|  |  |
| --- | --- |
| **SARDAR PATEL COLLEGE OF ENGINEERING, BAKROL , ANAND**  **LAB MANUAL** | |
| **Subject Name : Operating System& Virtualization** | **Semester: 4th** |
| **Subject Code: 3141601** | **Branch: IT** |

**INDEX**

|  |  |
| --- | --- |
| **Sr. No.** | **Title** |
| 1 | Study of Basic commands of Linux/UNIX. |
| 2 | Study of Advance commands and filters of Linux/UNIX. |
| 3 | Write a shell script to generate marksheet of a student. Take 3 subjects, calculate and display total marks, percentage and Class obtained by the student. |
| 4 | Write a shell script to display multiplication table of given number . |
| 5 | Write a shell script to find factorial of given number n. |
| 6 | Write a shell script which will accept a number b and display first n prime numbers as output. |
| 7 | Write a shell script which will generate first n fibonnacci numbers like: 1, 1, 2, 3, 5, 13, … |
| 8 | Write a menu driven shell script which will print the following menu and execute the given task.  a. Display calendar of current month  b. Display today’s date and time  c. Display usernames those are currently logged in the system  d. Display your name at given x, y position  e. Display your terminal number |
| 9 | Write a shell script to read n numbers as command arguments and sort them in descending order. |
| 10 | Write a shell script to display all executable files, directories and zero sized files from current directory. |
| 11 | Write a shell script to check entered string is palindrome or not. |
| 12 | Study of Unix Shell and Environment Variables. |
| 13 | Write a shell script to validate the entered date. (eg. Date format is :dd-mm-yyyy). |
| 14 | Write a program for process creation using C. (Use of gcc compiler). |

**Practical 1**

**Aim: Study of Basic commands of Linux/UNIX.**

**Files and Directories:**

These commands allow you to create directories and handle files.

**Command** **Description**

cat Display File Contents

cd Changes Directory to dirname

chgrp change file group

chmod Changing Permissions

cp Copy source file into destination

file Determine file type

find Find files

grep Search files for regular expressions.

head Display first few lines of a file

ln Create softlink on oldname

ls Display information about file type.

mkdir Create a new directory dirname

more Display data in paginated form.

mv Move (Rename) a oldname to newname.

pwd Print current working directory.

rm Remove (Delete) filename

rmdir Delete an existing directory provided it is empty.

tail Prints last few lines in a file.

touch Update access and modification time of a file.

**Manipulating data:**

The contents of files can be compared and altered with the following commands.

**Command** **Description**

awk Pattern scanning and processing language

cmp Compare the contents of two files

comm Compare sorted data

cut Cut out selected fields of each line of a file

diff Differential file comparator

expand Expand tabs to spaces

join Join files on some common field

perl Data manipulation language

sed Stream text edito

sort Sort file data

split Split file into smaller files

tr Translate characters

uniq Report repeated lines in a file

wc Count words, lines, and characters

vi Opens vi text editor

vim Opens vim text editor

fmt Simple text formatter

spell Check text for spelling error

ispell Check text for spelling error

ispell Check text for spelling error

emacs GNU project Emacs

ex, edit Line editor

emacs GNU project Emacs

emacs GNU project Emacs

**Compressed Files:**

Files may be compressed to save space. Compressed files can be created and examined:

**Command** **Description**

compress Compress files

gunzip Uncompressgzipped files

gzip GNU alternative compression method

uncompressUncompress files

unzip List, test and extract compressed files in a ZIP archive

zcat Cat a compressed file

zcmp Compare compressed files

zdiff Compare compressed files

zmore File perusal filter for crt viewing of compressed text

**Getting Information:**

Various Unix manuals and documentation are available on-line. The following Shell commands give information:

**Command** **Description**

apropos Locate commands by keyword lookup

info Displays command information pages online

man Displays manual pages online

whatis Search the whatis database for complete words.

yelp GNOME help viewer

**Network Communication:**

These following commands are used to send and receive files from a local UNIX hosts to the remote host around the world.

