Week2-UNIX SHELL

The Unix shell is a program that handles interaction between the user and the system. Many of the commands that are typically thought of as making up the Unix system are provided by the shell. Commands can be saved as files called scripts, which can be executed like a program.

There are four common shells in use:

1. Bourne shell(sh) 2. Korn shell(ksh) 3. C shell(csh) 4. Bourne Again shell(bsh)

Shell Concepts

This section will describe some of the features that are common in all of the shells.

1. **REDIRECTION**
   SYNTAX: OUTPUT
   command > filename
   command >> filename
   INPUT
   command < filename
   EXERCISE: 1. Write a text block into a new/old file
   2. Mail the content of a file to yourself
2. **FILENAME SUBSTIUTION (WILD CARDS)**
   SYNTAX: command …*
   command …?…
   command …[…]…
   EXERCISE: 1.List all the files
   2.List all .c files any subdirectory
   3.List all the files whose extension has only one character
   4.List all the files which starts with either a or A
3. **PIPES**
   SYNTAX: command1 | command2
   EXERCISE: 1.Count the number of users logged on to the system
   2.Count the number of files in the current directory
4. **COMMAND SUBSTITUTION**
   SYNTAX: command …..'command'…..
   EXERCISE: 1.Echo today's date along with the string `The date today is:`
5. **SEQUENCES**
   SYNTAX: command1 ; command2
   EXERCISE: 1.Execute the following commands in sequence:
   i)date ii)ls iii)pwd
   2. Repeat the above problem, but, redirect the output to a file.

6. CONDITIONAL SEQUENCES

SYNTAX: ON SUCCESS
command1 && command2
ON FAILURE
command1 || command2

EXERCISE: 1.Display the output, if the compilation of a program succeeds
2.Display an error message, if the compilation of a program fails

NOTE: All built-in shell commands returns1(0) if they fail(succeed)

7. GROUPING COMMANDS

SYNTAX: (command1; command2; command3….)

EXERCISE: 1.Repeat exercise 5.2, but, redirect the output to the same file.

NOTE: Grouping the commands causes them to be executed by a child shell and the group may be redirected or piped as if it were a simple command

8. BACKGROUND PROCESSING

SYNTAX: command &

EXERCISE: 1.Run a command/ exe file in background by redirecting its input and output

NOTE: Background processing is very useful in performing several tasks simultaneously as long as background tasks do not require standard input and output

9. SHELL PROGRAMS: SCRIPTS

SYNTAX: scriptname

EXERCISE: 1.Write a script to include n different commands

NOTE: A file that contains shell commands is called a script. Before a script can be run, it must be given execute permission by using chmod utility (chmod +x script). To run the script, only type its name. They are useful for strong commonly used sequences of commands to full-blown programs.

10. SUBSHELLS

EXERCISE: 1.Try the following commands:

```
$ pwd
$ (cd /; pwd)
$ pwd
```

NOTE: The current(parent) shell creates a new(child) shell to perform some tasks: execution of group command / script / background process. A subshell has its own current working directory.

11. VARIABLES

EXERCISE: 1.Try the following:

```
$ echo HOME=$HOME, PATH=$PATH
$ echo MAIL = $MAIL
$ echo USER = $USER, SHELL = $SHELL, TERM = $TERM
```

2.Try the following, which illustrates the difference between local and environment variable:

```
$ firstname=Rakesh        ……local varables
$ lastname=Sharma
$ echo $fristname $lastname
$ export lastname         …..make “lastname” an envi var
$ sh                      …..start a child shell
$ echo $firstname $lastname
$ ^D                      …..terminate child shell
$ echo $firstname $lastname
```

3.Try the following, which illustrates the meaning of special local variables:

```
$ cat >script.sh
echo the nameof this script is $0
echo the first argument is $1
echo a list of all the arguments is $*
echo this script places the date into a temporary file
echo called $1.$$
date > $1.$$          # redirect the output of date
ls $1.$$              # list the file
rm $1.$$              # remove the file
^D
$ chmod +x script.sh
$ ./script.sh Rahul Sachin Kumble
```

NOTE: A shell supports two kinds of variables: local and environment variables. Both hold data in a string format. The main difference between them is that when a shell invokes a subshell, the child shell gets a copy of its parent shell’s environment variables, but not its local variables. Environment variables are therefore used for transmitting useful information between parent shells and their children

Few predefined environment variables:

$HOME	pathname of our home directory
$PATH	list of directories to search for commands
$MAIL	pathname of our mailbox
$USER	our username
$SHELL	pathname of our login shell
$TERM	type of your terminal

Creating a local variable:

variableName=value

Special built-in shell variables:

$$	The process ID of the shell
$0	The name of the shell script(if applicable)
$1…$9	$n refers to nth command line argument(if applicable)
$*	A list of all the command-line arguments

12. JOB CONTROL

EXERCISE: 1.Try the following, which illustrates the usage of ps:

```
$ (sleep 10; echo done) &
$ ps
```

2.Try the following, which illustrates the usage of kill:

```
$ (sleep 10; echo done) &
$ kill pid        …..pid is the process id of background process
```

3.Try the following, which illustrates the usage of wait:

```
$ (sleep 10; echo done  1 ) &
$ (sleep 10; echo done  2 ) &
$ echo done 3; wait ;  echo done 4          ….wait for children
```

NOTE: The following two utilities and one built-in command allow the listing / controlling the current processes:

ps: generates a list of processes and their attributes, including their names, process ID numbers, controlling terminals, and owners

kill: allows to terminate a process on the basis of its ID number

wait: allows a shell to wait for one or all of its child processes to terminate

****