Prog-1

#!/bin/bash  
OS=`uname -s`  
if [ “$OS” = “FreeBSD” ]; then  
echo “This Is FreeBSD”  
elif [ “$OS” = “CYGWIN\_NT-5.1” ]; then  
echo “This is Cygwin”  
elif [ “$OS” = “SunOS” ]; then  
echo “This is Solaris”  
elif [ “$OS” = “Darwin” ]; then  
echo “This is Mac OSX”  
elif [ “$OS” = “AIX” ]; then  
echo “This is AIX”  
elif [ “$OS” = “Minix” ]; then  
echo “This is Minix”

elif [ “$OS” = “Linux” ]; then  
echo “This is Linux”  
else  
echo “Failed to identify this OS”  
fi

Prog-2

#!/bin/bash

if [ $# -eq 0 ]

then

echo "$0 : You must give/supply one integers"

exit 1

fi

if test $1 -gt 0

then

echo "$1 number is positive"

else

echo "$1 number is negative"

fi

Prog-3 –Car Rental-case

#!/bin/bash

if [ -z $1 ]

then

rental="\*\*\* Unknown vehicle \*\*\*"

elif [ -n $1 ]

then

# otherwise make first arg as rental

rental=$1

fi

case $rental in

"car") echo "For $rental Rs.20 per k/m";;

"van") echo "For $rental Rs.10 per k/m";;

"jeep") echo "For $rental Rs.5 per k/m";;

"bicycle") echo "For $rental 20 paisa per k/m";;

\*) echo "Sorry, I can not gat a $rental for you";;

esac

#!/bin/bash

hostname

uname

whoami

who

pwd

ps –e

top

logout

#!/bin/bash

echo this $0 command has $# arguments.  
echo They are $\*

The echo command echoes the rest of the line to the screen by default. Within a shell script, $0 denotes the script’s name, $1 denotes the first argument mentioned on the command line and so on.

#!/bin/bash

# Example 0 : While loop. Keeping looping while i is less than 10

# The first line creates a variable. Note that to read a

# variable you need to put a ’$’ before its name

i=0

while [ $i -lt 10 ]

do

echo i is $i

let i=$i+1

done

#!/bin/bash

# example 2: For Loop.  
# Do a letter, word and line count of all the files in  
# the current directory.  
# The ‘\*’ below is expanded to a list of files. The  
# variable ‘file’ successively takes the value of  
# these filenames. Preceding a variable name by ‘$’  
# gives its value.  
for file in \*  
do  
echo "wc $file gives"  
wc $file  
done

# Example 3: If.  
# like the above, but doesn’t try to run wc on directories  
for file in \*  
do  
if [ ! -d $file ] #ie: if $file isn’t a directory  
then  
echo "wc $file gives"  
wc $file  
else  
echo "$file is a directory"  
fi  
done

# Example 4 : Case - a multiple if  
# Move to the user’s home directory  
cd  
for file in .?\*  
do  
#Now check for some common filenames.  
case $file in  
.kshrc) echo "You have a Korn Shell set-up file";;  
.bashrc) echo "You have a Bash Shell set-up file";;  
.Xdefaults) echo "You have an X resource file";;  
.profile) echo "You have a shell login file";;  
esac  
done

#!/bin/bash

colors[1]=red

colors[2]=green

colors[3]=blue

echo The array colors has ${#colors[\*]} elements.

echo They are ${colors[\*]}

*•* Arguments  
#!/bin/sh  
if [ $# = 1 ]  
then  
string="It is "  
ending=""  
else  
string="They are "  
ending="s"  
fi  
echo This $0 command has $# argument${ending}.  
if [ $# != 0 ]  
then  
echo $string $\*  
fi

*•* Adding Numbers  
#!/bin/sh  
echo "input a number"  
read number1  
echo "now input another number"  
read number2  
let answer=$number1+$number2  
echo "$number1 + $number2 = $answer"

*•* Login Counting Script  
#!/bin/sh  
times=$(who | grep $1 | wc -l)  
echo "$1 is logged on $times times."

