# Name: Nagaraju Emp.ID: 213495

# Title: Bash Scripting

Contents

[Title: Bash Scripting 1](#_Toc111414531)

[Steps: 1](#_Toc111414532)

[1) Hello World Bash Shell Script 1](#_Toc111414533)

[2) Simple Backup bash shell script 2](#_Toc111414534)

[3) Variables in Bash scripts 2](#_Toc111414535)

[4) Global vs. Local variables 2](#_Toc111414536)

[5) Passing arguments to the bash script 3](#_Toc111414537)

[6) Executing shell commands with bash 4](#_Toc111414538)

[7) Reading User Input 4](#_Toc111414539)

[8) Bash Trap Command 5](#_Toc111414540)

[9) Arrays 7](#_Toc111414541)

[10) Read file into bash array 8](#_Toc111414542)

[**11)** Bash if / else / fi statements 9](#_Toc111414543)

[12) Nested if/else 10](#_Toc111414544)

[13) Bash Comparisons 11](#_Toc111414545)

[14) Bash File Testing 14](#_Toc111414546)

[15) Loops 15](#_Toc111414547)

[16) Bash Functions 19](#_Toc111414548)

[17) Bash Select 20](#_Toc111414549)

[18) Case statement conditional 21](#_Toc111414550)

[19) Bash quotes and quotations 22](#_Toc111414551)

[20) Arithmetic Operations 24](#_Toc111414552)

[21) Redirections 28](#_Toc111414553)

## Steps:

### Hello World Bash Shell Script

* The following command reveals that the Bash shell is stored in /bin/bash.