**Command** **Description**

ftp File transfer program

rcp Remote file copy

rlogin Remote login to a UNIX host

rsh Remote shell

tftp Trivial file transfer program

telnet Make terminal connection to another host

ssh Secure shell terminal or command connection

scp Secure shell remote file copy

sftp secure shell file transfer program

Some of these commands may be restricted at your computer for security reasons.

**Messages between Users:**

The UNIX systems support on-screen messages to other users and world-wide electronic mail:

**Command** **Description**

evolution GUI mail handling tool on Linux

mail Simple send or read mail program

mesg Permit or deny messages

parcel Send files to another user

pine Vdu-based mail utility

talk Talk to another user

write Write message to another user

**Programming Utilities:**

The following programming tools and languages are available based on what you have installed on your Unix.

**Command** **Description**

dbx Sun debugger

gdb GNU debugger

make Maintain program groups and compile programs.

nm Print program's name list

size Print program's sizes

strip Remove symbol table and relocation bits

cb C program beautifier

cc ANSI C compiler for Suns SPARC systems

**Practical 2**

**Aim: Study of Advance commands and filters of Linux/UNIX.**

**The grep Command:**

The grep program searches a file or files for lines that have a certain pattern. The syntax is:

$grep pattern file(s)

The name "grep" derives from the ed (a UNIX line editor) command g/re/p which means "globally search for a regular expression and print all lines containing it."

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| $ls -l | grep "Aug" | | |  |  |  |  |  |
| -rw-rw-rw- | 1 | john | doc | 11008 | Aug | 6 | 14:10 ch02 |
| -rw-rw-rw- | 1 | john | doc | 8515 | Aug | 6 | 15:30 ch07 |
| -rw-rw-r-- | 1 | john | doc | 2488 | Aug | 15 | 10:51 intro |
| -rw-rw-r-- | 1 | carol doc | | 1605 | Aug | 23 | 07:35 macros |
| $ |  |  |  |  |  |  |  |

There are various options which you can use along with grep command:

**Option -v**

**-n**

**-l**

**Description**

Print all lines that do not match pattern. Print the matched line and its line number.

Print only the names of files with matching lines (letter "l")

**-c** Print only the count of matching lines. **-i**Match either upper- or lowercase.

Next, let's use a regular expression that tells grep to find lines with "carol", followed by zero or more other characters abbreviated in a regular expression as ".\*"), then followed by "Aug".

Here we are using *-i* option to have case insensitive search:

$ls -l | grep -i "carol.\*aug"

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

**The sort Command:**

The **sort** command arranges lines of text alphabetically or numerically. The example below sorts the lines in the food file:

$sort food Afghani Cuisine Bangkok Wok Big Apple Deli Isle of Java Mandalay

Sushi and Sashimi Sweet Tooth

Tio Pepe's Peppers

$

The **sort** command arranges lines of text alphabetically by default. There are many options that control the sorting:

|  |  |  |
| --- | --- | --- |
| **Option** | **Description** |  |
| **-n** | Sort numerically (example: 10 will sort after 2), ignore blanks and |  |
| tabs. |  |
|  |  |

**-r** Reverse the order of sort.

**-f**Sort upper- and lowercase together. **+x** Ignore first x fields when sorting.

More than two commands may be linked up into a pipe. Taking a previous pipe example using **grep**, we can further sort the files modified in August by order of size.

