#!/bin/bash

echo "Printing text"

echo -n "Printing text without newline"

echo -e "\nRemoving \t special \t characters\n"

Run the script to see what it does. The **-e** option is used for telling echo that the string passed to it contains special characters and requires extended functionality.

**Adding two numbers:**

#!/bin/bash

Echo -n "Enter first value num1:"

read num1

echo -n "Enter second value num2:"

read num2

sum=$(($num1 + $num2))

echo " $sum"

**While:**

#!/bin/bash

i=5

while [ $i -ge 0 ]

do

echo " $i "

((--i))

done

**For loop:**

#!/bin/bash

for (( count=0; count<=10; count++ ))

do

echo " $count "

done

**If loop:**

#!/bin/bash

Echo -n "Enter the number: "

read num

if [[ $num -gt 5 ]]

then

echo –n "given number is greater than 5"

else

echo -n "given number is lesser than 5"

fi

**Odd or Even ::**

#!/bin/bash

echo –n "Enter the number: "

read num

if [[ **( $num -le 10 ) && ( $num%2 -eq 0 )** ]]

then

echo "given number is even $num "

else

echo "given number is odd $num "

fi

**IF and ELIF**

#!/bin/bash

echo "Enter the number: "

read num

if [[ $num -le 10 ]]

then

echo " Given number is within ten "

elif [[ ( $num -gt 10 ) && ( $num -le 20 ) ]]

then

echo "Given number is above ten and lesser than 20"

else

echo "Given number is above twenty"

fi

**CASE ::**

#!/bin/bash

echo "Enter the number: "

read num

case $num in

100)

echo "Given num is "hundread"" ;;

200)

echo "Given num is "two hundread"" ;;

\*)

echo "Nither 100 nor 200" ;;

esac

**Special variables:**

#!/bin/bash

echo "File name :$0 "

echo "position of an arguments :$n "

echo "total numner of arguments :$# "

echo "all arguments :$\* "

echo "all argumets :$@ "

echo "exit status of last cmd :$? "

echo "process no of current shell :$$ "

echo "process no of last background job :$! "

**Concatenate string:**

#!/bin/bash

echo "Enter first string: "

read string1

echo "Enter second string: "

read string2

string**=$string1$string2**

echo " $string "

**Functions:**

#!/bin/bash

function Add()

{

echo -n "Enter first number: "

read a

echo -n "Enter second number: "

read b

sum=$(($a+$b))

echo "sum is $sum "

}

Add

function Sub()

{

echo -n "Enter first number: "

read a

echo -n "Enter second number: "

read b

sub=$(($a-$b))

echo "sub is $sub "

}

Sub

**Create directories by given input:**

#!/bin/bash

echo -n "Enter directory name: "

read dir

**if [ -d $dir ]**

then

echo "Directory exists"

else

`mkdir -p $dir`

echo "Directory created"

fi

**Appending to Files**

The below shell script example will show you how to append data to a file on your filesystem using bash scripts. It adds an additional line to the earlier editors.txt file.

#!/bin/bash

echo "Before appending the file"

cat editors.txt

echo "6. NotePad++" >> editors.txt

echo "After appending the file"

cat editors.txt

#### ****Test File Existence****

#!/bin/bash

**filename=$1**

**if [ -f $filename ]**

then

echo "file exits"

else

echo "file doesn't exits"

fi

#### ****The Sleep Command****

#!/bin/bash

echo "How long to wait?"

read time

sleep $time

echo "Waited for $time seconds!"

#### ****The Wait Command****

#!/bin/bash

echo "Testing wait command"

sleep 5 &

pid=$!

kill $pid

wait $pid

echo $pid was terminated.

**Displaying the Last Updated File**

Sometimes you might need to find the last updated file for certain operations. The following simple program shows us how to do this in bash using the awk command. It will list either the last updated or created file in your current working directory.

#!/bin/bash

ls -lrt | grep ^- | awk 'END{print $NF}'

#### ****Print Number of Files or Directories****

#!/bin/bash

**if [ -d "$@" ]**

then

**echo "Files found: $(find "$@" -type f | wc -l)"**

**echo "Folders found: $(find "$@" -type d | wc -l)"**

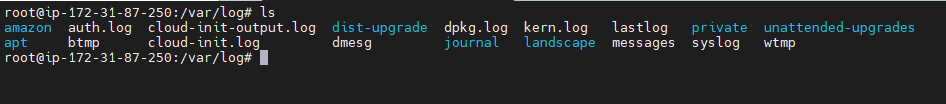
else

echo "ERROR please retry with another folder"

exit 1

fi

#### ****Cleaning Log Files****



#!/bin/bash

LOG\_DIR=/var/log

cd $LOG\_DIR

cat /dev/null > messages

cat /dev/null > wtmp

echo "logs cleaned"

**Check Whether You’re Root**

The below example demonstrates a quick way to find out whether a user is root or not from Linux bash scripts.

#!/bin/bash

ROOT\_UID=0

if [ "$UID" -eq "$ROOT\_UID" ]

then

echo "You are root."

else

echo "You are not root"

fi

exit 0

**Removing Duplicate Lines from Files**

File processing takes considerable time and hampers the productivity of admins in many ways. Searching for duplicates in your files can become a daunting task. Luckily, you can do this with a short shell script.

#! /bin/sh

echo -n "Enter Filename-> "

read filename

if [ -f "$filename" ]; then

**sort $filename | uniq | tee sorted.txt**

else

echo "No $filename in $pwd...try again"

fi

exit 0

The above script goes line by line through your file and removes any duplicative line. It then places the new content into a new file and keeps the original file intact.

**Shell script to do disk cleanup?**

#### ****System Maintenance****

I often use a little Linux shell script to upgrade my system instead of doing it manually. The below simple shell script will show you how to do this.

#!/bin/bash

echo -e "\n$(date "+%d-%m-%Y --- %T") --- Starting work\n"

apt-get update

apt-get -y upgrade

apt-get -y autoremove

apt-get autoclean

echo -e "\n$(date "+%T") \t Script Terminated"

The script also takes care of old packages that are no longer needed. You need to run this script using sudo else it will not work properly.