
done
```

 - command_set is executed with each value of var (value1, value2, ...) in sequence

for Loop Example (1)

```
#!/bin/sh
```

```
# timestable – print out a multiplication table
```

```
for i in 1 2 3
```

```
do
```

```
  for j in 1 2 3
```

```
  do
```

```
    value=`expr $i \* $j`
```

```
    //In shell, * represents all files in the current directory so use `*`
```

```
    echo -n "$value "
```

```
  done
```

```
  echo
```

```
done
```

for Loop Example (2)

```
#!/bin/sh
files=`ls`
for i in $files
do
    echo -n "$i "
done
```

- Find filenames in files in current directory

for Loop Example (3)

```
#!/bin/sh
```

```
# file-poke – tell us stuff about files
```

```
for i in *; do
```

```
    echo -n "$i "
```

```
done
```

- Same as previous slide, only a little more condensed.