$ which bash

/bin/bash

* Hello world bash shell script

*#!/bin/bash*

*# declare STRING variable*

*STRING="Hello World"*

*# print variable on a screen*

*echo $STRING*

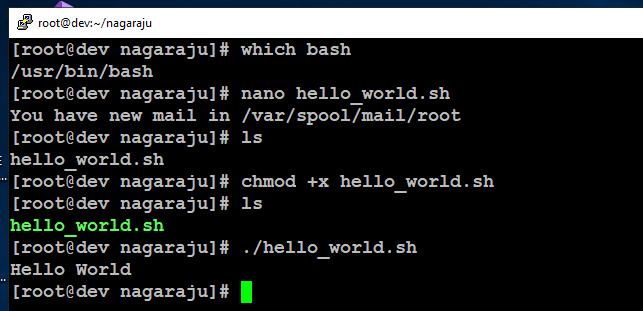


Figure 1: Hello World Script Execution

### Simple Backup bash shell script

*#!/bin/bash*

*tar -czf myhome\_directory.tar.gz /home/ansibleraju*

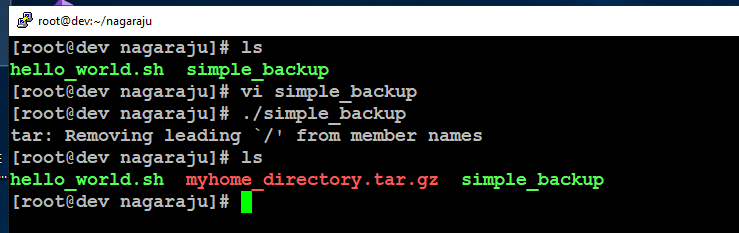


Figure 2: Simple Backup script file creation

### Variables in Bash scripts

*#!/bin/bash*

*STRING="HELLO WORLD!!!"*

*echo $STRING*

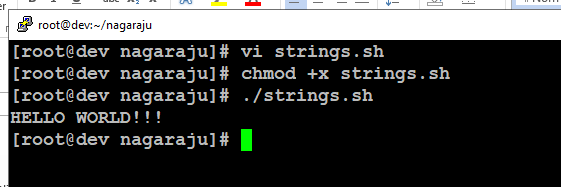


Figure 3: Variables script execution

### Backup with Timestamp:

*#!/bin/bash*

*OF=myhome\_directory\_$(date +%Y%m%d).tar.gz*

*tar -czf $OF /home/ansibleraju*

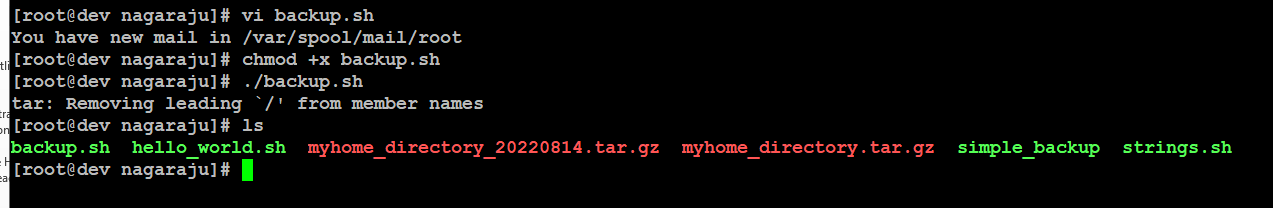


Figure 4: backup with timestamp script exection

### Global vs. Local variables

*#!/bin/bash*

*# Define bash global variable*

*# This variable is global and can be used anywhere in this bash script*

*VAR="global variable"*

*function bash {*

*# Define bash local variable*

*# This variable is local to bash function only*

*local VAR="local variable"*

*echo $VAR*

*}*

*echo $VAR*

*bash*

*# Note the bash global variable did not change*

*# "local" is bash reserved word*

*echo $VAR*

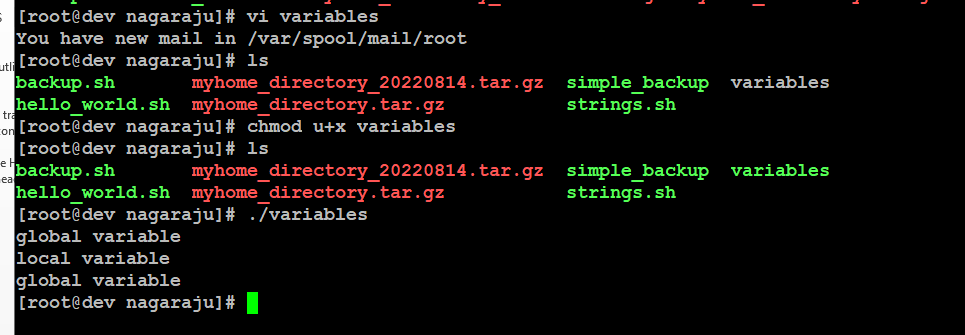


Figure 5: using global and local variables script executed

### Passing arguments to the bash script

*#!/bin/bash*

*# use predefined variables to access passed arguments*

*#echo arguments to the shell*

*echo $1 $2 $3 ' -> echo $1 $2 $3'*

*# We can also store arguments from bash command line in special array*

*args=("$@")*

*#echo arguments to the shell*

*echo ${args[0]} ${args[1]} ${args[2]} ' -> args=("$@"); echo ${args[0]}*

*${args[1]} ${args[2]}'*

*#use $@ to print out all arguments at once*

*echo $@ ' -> echo $@'*

*# use $# variable to print out*

*# number of arguments passed to the bash script*

*echo Number of arguments passed: $# ' -> echo Number of arguments passed: $#'*

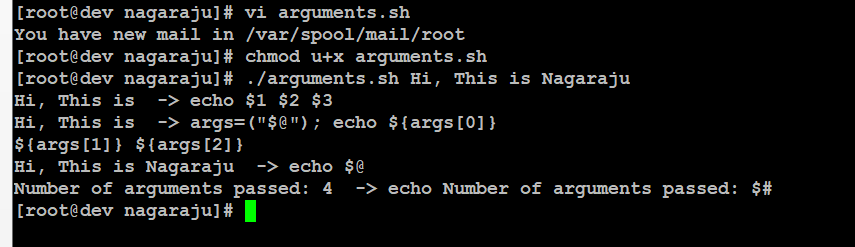


Figure 6: Passing arguments to the script

### Executing shell commands with bash

*!/bin/bash*

*# use a subshell $() to execute shell command*

*echo $(uname -o)*

*# executing bash command without subshell*

*echo uname –o*

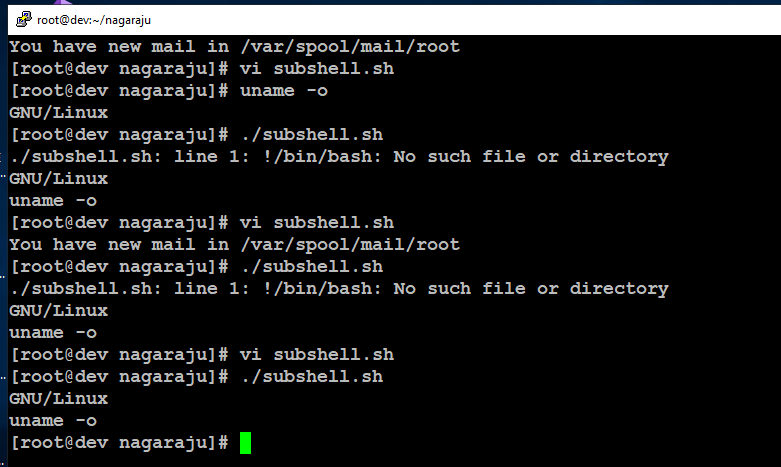


Figure 7: Executing shell commands with bash

### Reading User Input

*#!/bin/bash*

*echo -e "Hi, please type the word: \c "*

*read word*

*echo "The word you entered is: $word"*

*echo -e "Can you please enter two words? "*

*read word1 word2*

*echo "Here is your input: \"$word1\" \"$word2\""*

*echo -e "How do you feel about bash scripting? "*

*# read command now stores a reply into the default build-in variable $REPLY*

*read*

*echo "You said $REPLY, I'm glad to hear that! "*

*echo -e "What are your favorite colours ? "*

*# -a makes read command to read into an array*

*read -a colours*

*echo "My favorite colours are also ${colours[0]}, ${colours[1]} and*

*${colours[2]}:-)"*

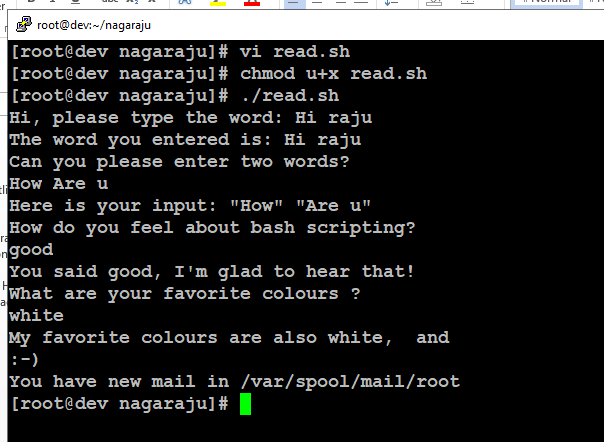


Figure 8: reading user input script

### Bash Trap Command

*#!/bin/bash*

*# bash trap command*

*trap bashtrap INT*

*# bash clear screen command*

*clear;*

*# bash trap function is executed when CTRL-C is pressed:*

*# bash prints message => Executing bash trap subrutine !*

*bashtrap()*

*{*

*echo "CTRL+C Detected !...executing bash trap !"*

*}*

*# for loop from 1/10 to 10/10*

*for a in `seq 1 10`; do*

*echo "$a/10 to Exit."*

*sleep 1;*

*done*

*echo "Exit Bash Trap Example!!!"*

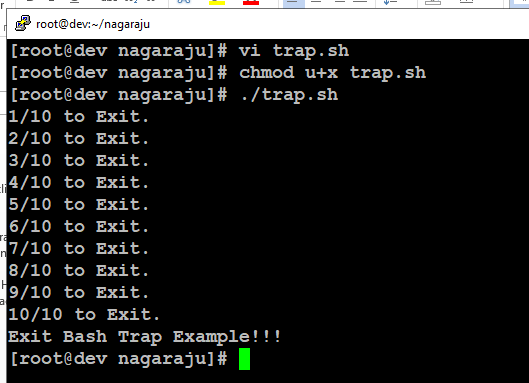
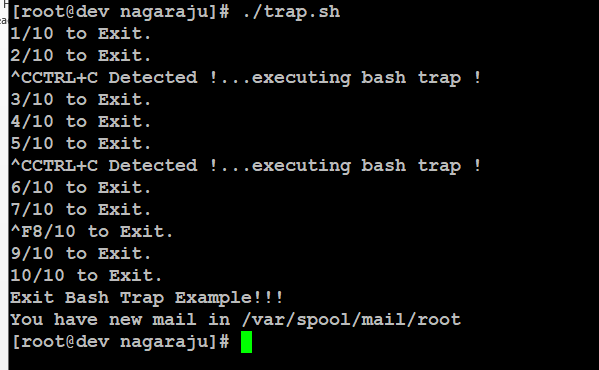