*•* List Directories  
#!/bin/sh  
for file in $\*  
do  
if [ -d $file ]  
then  
10  
ls $file  
fi  
done

*•* See Script  
#!/bin/sh  
for file in $\*  
do  
if [ -d $file ]  
then  
echo "using ls"  
ls $file  
else  
more $file  
fi  
done

*•* Word-length script  
#!/bin/sh  
echo "Type a word"  
read word  
echo $word is $(echo -n $word | wc -c) letters long  
echo or $word is ${#word} letters long

*•* Safe Copying  
#!/bin/sh  
if [ -f $2 ]  
then  
echo "$2 exists. Do you want to overwrite it? (y/n)"  
read yn  
if [ $yn = "N" -o $yn = "n" ]  
then  
exit 0  
fi  
fi  
cp $1 $2

*•* Mailmerge  
#!/bin/sh  
for name in $(<names)  
do  
sed s/NAME/$name/ <template >letter  
# here you could print the letter file out  
done

**Shell Creation**  
Whenever you invoke a command that is not built into the shell, a new shell process is created which inherits many of the properties of its parent. However, variables and aliases are not inherited unless they are exported. Type “export” and you will see what’s been

$ she=janet ; he=john ; export she he  
$ echo $she $he  
$ ps –f

$ export

$ jobs

$ top

If you type “ps -f” you will get an output something like  
UID PID PPID C STIME TTY TIME CMD  
tpl 6006 31173 0 14:03 pts/3 00:00:00 bash  
tpl 6027 6006 0 14:03 pts/3 00:00:00 ps -f  
tpl 31173 31172 0 09:01 pts/3 00:00:00 -bash  
Notice that you are running 2 bash processes. The PPID column gives the PID of the parent process so you can see that the 2nd shell was started from within the 1st. You can kill this new shell by typing exit, you can suspend the shell by typing <CTRL> Z and fall back into the original shell.

**Signals and Temporary Files**  
A script may need to create temporary files to hold intermediate results. The safest way to  
do this is to use mktemp which returns a currently unused name. The following command  
creates a new file in /tmp.  
newfile=$(mktemp)  
If a script is prematurely aborted (the user may press ˆC for example) it’s good practise  
to remove any temporary files. The trap command can be used to run a tidy-up routine  
when the script (for whatever reason) exits. To see this in action start the following script  
then press ˆC  
newfile=$(mktemp)  
trap "echo Removing $newfile ; rm -f $newfile" 0  
sleep 100

**What's Kernel**  
Kernel is heart of Linux O/S. It manages resources of Linux O/S. Resources means  
facilities available in Linux. For eg. Facility to store data, print data on printer,  
memory, file management etc . Kernel decides who will use this resource, for how  
long and when. It runs your programs. It's Memory resident. It does the following task :-  
● I/O management  
● Process management  
● Device management  
● File management  
● Memory management

**What's Linux Shell**  
Computer understand the language of 0's and 1's called binary language, In early  
days of computing, instruction are provided using binary language, which is difficult for all of us, to read and write. So in O/s there is special program called Shell. Shell accepts your instruction or commands in English and translate it into computers native binary language. This is what Shell Does for US

It's environment provided for user interaction. Shell is an command language  
interpreter that executes commands read from the standard input device  
(keyboard) or from a file. Linux may use one of the following most popular shells  
(In MS-DOS, Shell name is COMMAND.COM which is also used for same purpose,  
but it's not as powerful as our Linux Shells are!)

**Pipes**  
A pipe is a way to connect the output of one program to the input of another  
program without any temporary file.

**Redirection of Standard output/input or Input - Output**  
**redirection**  
Mostly all command gives output on screen or take input from keyboard, but in  
Linux it's possible to send output to file or to read input from file. For e.g. **$ ls**  
command gives output to screen; to send output to file of ls give command , **$ ls**  
**> filename**. It means put output of ls command to filename. There are three main  
redirection symbols >,>>,<  
**(1) > Redirector Symbol**  
Syntax: *Linux-command > filename*  
To output Linux-commands result to file. Note that If file already exist, it will be  
overwritten else new file is created. For e.g. To send output of ls command give  
**$ ls > myfiles**  
Now if 'myfiles' file exist in your current directory it will be overwritten without any  
type of warning. (What if I want to send output to file, which is already exist and  
want to keep information of that file without loosing previous information/data?,  
For this Read next redirector)  
**(2) >> Redirector Symbol**  
Syntax: *Linux-command >> filename*  
To output Linux-commands result 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. For e.g.  
To send output of date command to already exist file give  
**$ date >> myfiles**  
**(3) < Redirector Symbol**  
Syntax: *Linux-command < filename*  
To take input to Linux-command from file instead of key-board. For e.g. To take  
input for cat command give  
**$ cat < myfiles**

Use to perform arithmetic operations For e.g.  
**$ expr 1 + 3**  
**$ expr 2 - 1**  
**$ expr 10 / 2**  
**$ expr 20 % 3 # remainder read as 20 mod 3 and remainder is 2)**  
**$ expr 10 \\* 3 # Multiplication use \\* not \* since its wild card)**  
**$ echo `expr 6 + 3`**

