Manual and Assignment for Operating System Lab (CS693)

Lab # 3:

Objectives

Essentials of shell programming

1. **vi editor:** There are many ways to edit files in Unix. Editing files using the screen-oriented text editor **vi** is one of the best ways. This editor enables you to edit lines in context with other lines in the file.

   The following table lists out the basic commands to use the vi editor −

S.No.	Command & Description
1	**vi filename** Creates a new file if it already does not exist, otherwise opens an existing file.
2	**vi -R filename** Opens an existing file in the read-only mode.
3	**view filename** Opens an existing file in the read-only mode.

On opening a file you will notice a **tilde** (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

Operation Modes

While working with the vi editor, we usually come across the following two modes −

- **Command mode** − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

Moving within a File

S.No.	Command & Description
1	**k** Moves the cursor up one line
2	**j** Moves the cursor down one line
3	**h** Moves the cursor to the left one character position
4	**l** Moves the cursor to the right one character position

Editing Files

S.No.	Command & Description
1	**i** Inserts text before the current cursor location
2	**I** Inserts text at the beginning of the current line
3	**a** Inserts text after the current cursor location
4	**A** Inserts text at the end of the current line
5	**o** Creates a new line for text entry below the cursor location
6	**O** Creates a new line for text entry above the cursor location

Deleting Characters

S.No.	Command & Description
1	**x** Deletes the character under the cursor location
2	**X** Deletes the character before the cursor location
3	**dw** Deletes from the current cursor location to the next word
4	**d^** Deletes from the current cursor position to the beginning of the line
5	**d$** Deletes from the current cursor position to the end of the line
6	**D** Deletes from the cursor position to the end of the current line
7	**dd** Deletes the line the cursor is on

2. **Shell basics:** A UNIX shell script is a human-readable text file containing a group of commands that could also be manually executed one-by-one at the UNIX operating system command prompt. They are often kept in a file (script) because they are too numerous to type in at the command prompt each time you want to perform a specific task, or because together they form a complex computer program.

   The vi program, or any one of the many UNIX text editors out there, can be used to create a shell script and save it to disk for subsequent execution. Unlike other high-level programming languages, such as C or C++, shell scripts do not need to be compiled into a binary format for execution. Because of this, programming syntax errors are not revealed until the script is executed.

   Steps in creating a Shell Script

Create a file using a vi editor(or any other editor). Name script file with extension .sh

Start the script with #! /bin/sh

Write some code.

Save the script file as filename.sh

For executing the script type bash filename.sh

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.

Let create a small script -

#!/bin/sh

ls

To create the script:

1.$vi demo.sh
2. #!/bin/sh
ls
~
3.bash demo.sh

3. **Command line arguments:** The command-line arguments $1, $2, $3, ...$9 are positional parameters, with $0 pointing to the actual command, program, shell script, or function and $1, $2, $3, ...$9 as the arguments to the command.

$0: The filename of the current script.

$#: The number of arguments supplied to a script.

$*: All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.

$@: All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.

Example1: demo.sh

```
#!/bin/sh

echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"

execute: $sh demo.sh darothi sarkar
```

```
First Parameter : darothi
Second Parameter : sarkar
Quoted Values: darothi sarkar
Quoted Values: darothi sarkar
Total Number of Parameters : 2
```

Example2: demo1.sh

```
#!/bin/bash

 # Call this script with at least 3 parameters, for example
 # sh scriptname 1 2 3
 echo "first parameter is $1"
 echo "Second parameter is $2"
 echo "Third parameter is $3"
 exit 0

execute: $sh demo1.sh 10 20 30

first parameter is 10

Second parameter is 20
Third parameter is 30
```

4. **Shell variables:** A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

   For example, first we set a variable TEST and then we access its value using the **echo** command −

   ```
   $TEST="Unix Programming"
   $echo $TEST
   ```

   It produces the following result.

   ```
   Unix Programming
   ```

   The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

   By convention, Unix shell variables will have their names in UPPERCASE.

   The following examples are valid variable names −

   ```
   _ALI

   TOKEN_A
   VAR_1
   VAR_2
   ```

   Following are the examples of invalid variable names −

   ```
   2_VAR
   -VARIABLE
   VAR1-VAR2
   ```

```
VAR_A!
```

The reason you cannot use other characters such as **!**, *****, or **-** is that these characters have a special meaning for the shell.

5. **If else block:**

Syntax

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

Example

```
a=10

b=20

if [ $a == $b ]
then
   echo "a is equal to b"
elif [ $a -gt $b ]
then
   echo "a is greater than b"
elif [ $a -lt $b ]
then
   echo "a is less than b"
else
   echo "None of the condition met"
fi
```

Assignment

1. Write a shell program to display the content of a file after reading the file.

2. Write a shell program to display the first three lines of a file.

3. Write a shell program to perform the swapping between two numbers taken from user during run time.

4. Write a shell program to print the largest among three numbers by passing the numbers through command line arguments.

5. Write a shell program to display the following mark sheets of students by taking the input marks of student through the terminal

Marks range	Grade
90>=M<=100	A
70>=M<=89	B
40>=M<=69	C
M<40	F