Figure 9: Bash trap command script execution



### Arrays

##### Declare simple bash array

*#!/bin/bash*

*#Declare array with 4 elements*

*ARRAY=( 'Debian Linux' 'Redhat Linux' Ubuntu Linux )*

*# get number of elements in the array*

*ELEMENTS=${#ARRAY[@]}*

*# echo each element in array*

*# for loop*

*for (( i=0;i<$ELEMENTS;i++)); do*

*echo ${ARRAY[${i}]}*

*done*

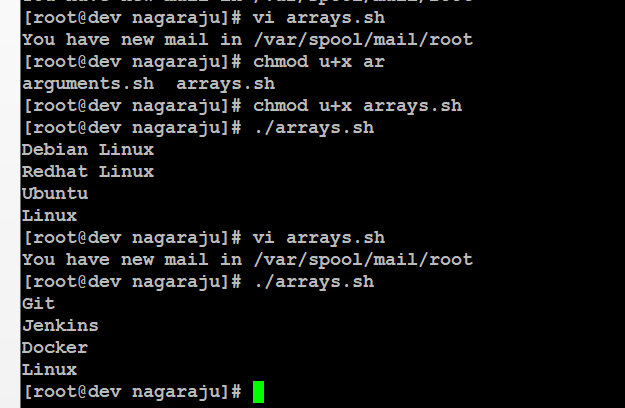


Figure 10: Declare simple bash array

### Read file into bash array

*#!/bin/bash*

*# Declare array*

*declare -a ARRAY*

*# Link filedescriptor 10 with stdin*

*exec 10<&0*

*# stdin replaced with a file supplied as a first argument*

*exec < $1*

*let count=0*

*while read LINE; do*

*ARRAY[$count]=$LINE*

*((count++))*

*done*

*echo Number of elements: ${#ARRAY[@]}*

*# echo array's content*

*echo ${ARRAY[@]}*

*# restore stdin from filedescriptor 10*

*# and close filedescriptor 10*

*exec 0<&10 10<&-*

$ cat bash.txt

Bash

Scripting

Tutorial

Guide

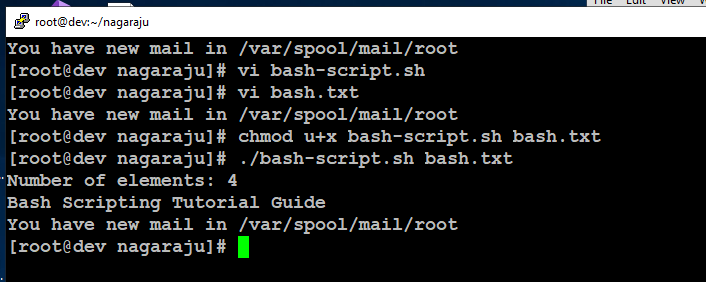


Figure 11: Read file into bash array

### Bash if / else / fi statements

*#!/bin/bash*

*directory="./BashScripting"*

*# bash check if directory exists*

*if [ -d $directory ]; then*

*echo "Directory exists"*

*else*

*echo "Directory does not exist"*

*fi*

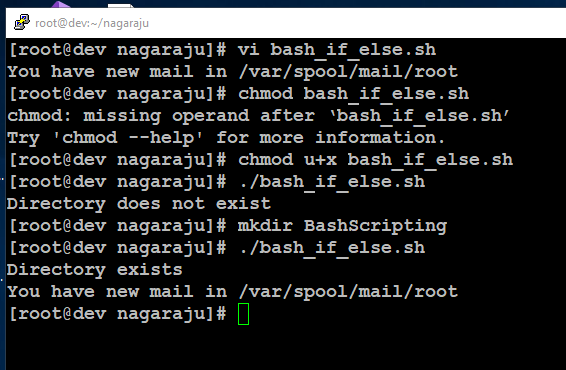


Figure 12: if / else / fi statements

### Nested if/else

*#!/bin/bash*

*# Declare variable choice and assign value 4*

*choice=4*

*# Print to stdout*

*echo "1. Bash"*

*echo "2. Scripting"*

*echo "3. Tutorial"*

*echo -n "Please choose a word [1,2 or 3]? "*

*# Loop while the variable choice is equal 4*

*# bash while loop*

*while [ $choice -eq 4 ]; do*

*# read user input*

*read choice*

*# bash nested if/else*

*if [ $choice -eq 1 ] ; then*

*echo "You have chosen word: Bash"*

*else*

*if [ $choice -eq 2 ] ; then*

*echo "You have chosen word: Scripting"*

*else*

*if [ $choice -eq 3 ] ; then*

*echo "You have chosen word: Tutorial"*

*else*

*echo "Please make a choice between 1-3 !"*

*echo "1. Bash"*

*echo "2. Scripting"*

*echo "3. Tutorial"*

*echo -n "Please choose a word [1,2 or 3]? "*

*choice=4*

*fi*

*fi*

*fi*

*done*

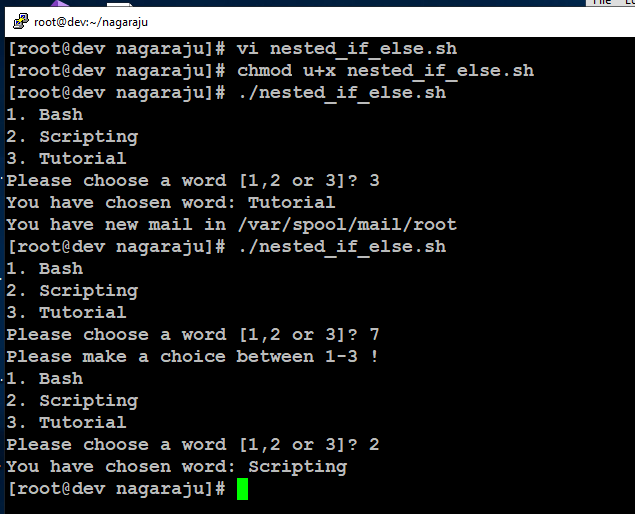


Figure 13: Nested if/else script

### Bash Comparisons

#### Arithmetic Comparisons

*#!/bin/bash*

*# declare integers*

*NUM1=2*

*NUM2=2*

*if [ $NUM1 -eq $NUM2 ]; then*

*echo "Both values are equal"*

*else*

*echo "Values are NOT equal"*

*fi*

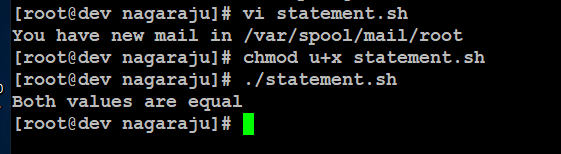


Figure 14: Arithmetic Comparisons script output1

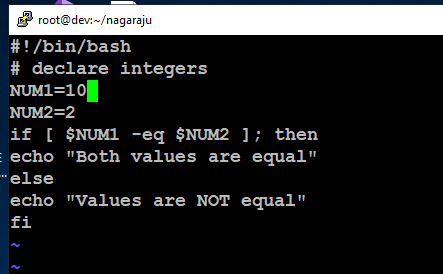


Figure 15: Arithmetic Comparisons script updated

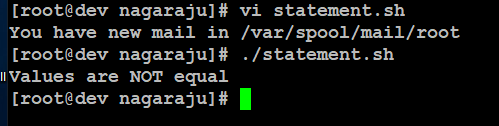


Figure 16: Arithmetic Comparisons script updated output2

* Changing one of the numbers.

*#!/bin/bash*

*# declare integers*

*NUM1=2*

*NUM2=1*

*if [ $NUM1 -eq $NUM2 ]; then*

*echo "Both values are equal"*

*elif [ $NUM1 -gt $NUM2 ]; then*

*echo "NUM1 is greater than NUM2"*

*else*

*echo "NUM2 is greater than NUM1"*

*fi*

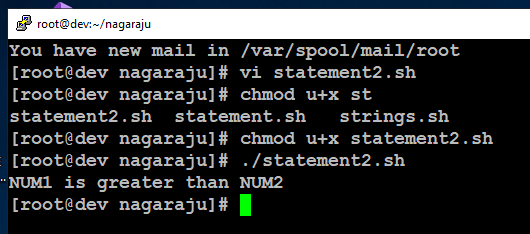
**

Figure 17: Changed the number and executed

#### String Comparisons

*#!/bin/bash*

*#Declare string S1*

*S1="Bash"*

*#Declare string S2*

*S2="Scripting"*

*if [ $S1 = $S2 ]; then*

*echo "Both Strings are equal"*

*else*

*echo "Strings are NOT equal"*

*fi*

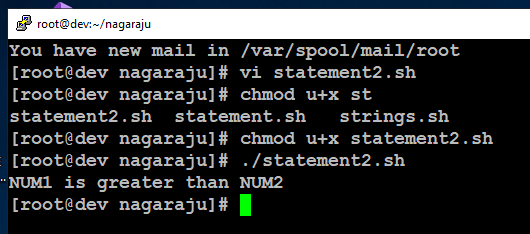


Figure 18: String Comparisons

* Updated Script

*#!/bin/bash*

*#Declare string S1*

*S1="Bash"*

*#Declare string S2*

*S2="Bash"*

*if [ $S1 = $S2 ]; then*

*echo "Both Strings are equal"*

*else*

*echo "Strings are NOT equal"*

*fi*

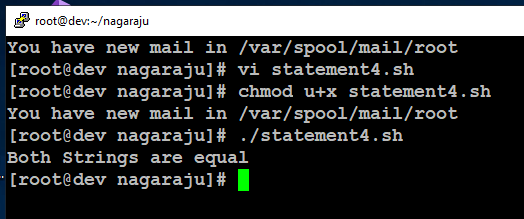


Figure 19: values updated in String Comparisons script

### Bash File Testing

*#!/bin/bash*

*file="./file"*

*if [ -e $file ]; then*

*echo "File exists"*

*else*

*echo "File does not exist"*

*fi*

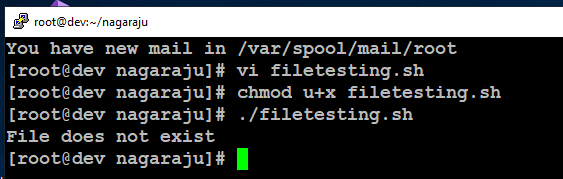


Figure 20: File Testing(exists or not)

