***Lab 04 – Exploring File Commands and Conditional Structures***

|  |
| --- |
| **Objectives** |
| * To learn the usage of Linux commands related to files. * To learn the usage of control structures in shell scripts. |

## File Management Commands

The commands provided in this section generally addresses the creation, modification and calling of files within the Linux environment. More details on purpose and usage of the given commands can be obtained through the ***man*** command.

|  |
| --- |
| **touch Command:**  The touch command is the easiest way to create new empty files. It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files. The timestamps of the file is changed without modifying or replacing the contents of that file. The type of the created file would depend on the provided extension for the file.  **Examples:**  $ touch file.txt // Creates a file with extension .txt  $ touch file.txt // Changes the timestamp of the file without changing its contents  $ touch file.sh // Creates a file with extension .sh |

|  |
| --- |
| **The > and >> operators:**  You can redirect the standard output channel using the shell operator “>”, i.e. (the “greater-than” sign). In the following example, the output of “ls -la” is redirected to a file called "filelist" and not on the screen. If the “filelist” file does not exist, then it is created. Should a file by that name exist, then its content will be overwritten. If you want to append a command’s output to an existing file without replacing its previous content, use the “>>” operator instead. If that file does not exist, it will be created in this case as well.  **Examples:**  $ ls -la > filelist.txt  $ echo “The End of File” >> filelist.txt |

|  |
| --- |
| **The < operator:**  You can use “<” (the “less-than” sign) to redirect the standard input channel. This will read the content of the specified file instead of keyboard input. There is no “<<” redirection operator to concatenate multiple input files.  **Examples:**  $ wc -w < document.txt // The wc filter command counts the words in file document.txt |

|  |
| --- |
| **Using the redirecting operators together:**  The standard input and output may be redirected at the same. In the example below, the word counts output is written to a file called “wordcount.txt”, where -w option is used for obtaining a word count of the input file “document.txt”. A character count can be obtained using the -m option instead of -w.  **Examples:**  $ wc -w < document.txt > wordcount.txt |

|  |
| --- |
| **source Command:**  The source command reads a file containing shell commands as if they had been entered on the command line directly.  **Examples:**  $ source file.txt |

|  |
| --- |
| **cat command:**  The cat (“concatenate”) command is intended to join several files named on the command line into one. If you pass just a single file name, the content of that file will be written to standard output. If you do not pass a file name at all, cat reads its standard input. The following examples provide several usage of the cat command.  **Examples:**  $ cat file1.txt > file2.txt // Copies the content of file1.txt into file2.txt  $ cat file1.txt file2.txt file3.txt > file4.txt // Concatenates in sequence the contents of file1.txt, // file2.txt and file3.txt into file4.txt  $ cat file1.txt file2.txt file3.txt // Concatenates in sequence the contents of file1.txt, // file2.txt and file3.txt and displays it on screen |

|  |
| --- |
| **read command:**  To get input from the keyboard, you use the read command. The read command takes input from the keyboard and assigns it to a variable. The example below provides a demonstration on how to use this command.  $ read variable // Takes a value as an input from the user and stores // it in “variable”. Later, you can use $variable to use // the stored value. E.g. $echo $variable |

## Basic arithmetic in Shell Scripts

Arithmetic expansion allows the evaluation of an arithmetic expression and the substitution of the result. The format for arithmetic expansion is:

$(( EXPRESSION ))

The expression is treated as if it were within double quotes, but a double quote inside the parentheses is not treated specially. All tokens in the expression undergo parameter expansion, command substitution, and quote removal. Arithmetic substitutions may be nested as well. It is natural to expect that shell scripts can perform some simple arithmetic. The shell provides some features for integer arithmetic. If you must deal with fractional numbers, there is a separate program called “bc” which provides an arbitrary precision calculator language that can be used in shell scripts, but is beyond the scope of this manual. When you surround an arithmetic expression with the double parentheses, the shell will perform arithmetic evaluation, as shown in the example below:

|  |
| --- |
| $ echo $(( 2+2 )) // Displays the result 4 |

In the example below, notice how the leading "$" is not needed to reference variables inside the arithmetic expression such as "first\_num + second\_num".