The following pipe consists of the commands **ls, grep,** and **sort**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $ls -l | grep | "Aug" | | sort +4n | |  |  |  |
| -rw-rw-r-- 1 | carol | doc | 1605 | Aug | 23 | 07:35 macros |
| -rw-rw-r-- 1 | John | doc | 2488 | Aug | 15 | 10:51 intro |
| -rw-rw-rw- 1 | John | doc | 8515 | Aug | 6 | 15:30 ch07 |
| -rw-rw-rw- 1 | John | doc | 11008 | Aug | 6 | 14:10 ch02 |
| $ |  |  |  |  |  |  |

This pipe sorts all files in your directory modified in August by order of size, and prints them to the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

**The pg and more Commands:**

A long output would normally zip by you on the screen, but if you run text through more or pg as a filter, the display stops after each screenful of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through **more** as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $ls -l | grep | "Aug" | | sort +4n | more | | |  |  |
| -rw-rw-r-- 1 | carol | doc | 1605 | Aug | 23 | 07:35 macros |
| -rw-rw-r-- 1 | John | doc | 2488 | Aug | 15 | 10:51 intro |
| -rw-rw-rw- 1 | John | doc | 8515 | Aug | 6 | 15:30 ch07 |
| -rw-rw-r-- 1 | John | doc | 14827 | Aug | 9 | 12:40 ch03 |
| . |  |  |  |  |  |  |
| . |  |  |  |  |  |  |
| . |  |  |  |  |  |  |
| -rw-rw-rw- 1 | John | doc | 16867 | Aug | 6 | 15:56 ch05 |
| --More--(74%) |  |  |  |  |  |  |

The screen will fill up with one screenful of text consisting of lines sorted by order of file size. At the bottom of the screen is the **more** prompt where you can type a command to move through the sorted text

**Practical 3**

**Aim: Write a shell script to generate mark sheet of a student. Take 3 subjects, calculate and display total marks, percentage and Class obtained by the student.**

echo "Enter marks subject1:"

read s1

echo "Enter marks subject2:"

read s2

echo "Enter marks subject3:"

read s3

sum=`expr $s1 + $s2 + $s3`

echo "sum is:" $sum

avg=`expr $sum / 3`

echo "Avgrage is:" $avg

per=`expr $sum / 3`

echo "Percantage is:" $per

if [ $per -ge 70 ]

then

echo "...DISCTIONTION..."

elif [ $per -ge 60 ]

then

echo "... First CLASS..."

elif [ $per -ge 50 ]

then

echo "... Second CLASS..."

elif [ $per -ge 40 ]

then

echo "... PASS CLASS..."

else [ $per -ge 30 ]

echo "...You are FAIL..."

fi

**Output:**

Enter marks subject1:

55

Enter marks subject1:

60

Enter marks subject1:

65

Sum is: 180

Average is: 60

Percentage is: 60

... First CLASS...

**Practical 4**

**Aim: Write a shell script to display multiplication table of given number.**

echo "Enter a Number"

read n

i=0

while [ $i -le 10 ]

do

    echo " $n x $i = `expr $n \\* $i`"

    i=`expr $i + 1`

done

**Output**

Enter a Number : 2

2 \* 1 = 2

2 \* 2 = 4

2 \* 3 = 6

2 \* 4 = 8

2 \* 5 = 10

2 \* 6 = 12

2 \* 7 = 14

2 \* 8 = 16

2 \* 9 = 18

2 \* 10 = 20

**Practical 5**

**Aim: Write a shell script to find factorial of given number n.**

clear

i=1

fact=1

echo "enter the n value:"

read n

while [ $i -ne $n ]

do

i=`expr $i + 1`

fact=`expr $fact \\* $i`

done

echo "The factorial of $n is:"$fact

**Output:**

enter the n value:

4

The factorial of 4 is: 24

**Practical 6**

**Aim: Write a shell script which will accept a number b and display first n prime numbers as output.**

#!/bin/bash

prime\_1=0

echo "enter the range"

read n

echo " Primenumber between 1 to $n is:"

echo "1"

echo "2"

for((i=3;i<=n;))

do

for((j=i-1;j>=2;))

do

if [ `expr $i % $j` -ne 0 ] ; then

prime\_1=1

else

prime\_1=0

break

fi

j=`expr $j - 1`

done

if [ $prime\_1 -eq 1 ] ; then

echo $i

fi

i=`expr $i + 1`

done

**Output** :

Enter the range 10

Primenumber between 1 to 10 is :