$ touch file

$ ./filetesting.sh

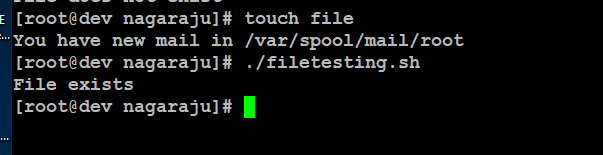


Figure 21: Created file and executed script

* Using while loop

*#!/bin/bash*

*while [ ! -e myfile ]; do*

*# Sleep until file does exists/is created*

*sleep 1*

*done*

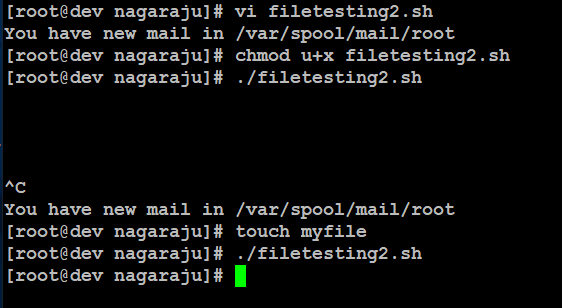


Figure 22: Using while loop if file not present goes to sleep

### Loops

#### Bash for loop

*#!/bin/bash*

*# bash for loop*

*for f in $( ls /var/ ); do*

*echo $f*

*done*

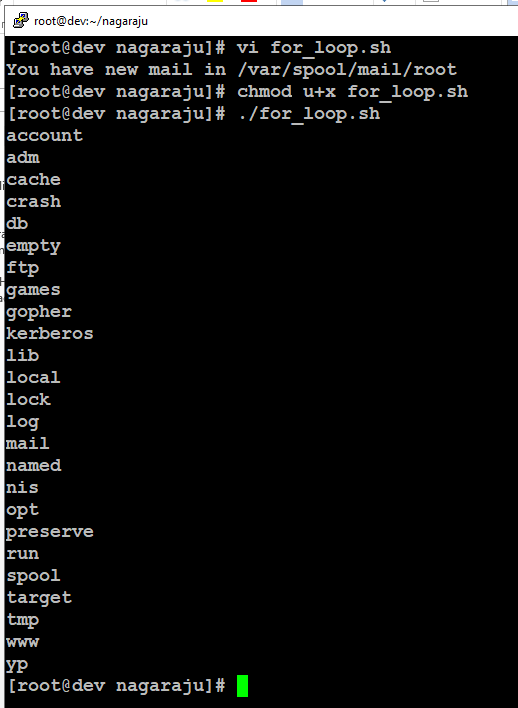


Figure 23: loops script executed

With command

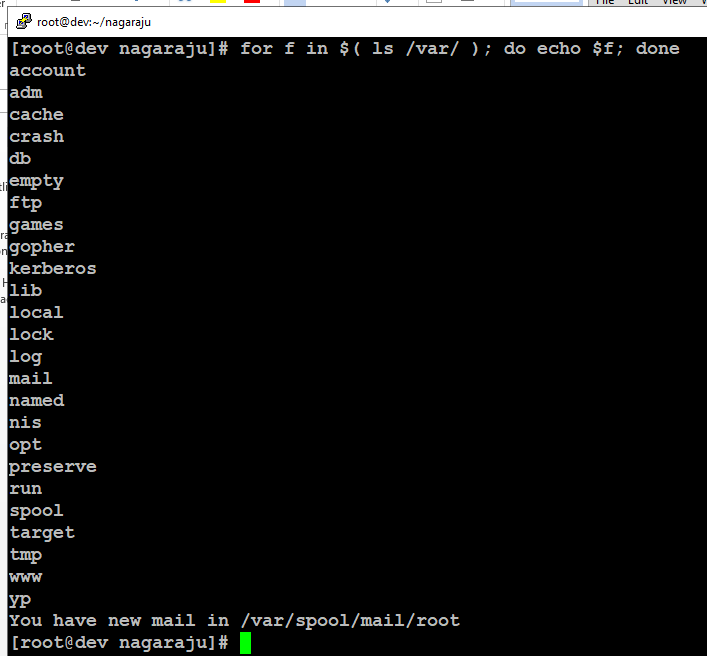


Figure 24: with command executed loops

#### Bash while loop

*#!/bin/bash*

*COUNT=6*

*# bash while loop*

*while [ $COUNT -gt 0 ]; do*

*echo Value of count is: $COUNT*

*let COUNT=COUNT-1*

*done*

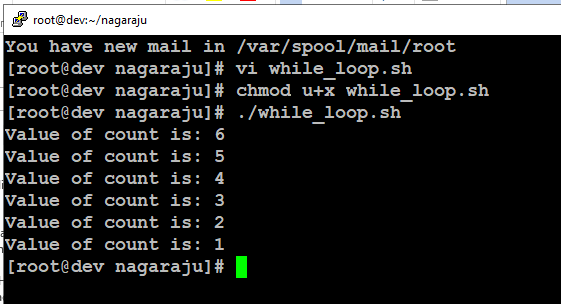


Figure 25: using while loop script executed

#### Bash until loop

*# bash until loop*

*until [ $COUNT -gt 5 ]; do*

*echo Value of count is: $COUNT*

*let COUNT=COUNT+1*

*done*

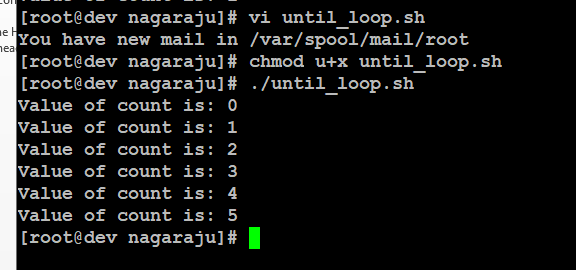


Figure 26: Using until loop script executed

#### Control bash loop with input

*#!/bin/bash*

*# This bash script will locate and replace spaces*

*# in the filenames*

*DIR="."*

*# Controlling a loop with bash read command by redirecting STDOUT as*

*# a STDIN to while loop*

*# find will not truncate filenames containing spaces*

*find $DIR -type f | while read file; do*

*# using POSIX class [:space:] to find space in the filename*

*if [[ "$file" = \*[[:space:]]\* ]]; then*

*# substitute space with "\_" character and consequently rename the file*

*mv "$file" `echo $file | tr ' ' '\_'`*

*fi;*

*# end of while loop*

*done*

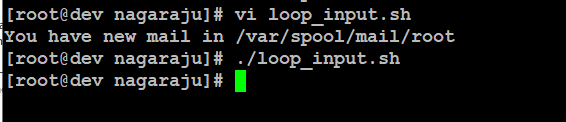


Figure 27: Control bash loop with input

### Bash Functions

*!/bin/bash*

*# BASH FUNCTIONS CAN BE DECLARED IN ANY ORDER*

*function function\_B {*

*echo Function B.*

*}*

*function function\_A {*

*echo $1*