**#!/bin/sh**  
**#**  
**# Script that demos, command line args**  
**#**  
**echo "Total number of command line argument are $#"**  
**echo "$0 is script name"**  
**echo "$1 is first argument"**  
**echo $2 is second argument"**  
**echo "All of them are :- $\*"**

and after that if you give command **$ echo $?**  
it will print nonzero value(>0) to indicate error. Now give command  
**$ ls**  
**$ echo $?**  
It will print 0 to indicate command is successful. Try the following commands and not down there  
exit status  
**$ expr 1 + 3**  
**$ echo $?**  
**$ echo Welcome**  
**$ echo $?**  
**$ wildwest canwork?**  
**$ echo $?**  
**$ date**  
**$ echo $?**  
**$ echon $?**

**$ echo $?**

**#!/bin/sh**  
**#**  
**# Script to see whether argument is positive**  
**#**  
**if test $1 -gt 0**  
**then**  
**echo "$1 number is positive"**  
**fi**

Math- ematical  
Operator in   
Shell Script   
Meaning Normal Arithmetical/  
Mathematical Statements But in Shell  
For test statement  
with if command  
For [ expr ]  
statement with if  
command  
-eq is equal to 5 == 6 if test 5 -eq 6 if expr [ 5 -eq 6 ]  
-ne is not equal to 5 != 6 if test 5 -ne 6 if expr [ 5 -ne 6 ]  
-lt is less than 5 < 6 if test 5 -lt 6 if expr [ 5 -lt 6 ]  
-le is less than or  
equal to 5 <= 6 if test 5 -le 6 if expr [ 5 -le 6 ]  
-gt is greater than 5 > 6 if test 5 -gt 6 if expr [ 5 -gt 6 ]  
-ge  
is greater than or  
equal to 5 >= 6 if test 5 -ge 6 if expr [ 5 -ge 6 ]

**$ cat > menuui**  
**#**  
**# Script to create simple menus and take action according to that selected**  
**# menu item**  
**#**  
**while :**  
**do**  
**clear**  
**echo "-------------------------------------"**  
**echo " Main Menu "**  
**echo "-------------------------------------"**  
**echo "[1] Show Todays date/time"**  
**echo "[2] Show files in current directory"**  
Linux Shell Script Tutorial  
http://www.freeos.com/guides/lsst/maspc.htm (6 of 17) [17/08/2001 17.42.32]  
**echo "[3] Show calendar"**  
**echo "[4] Start editor to write letters"**  
**echo "[5] Exit/Stop"**  
**echo "======================="**  
**echo -n "Enter your menu choice [1-5]: "**  
**read yourch**  
**case $yourch in**  
**1) echo "Today is `date` , press a key. . ." ; read ;;**  
**2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ; read ;;**  
**3) cal ; echo "Press a key. . ." ; read ;;**  
**4) vi ;;**  
**5) exit 0 ;;**  
**\*) echo "Opps!!! Please select choice 1,2,3,4, or 5";**  
**echo "Press a key. . ." ; read ;;**  
**esca**  
**done**

yesno box using dialog utility  
**$ cat > dia3**  
**dialog --title "Alert : Delete File" --backtitle "Linux Shell Script\**  
**Tutorial" --yesno "\nDo you want to delete '/usr/letters/jobapplication'\**  
**file" 7 60**  
**sel=$?**  
**case $sel in**  
**0) echo "You select to delete file";;**  
**1) echo "You select not to delete file";;**  
**255) echo "Canceled by you by pressing [ESC] key";;**  
**esac**

inputbox using dialog utility  
**$ cat > dia4**  
**dialog --title "Inputbox - To take input from you" --backtitle "Linux Shell\**  
**Script Tutorial" --inputbox "Enter your name please" 8 60 2>/tmp/input.$$**  
**sel=$?**  
**na=`cat /tmp/input.$$`**  
**case $sel in**  
**0) echo "Hello $na" ;;**  
**1) echo "Cancel is Press" ;;**  
**255) echo "[ESCAPE] key pressed" ;;**  
**esac**  
**rm -f /tmp/input.$$**

**$ cat > smenu**  
**#**  
**#How to create small menu using dialog**  
**#**  
**dialog --backtitle "Linux Shell Script Tutorial " --title "Main\**  
**Menu" --menu "Move using [UP] [DOWN],[Enter] to\**

**Select" 15 50 3\**  
**Date/time "Shows Date and Time"\**  
**Calendar "To see calendar "\**  
**Editor "To start vi editor " 2>/tmp/menuitem.$$**  
**menuitem=`cat /tmp/menuitem.$$`**  
**opt=$?**  
**case $menuitem in**  
**Date/time) date;;**  
**Calendar) cal;;**  
**Editor) vi;;**  
**esac**  
**rm -f /tmp/menuitem.$$**