2

3

5

7

**Practical 7**

**Aim: Write a shell script which will generate first n Fibonacci numbers like: 1, 1, 2, 3, 5, 13**

echo "Enter limit";

read n;

a=0;

b=1;

echo "Fibonacci Series upto $n:";

echo "$a";

echo "$b";

for((i=2;i<$n;i++))

do

c=`expr $a + $b`;

echo "$c";

a=$b;

b=$c;

done

**Output:**

Enter Limit:

6

Fibonacci Series upto 6:

0

1

1

2

3

5

**Practical 8**

**Aim:** Write a menu driven shell script which will print the following menu and execute the given task.

a. Display calendar of current month

b. Display today’s date and time

c. Display usernames those are currently logged in the system

d. Display your name at given x, y position

e. Display your terminal number

Echo “**MENU**

a.    . Display calendar of current month

b.    . Display today’s date and time

c.    . Display usernames those are currently logged in the system

d.    . Display your name at given x, y position

e.    . Display your terminal number

f.     . Exit

“

Read i

Case “$i” in

1)      Cal ;;

2)         ;;

3)      ;;

4)      Tput cup 10 10

Echo Jalpa ;;

5)      Pwd ;;

6)      Exit ;;

\*) echo “enter valid in put” ;;

esac

**Practical 9**

**Aim: Write a shell script to read n numbers as command arguments and sort them in descending order.**

#!/bin/bash

echo "enter maximum number"

read n

# taking input from user

echo "enter  Numbers in array:"

for (( i = 0; i< $n; i++ ))

do

readnos[$i]

done

#printing the number before sorting

echo "  Numbers in an array are:"

for (( i = 0; i< $n; i++ ))

do

echo ${nos[$i]}

done

# Now do the Sorting of numbers

for (( i = 0; i< $n ; i++ ))

do

for (( j = $i; j < $n; j++ ))

do

if [ ${nos[$i]} -lt ${nos[$j]}  ]; then

t=${nos[$i]}

nos[$i]=${nos[$j]}

nos[$j]=$t

fi

done

done

# Printing the sorted number in descending order

echo -e "\nSorted Numbers "

for (( i=0; i< $n; i++ ))

do

echo ${nos[$i]}

done

**Output :**

Enter maximum number 5

Enter Numbers in array :

10 3 2 45 8

Numbers in array are :

10

3

2

45

8

Sorted Numbers

45

10

8

3

2

**Practical 10**

**Aim: Write a shell script to display all executable files, directories and zero sized files from current directory.**

find $dir -size 0

DU ---- for dir

**Practical 11**

**Aim: Write a shell script to check entered string is palindrome or not.**

clear

echo "Enter a string to be entered:"

readstr

echo

len=`echo $str | wc -c`

len=`expr $len - 1`

i=1

j=`expr $len / 2`

while test $i -le $j

do

k=`echo $str | cut -c $i`

l=`echo $str | cut -c $len`

if test $k != $l

then

echo "String is not palindrome"

exit

fi

i=`expr $i + 1`

len=`expr $len - 1`

done

echo "String is palindrome"

**Output:**

Enter a string to be entered:

San

String is not palindrome

**Practical 12**

**Aim : Study of Unix Shell and Environment Variables.**

An **environment variable** is a setting normally inherited or declared when a shell is started. You can use shells to set variables; the syntax varies but Bourne shells use:

**$ *VARNAME*=*"new value"***

**$ export *VARNAME****or*

**$ export *VARNAME*=*"new value"***

Each program started from that shell will have *VARNAME* set to *newvalue*. The names of environment variables are case-sensitive; byconvention they are uppercase.

A **shell variable** is like an environment variable, except that it is not exported to new programs started from that shell. (You could export it, but normally you just write a shell initialisation script to set it in each shell.)