*}*

*function function\_D {*

*echo Function D.*

*}*

*function function\_C {*

*echo $1*

*}*

*# FUNCTION CALLS*

*# Pass parameter to function A*

*function\_A "Function A."*

*function\_B*

*# Pass parameter to function C*

*function\_C "Function C."*

*function\_D*

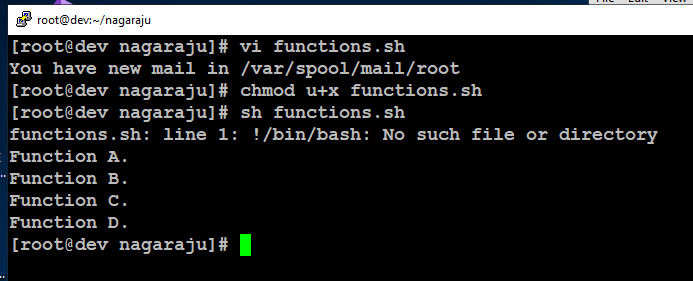


Figure 28: using Functions

### Bash Select

*#!/bin/bash*

*PS3='Choose one word: '*

*# bash select*

*select word in "linux" "bash" "scripting" "tutorial"*

*do*

*echo "The word you have selected is: $word"*

*# Break, otherwise endless loop*

*break*

*done*

*exit 0*

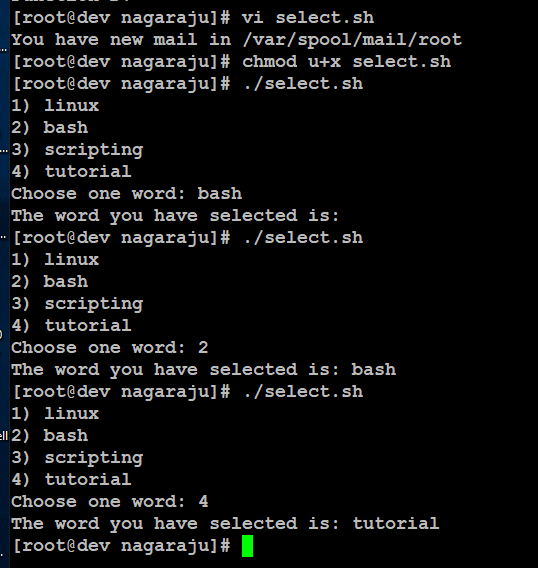


Figure 29: Select script

### Case statement conditional

*#!/bin/bash*

*echo "What is your preferred programming / scripting language"*

*echo "1) bash"*

*echo "2) perl"*

*echo "3) phyton"*

*echo "4) c++"*

*echo "5) I do not know !"*

*read case;*

*#simple case bash structure*

*# note in this case $case is variable and does not have to*

*# be named case this is just an example*

*case $case in*

*1) echo "You selected bash";;*

*2) echo "You selected perl";;*

*3) echo "You selected phyton";;*

*4) echo "You selected c++";;*

*5) exit*

*esac*

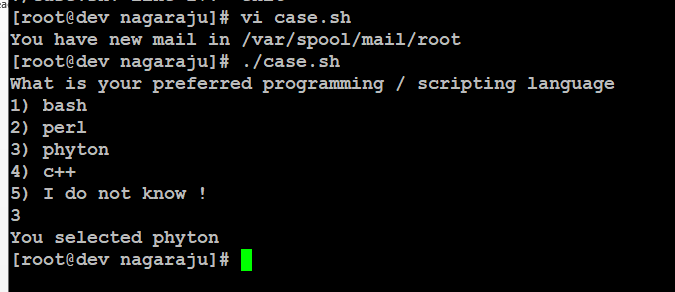


Figure 30: Case statement conditional script executed

### Bash quotes and quotations

*#!/bin/bash*

*#Declare bash string variable*

*BASH\_VAR="Bash Script"*

*# echo variable BASH\_VAR*

*echo $BASH\_VAR*

*#when meta character such us "$" is escaped with "\" it will be read literally*

*echo \$BASH\_VAR*

*# backslash has also special meaning and it can be suppressed with yet another "\"*

*echo "\\"*

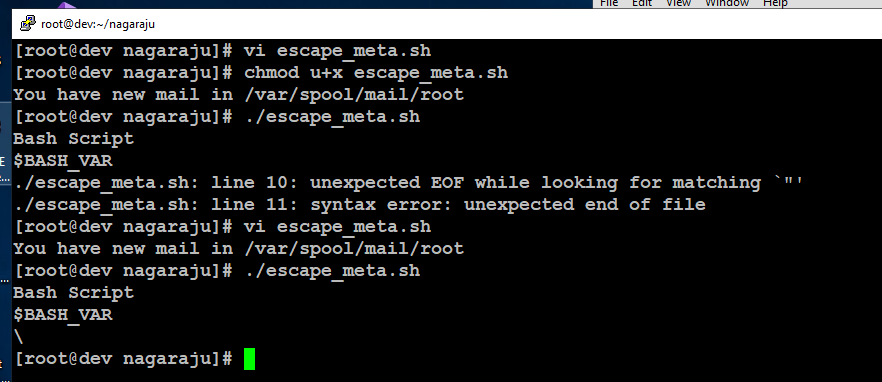


Figure 31: Bash quotes and quotations

#### Single quotes

*#!/bin/bash*

*# Declare bash string variable*

*BASH\_VAR="Bash Script"*

*# echo variable BASH\_VAR*

*echo $BASH\_VAR*

*# meta characters special meaning in bash is suppressed when using single quotes*