|  |
| --- |
| #!/bin/bash  first\_num=0  second\_num=0  echo "Enter the first number: "  read first\_num  echo "Enter the second number: "  read second\_num  echo "first number + second number = $((first\_num + second\_num))" // Summation  echo "first number - second number = $((first\_num - second\_num))" // Subtraction  echo "first number \* second number = $((first\_num \* second\_num))" // Multiplication  echo "first number / second number = $((first\_num / second\_num))" // Division  echo "first number % second number = $((first\_num % second\_num))" // Remainder |

The different operators that can be used within an arithmetic expression are roughly the same as in the C programming language. Some of the most significant ones are as given below:

|  |  |
| --- | --- |
| Operator | Meaning |
| <=, >=, < and > | relational operators |
| == and != | equality and inequality |
| && | logical AND |
| || | logical OR |

## Conditional Structures in Shell Scripts

The ***if*** condition is used for decision making in shell script, where if the given condition is true then the command is executed. The general forms for the ***if*** condition that can be used in shell scripting is given below.

|  |
| --- |
| # First form  if [condition]; then  commands  fi  # Second form  if [condition]; then  commands  else  commands  fi  # Third form  if [condition]; then  commands  elif [condition]; then  commands  fi |

Unix Shell supports following forms of ***if*** else statement:

if...fi statement

if...else...fi statement

if...elif...else...fi statement

Examples for using the ***if*** control structure are given below:

|  |
| --- |
| #!/bin/bash  number=1  if [ $number == 1 ]; then  echo "Number equals 1"  else  echo "Number does not equal 1"  fi |

|  |
| --- |
| #!/bin/bash  number=0  echo "Enter a number: "  read number  echo "The number is $number"  if [ $((number % 2)) == 0 ]; then  echo "Number is even"  else  echo "Number is odd"  fi |

The example below shows how to compare string values, where the compared string values are placed between double quotes and a single equal sign is used for comparison.

|  |
| --- |
| #!/bin/bash  echo -n "Enter a number between 1 and 3 inclusive > "  read character  if [ "$character" = "1" ]; then  echo "You entered one."  elif [ "$character" = "2" ]; then  echo "You entered two."  elif [ "$character" = "3" ]; then  echo "You entered three."  else  echo "You did not enter a number between 1 and 3."  fi |

## Arrays in C Language

## Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

To declare a single-dimensional array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

type arrayName [ arraySize ];

The following example shows how to use the concepts of arryas declaration, assignment, and accessing arrays:

|  |
| --- |
| #include <stdio.h>  int main ()  {  int n[ 10 ]; /\* n is an array of 10 integers \*/  int i,j;  /\* initialize elements of array n to 0 \*/  for ( i = 0; i < 10; i++ )  {  n[ i ] = i + 100; /\* set element at location i to i + 100 \*/  }  /\* output each array element's value \*/  for ( j = 0; j < 10; j++ )  {  printf( "Element[%d] = %d\n", j, n[j] );  }    return 0;  } |

The C programming language supports the concept of multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array. Here is the general form of a multidimensional array declaration:

type name[size1][size2]...[sizeN];

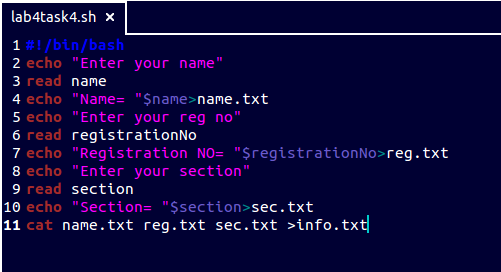
The following program uses a nested loop to handle a two-dimensional array:

|  |
| --- |
| #include <stdio.h>  int main ()  {  /\* an array with 5 rows and 2 columns\*/  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8} };  int i, j;  /\* output each array element's value \*/  for ( i = 0; i < 5; i++ )  {  for ( j = 0; j < 2; j++ )  {  printf( "a[%d][%d] = %d\n", i,j, a[i][j] );  }  }  return 0;  } |

