Lecture 5

Redirection operator, file creation, Introduction to shell scripting

Redirection concept with >,>>,<

- Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from a place called standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is also your terminal by default.

- There are three main redirection symbols **>,>>,<**

> Redirector Symbol

- To output Linux-commands result (output of command or shell script) to file.

- Note that if file already exist, it will be overwritten else new file is created

- Syntax:
  - Linux-command > filename

>> Redirector Symbol

- To output Linux-commands result (output of command or shell script) to END of file.

- Note that if file exist , it will be opened and new information/data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created.

- Syntax:
  - Linux-command >> filename

< Redirector Symbol

- To take input to Linux-command from file instead of key-board.

- Syntax:
  - Linux-command < filename

Executing multiple commands in single command line

- There are needs many a time to execute more than one commands on a line the syntax to do this,

- Syntax:
    - command1;command2

Pipe Concept

- You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.
- To make a pipe, put a vertical bar (|) on the command line between two commands.
- “A pipe is nothing but a temporary storage place where the output of one command is stored and then passed as the input for second command. Pipes are used to run more than two commands (Multiple commands) from same command line.”

File Creation

- Create a blank file
  - $ touch abc.txt
- Create multiple files with touch
  - $ touch abc.txt cde.txt xyz.txt
- Create lots and lots of files
  - # Create files with names A to Z
  - $ touch {A..Z}
  - # Create files with names 1 to 20
  - $ touch {1..20}
  - # Create files with extension
  - $ touch {1..1000}.txt

File Creation

- Avoid creating new files
  - $ touch -c hello.txt
- Change file access time - 'a’
  - $ touch -a abc.txt
- Change the modified time '-m’
  - $ touch -m a.txt
- Change access and modification time together
  - $ touch -am a.txt
- Set a specific access/modify time instead of current time
  - $ touch -c -t 202003061115 a.txt
- Use the timestamp of another file as reference
  - $ touch -r ref.txt abc.txt

Shell Introduction

- What is Linux Shell?

  A shell is a special-purpose program designed to read commands typed by a user and execute appropriate programs in response to those commands. Such a program is sometimes known as a command interpreter.

  The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt (% on our systems).

Important shells in market

- Bourne shell (sh): This is the oldest of the widely used shells, and was written by Steve Bourne. It was the standard shell for Seventh Edition UNIX. The Bourne shell contains many of the features familiar in all shells: I/O redirection, pipelines, filename generation (globbing), variables, manipulation of environment variables, command substitution, background command execution, and functions.
- All later UNIX implementations include the Bourne shell in addition to any other shells they might provide.

Important shells in market

C shell (csh): This shell was written by Bill Joy at the University of California at Berkeley. The name derives from the resemblance of many of the flow-control constructs of this shell to those of the C programming language. The C shell provided several useful interactive features unavailable in the Bourne shell, including command history, command-line editing, job control, and aliases. The C shell was not backward compatible with the Bourne shell. Although the standard interactive shell on BSD was the C shell, shell scripts (described in a moment) were usually written for the Bourne shell, so as to be portable across all UNIX implementations.

Important shells in market

- Korn shell (ksh): This shell was written as the successor to the Bourne shell by David Korn at AT&T Bell Laboratories. While maintaining backward compatibility with the Bourne shell, it also incorporated interactive features similar to those provided by the C shell.
- Bourne again shell (bash): This shell is the GNU project’s reimplementation of the Bourne shell. It supplies interactive features similar to those available in the C and Korn shells. The principal authors of bash are Brian Fox and Chet Ramey. Bash is probably the most widely used shell on Linux. (On Linux, the Bourne shell, sh, is actually provided by bash emulating sh as closely as possible.)

What is Shell Script ?

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use command one by one (sequence of ‘n’ number of commands) , the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands. This is known as shell script.

Shell script defined as:” Shell Script is series of command written in plain text file.

Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file.”

Why to Write Shell Script?

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- System Administration part can be also automated.

Passing values from command line:

$0 - Program name

$1 - First argument

$2 - Second arg.

$# - No.of arg

$? - Exit status

```
#print date and time and cmd arg -  today_1.sh
echo "Today is:"
date
echo “No .of arg:”$#
echo “Program name:”$0
```

<u>o/p:</u>

```
[oss@pc021698 ~]$ sh today_1.sh
Today is:
Fri Mar  7 00:28:27 IST 2008
“No .of arg:”0
“Program name:”today_1.sh
```

```
#grep using cmd line -grep_ex2.sh
echo "Entered search string is:"$1
echo "Entered file name :"$2
grep -n "$1" $2
# $? gives exit status of last command
if [ $? -eq 0 ]
then
    echo " Yes,the word present in the file"
else
    echo "No,word not present"
fi
```

o/p:

```
[oss@pc021698 ~]$ sh grep_ex2.sh date today.sh
Entered search string is:date
Entered file name :today.sh
2:date
 Yes,the word present in the file
```

```
#simple for loop
for i in 1 2 3
do
echo "==>$i"
done
#for loop  - for_ex.sh
for file in *.
do
echo “Hi”
done
#simple for loop-3
for (( j = 1 ; j <= 5; j++ ))
    do
          echo -n "$j "
    done
```

```
#print files with extension .c
for file in *.c
do
echo "Filename==>$file"
#place any no.of cmds
cp $file /home/oss/cprogs
done
```

while loop – syntax

```
while [ condition ]
do
    code block;
done
```

```
#while_ex.sh

verify="n"

while [ "$verify" != y ]

do

    echo "Enter option: "

    read option

    echo "You entered $option.  Is this
correct? (y/n)"

    read verify

done
```