*echo '$BASH\_VAR "$BASH\_VAR"'*

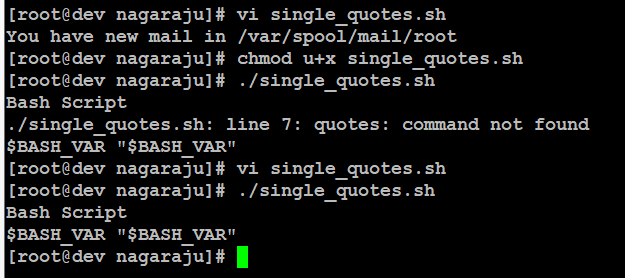


Figure 32: Single quotes

#### Double quotes

*#!/bin/bash*

*#Declare bash string variable*

*BASH\_VAR="Bash Script"*

*# echo variable BASH\_VAR*

*echo $BASH\_VAR*

*# meta characters and its special meaning in bash is*

*# suppressed when using double quotes except "$", "\" and "`"*

*echo "It's $BASH\_VAR and \"$BASH\_VAR\" using backticks: `date`"*

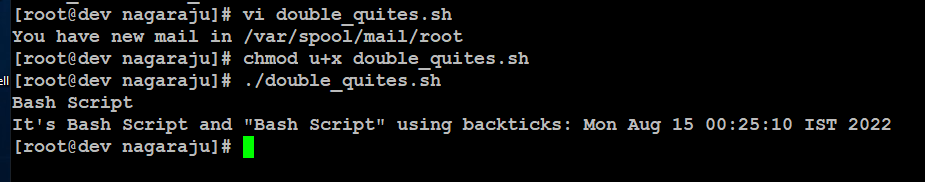


Figure 33: Double quotes

#### Bash quoting with ANSI-C style

*#!/bin/bash*

*# as a example we have used \n as a new line, \x40 is hex value for @*

*# and \56 is octal value for .*

*echo $'web: www.linuxconfig.org\nemail: web\x40linuxconfig\56org'*

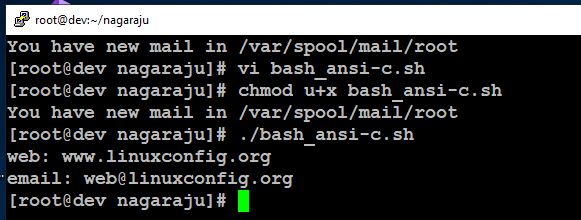


Figure 34: quoting with ANSI-C style

### Arithmetic Operations

Bash Addition Calculator Example

*#!/bin/bash*

*let RESULT1=$1+$2*

*echo $1+$2=$RESULT1 ' -> # let RESULT1=$1+$2'*

*declare -i RESULT2*

*RESULT2=$1+$2*

*echo $1+$2=$RESULT2 ' -> # declare -i RESULT2; RESULT2=$1+$2'*

*echo $1+$2=$(($1 + $2)) ' -> # $(($1 + $2))'*

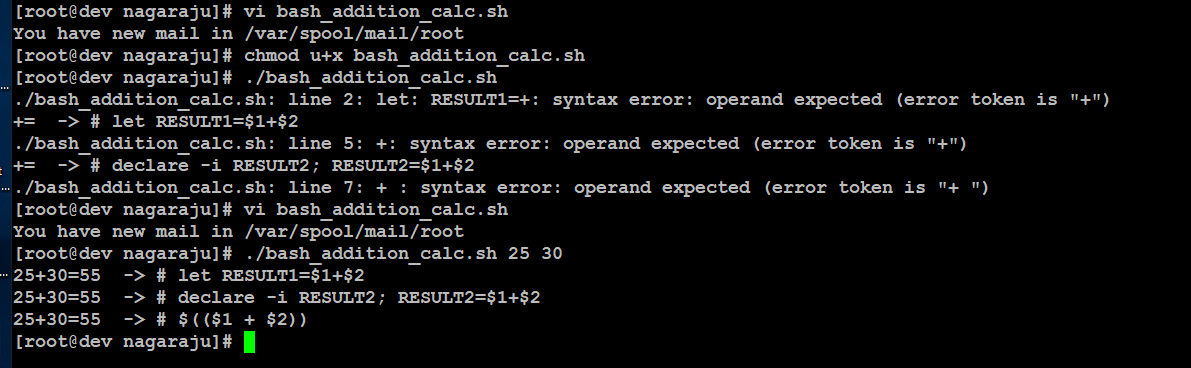


Figure 35: Addition Calculator Example

#### Bash Arithmetic’s

*#!/bin/bash*

*echo '### let ###'*

*# bash addition*

*let ADDITION=3+5*

*echo "3 + 5 =" $ADDITION*

*# bash subtraction*

*let SUBTRACTION=7-8*

*echo "7 - 8 =" $SUBTRACTION*

*# bash multiplication*

*let MULTIPLICATION=5\*8*

*echo "5 \* 8 =" $MULTIPLICATION*

*# bash division*

*let DIVISION=4/2*

*echo "4 / 2 =" $DIVISION*

*# bash modulus*

*let MODULUS=9%4*

*echo "9 % 4 =" $MODULUS*

*# bash power of two*

*let POWEROFTWO=2\*\*2*

*echo "2 ^ 2 =" $POWEROFTWO*

*echo '### Bash Arithmetic Expansion ###'*

*# There are two formats for arithmetic expansion: $[ expression ]*

*# and $(( expression #)) its your choice which you use*

*echo 4 + 5 = $((4 + 5))*

*echo 7 - 7 = $[ 7 - 7 ]*

*echo 4 x 6 = $((3 \* 2))*

*echo 6 / 3 = $((6 / 3))*