## Lab Exercise

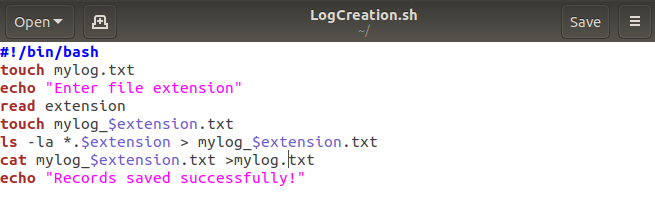
**Task1: Write a single shell script that creates four different files, while taking the names of all created files as input from the user. As the files contents, insert your name in the first file, registration number in the second and section details in the third. These should be followed by merging the contents of all three files into the fourth one.**

**Code:**

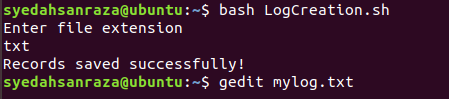


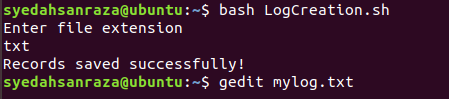
**Task2: Write a shell script that creates a “Files Location Log”. The paths of all files, having the same extension, should be stored in one log. The file extension should be taken as an input from the user, and the created logs should be named as “mylog\_extension.txt”, where “extension” is that taken as input from the user. The search process should be for all file in the system, starting from the root directory (/). All log files of different file extension should be stored inside a single directory by the name of “mylog” that would be present at your home directory.**

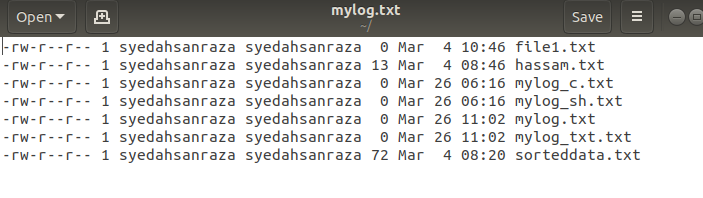
**Code:**



**Output:**

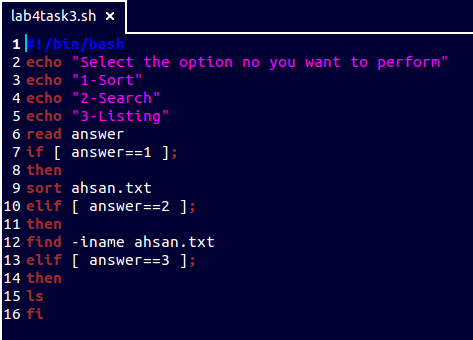




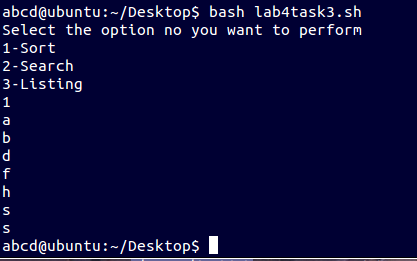


**Task3: Write a shell script that either performs a file sort, file search or directory listing operation based on the user’s selection of the operation he/she would like to execute.**

**Code:**



**Code**:



**Task4: Write a C program that takes values of two matrices of size (𝑚×1) and (1×𝑛) as input from the user. Multiply the above two matrixes and store the resulting (𝑚×𝑛) matrix in a 2D array. Display the contents of the first and second matrices and also the resulting matrix. Achieve alignment in the displayed content as much possible.**

**Code:**





**Output:**