The shell sets up some default shell variables; PS2 is one of them. Other useful shell variables that are set or used in the Korn shell are:

* \_ **(underscore)** -- When an external command is executed by the shell, this is set in the environment of the new process to the path of the executed command. In interactive use, this parameter is also set in the parent shell to the last word of the previous command.
* COLUMNS -- The number of columns on the terminal or window.
* ENV -- If this parameter is found to be set after any profile files are executed, the expanded value is used as a shell startup file. It typically contains function and alias definitions.
* ERRNO -- Integer value of the shell's errno variable -- this indicates the reason the last system call failed.
* HISTFILE -- The name of the file used to store history. When assigned, history is loaded from the specified file. Multiple invocations of a shell running on the same machine will share history if their HISTFILE parameters all point to the same file. If HISTFILE isn't set, the default history file is $HOME/.sh\_history.
* HISTSIZE -- The number of commands normally stored in the history file. Default value is 128.
* IFS -- Internal field separator, used during substitution and by the read command to split values into distinct arguments; normally set to space, tab, and newline.
* LINENO -- The line number of the function or shell script that is being executed. This variable is useful for debugging shell scripts. Just add an echo $LINENO at various points and you should be able to determine your location within a script.
* LINES -- Set to the number of lines on the terminal or window.
* PPID -- The process ID of the shell's parent. A read-only variable.

* PATH -- A colon-separated list of directories that are searched when seeking commands.
* PS1 -- The primary prompt for interactive shells.
* PS2 -- Secondary prompt string; default value is >. Used when more input is needed to complete a command.
* PWD -- The current working directory. This may be unset or null if shell does not know where it is.
* RANDOM -- A simple random number generator. Every time RANDOM is referenced, it is assigned the next number in a random number series. The point in the series can be set by assigning a number to RANDOM.
* REPLY -- Default parameter for the read command if no names are given.
* SECONDS -- The number of seconds since the shell started or, if the parameter has been assigned an integer value, the number of seconds since the assignment plus the value that was assigned.
* TMOUT -- If set to a positive integer in an interactive shell, it specifies the maximum number of seconds the shell will wait for input after printing the primary prompt (PS1). If this time is exceeded, the shell exits.
* TMPDIR -- Where the directory shell temporary files are created. If this parameter is not set, or does not contain the absolute path of a directory, temporary files are created in /tmp.

**Practical -13**

**Aim : Write a shell script to validate the entered date. (eg. Date format is :dd-mm-yyyy).**

DATETIME=$1

#validate datetime..

tmp=`date -d "$DATETIME" 2>&1` ; #return is: "date: invalid date `something'"

if [ "${tmp:6:7}" == "invalid" ]; then echo "Invalid datetime: $DATETIME" ;

else

... validdatetime, do something with it ...

fi

**Practical -14**

**Aim : Write a program for process creation using C. (Use of gcc compiler).**

#include<sys/types.h>  
#include<stdio.h>  
#include<process.h>  
int main()  
{  
int pid\_t,pid,pid1,p,p1;  
pid =fork();  
if (pid ==-1)  
{  
printf("enter in connection");  
}  
else  
if(pid==0)

{  
printf("\n child process1 :\n\n");  
p=getppid();  
printf("parent process id of child1: %d\n",p);  
p1=getpid();  
printf("parent process id of child1: %d\n",p1);  
}  
else  
{  
pid1=fork();  
if(pid==0)  
{  
printf("\nchild process 2:\n\n");  
p=getppid();  
printf("parent process id of child2: %d\n",p);  
p1=grtpid();  
printf("parent process id of child2: %d\n",p1);  
}  
  
else  
{  
printf("this is parent process \n");  
p=getppid();  
printf("grant parent: %d \n",p);  
p1=getpid();  
printf("process id of parent: %d \n",p1);  
}  
}  
return 0;  
}