*echo 8 % 7 = $((8 % 7))*

*echo 2 ^ 8 = $[ 2 \*\* 8 ]*

*echo '### Declare ###'*

*echo -e "Please enter two numbers \c"*

*# read user input*

*read num1 num2*

*declare -i result*

*result=$num1+$num2*

*echo "Result is:$result "*

*# bash convert binary number 10001*

*result=2#10001*

*echo $result*

*# bash convert octal number 16*

*result=8#16*

*echo $result*

*# bash convert hex number 0xE6A*

*result=16#E6A*

*echo $result*

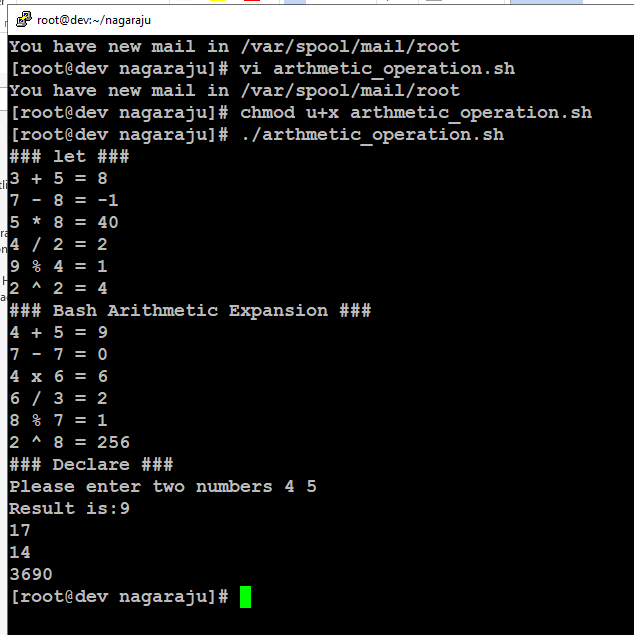


Figure 36: Arithmetic’s

#### Round floating point number

*#!/bin/bash*

*# get floating point number*

*floating\_point\_number=3.3446*

*echo $floating\_point\_number*

*# round floating point number with bash*

*for bash\_rounded\_number in $(printf %.0f $floating\_point\_number); do*

*echo "Rounded number with bash:" $bash\_rounded\_number*

*done*

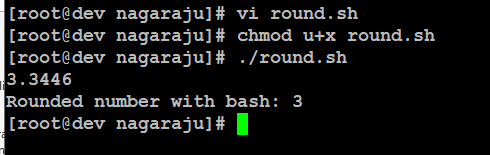


Figure 37: Round floating point number

#### Bash floating point calculations

*#!/bin/bash*

*# Simple linux bash calculator*

*echo "Enter input:"*

*read userinput*

*echo "Result with 2 digits after decimal point:"*

*echo "scale=2; ${userinput}" | bc*

*echo "Result with 10 digits after decimal point:"*

*echo "scale=10; ${userinput}" | bc*

*echo "Result as rounded integer:"*

*echo $userinput | bc*

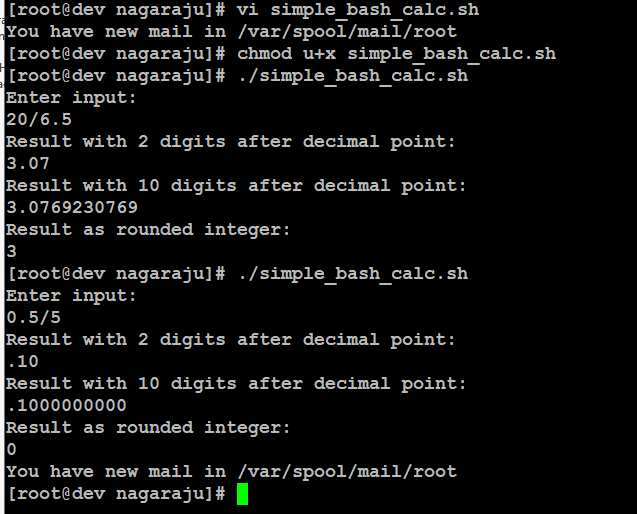


Figure 38: floating point calculations

### Redirections

#### STDOUT from bash script to STDERR

*#!/bin/bash*

*echo "Redirect this STDOUT to STDERR" 1>&2*

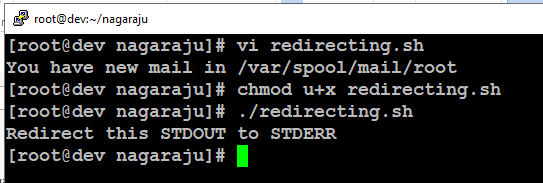


Figure 39: STDOUT from bash script to STDERR

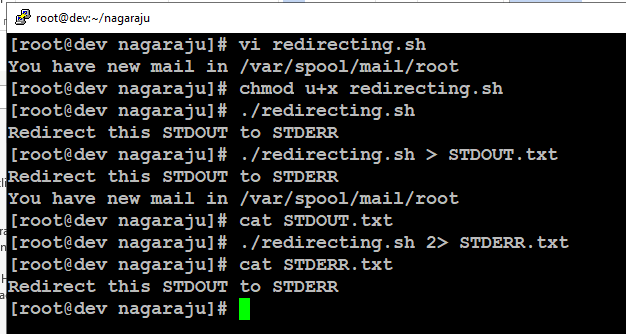


Figure 40: redirect script’s output to file

#### STDERR from bash script to STDOUT



Figure 41: STDERR from bash script to STDOUT

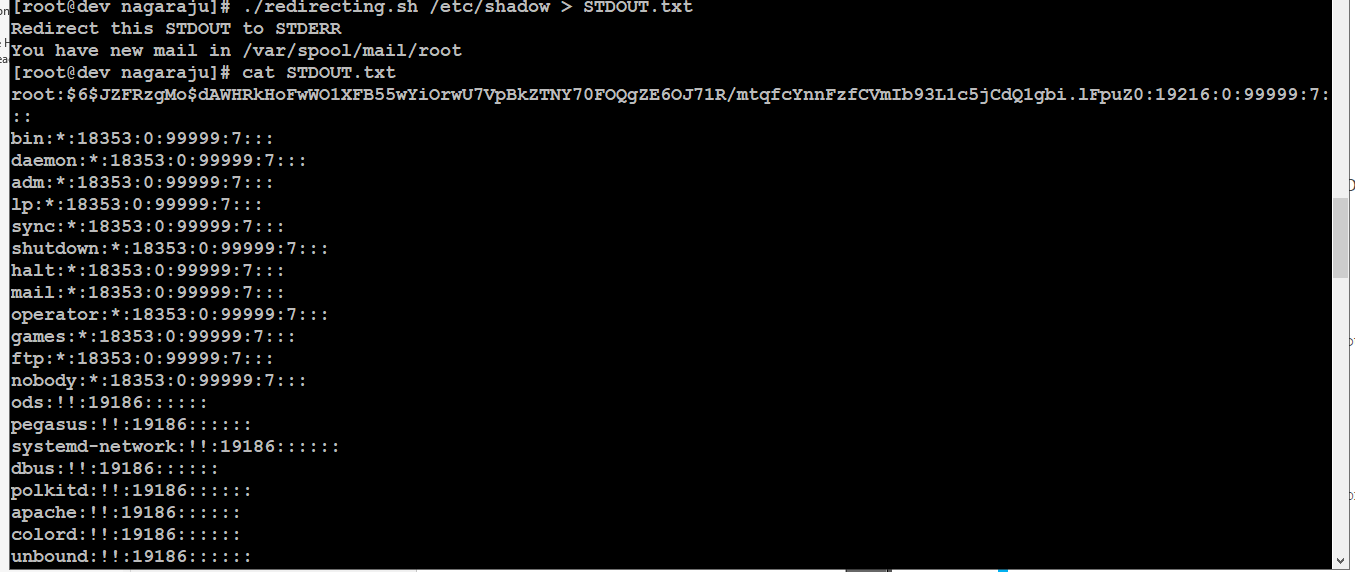
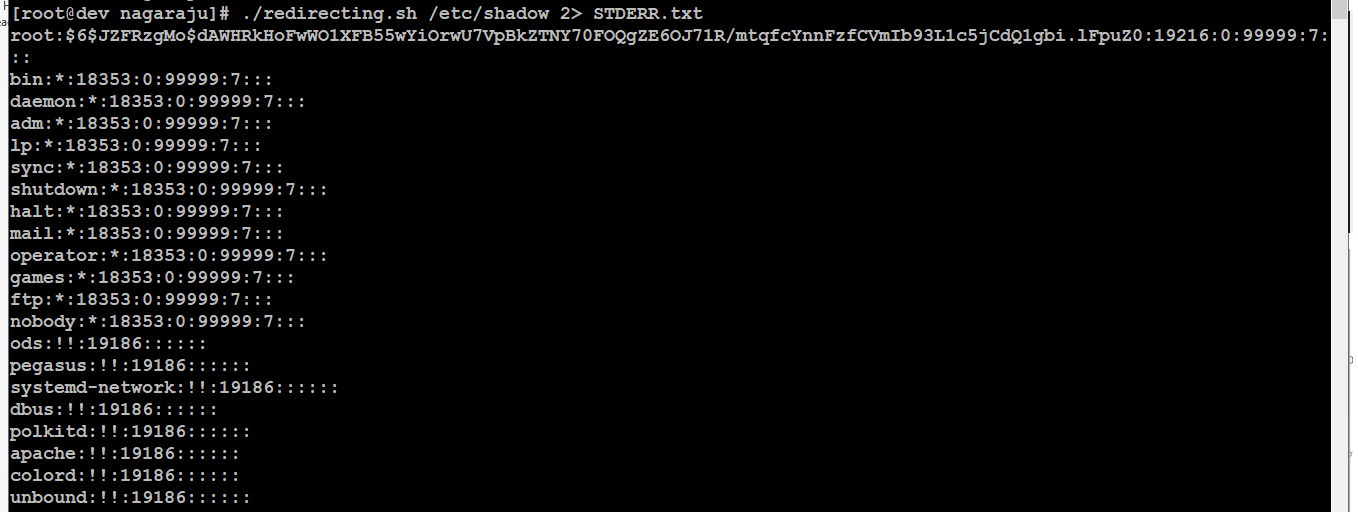
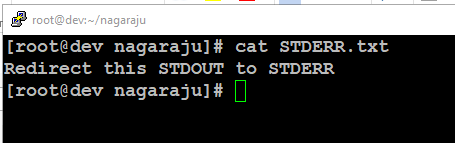


Figure 42: STDERR is redirected to STDOUT we can redirect script’s output to file





#### stdout to screen

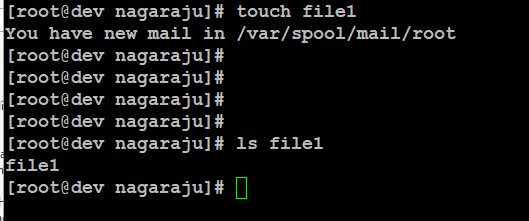


Figure 43: stdout to screen

#### stdout to file

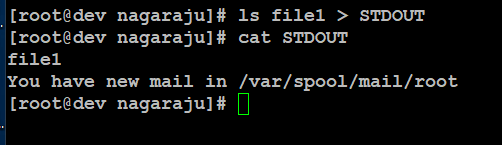


Figure 44: redirecting stdout to file

#### stderr to file

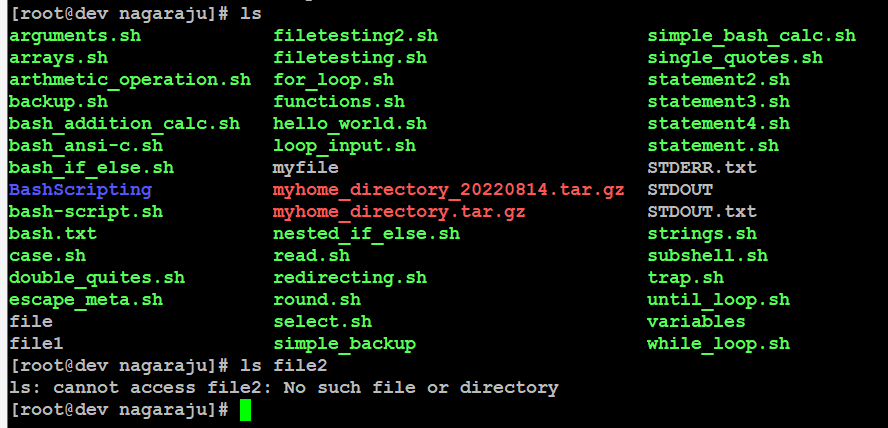


Figure 45: redirecting stderr to file

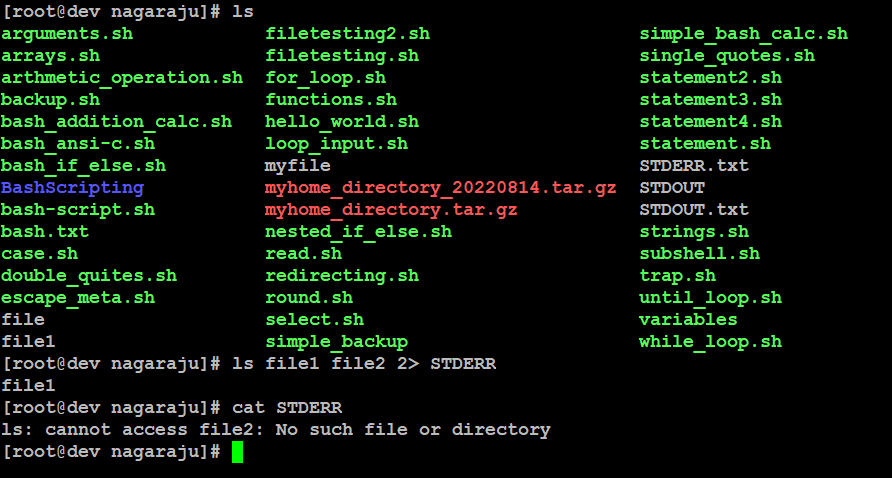


Figure 46redirecting STDERR and checking

#### stdout to stderr

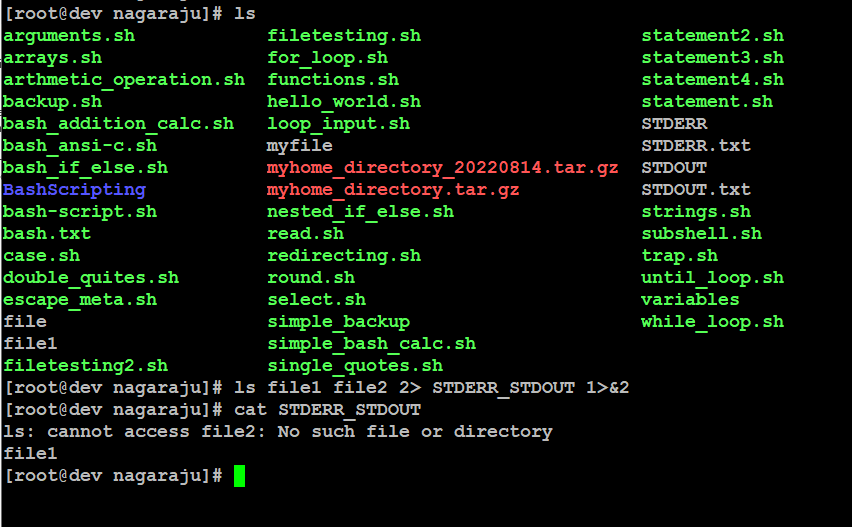


Figure 47: Redirecting stdout to stderr

#### stderr to stdout

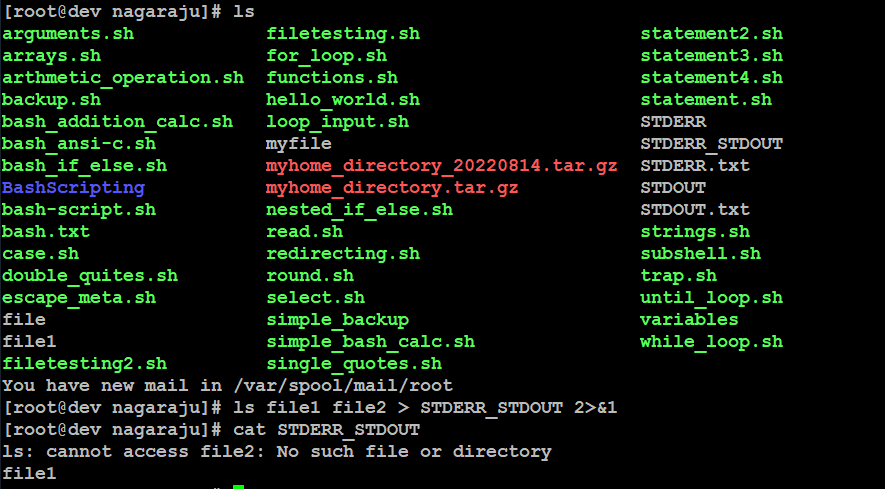


Figure 48: Redirectiong stdout to stderr

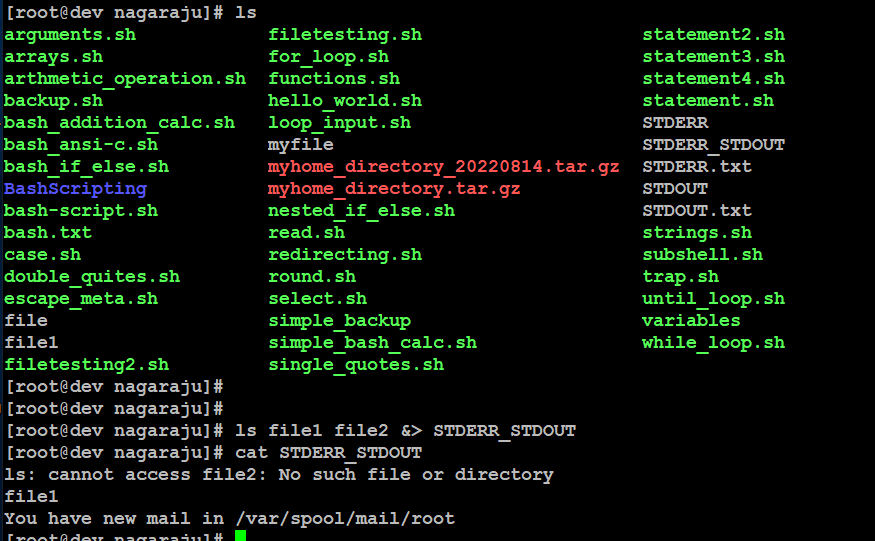


Figure 49: Another way to achieve the same effect is illustrated

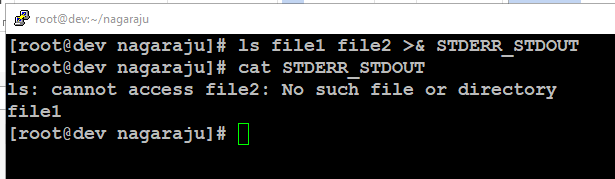


Figure 50: Checking redirected file