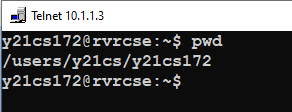
**SKILL ORIENTED COURSE - 2**

**CSSL2 – LINUX PROGRAMMING**

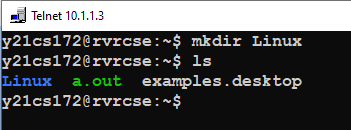
**MODULE – 1**

**DIRECTORY RELATED UTILITIES**

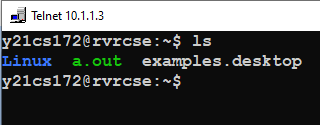
1. **pwd:** It is used to print the full path of the present working directory starting from the root.



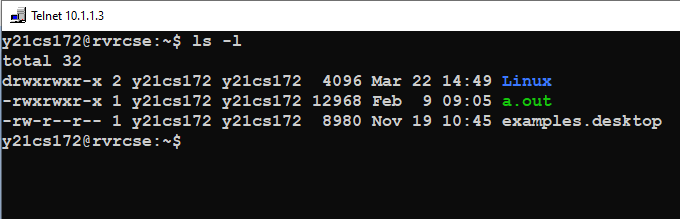
1. **mkdir:** It is used to create directory(ies), if they do not exist.



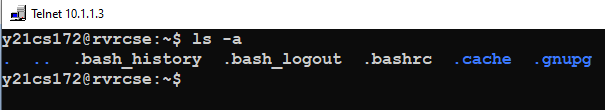
1. **ls:** It is used to list information about the files of the current directory by default (or) a specified directory.
   1. This command lists only the viewable contents of the directory



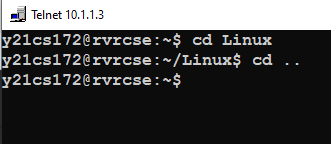
* 1. This command is used to print a larger list with all the permissions, users and owner names



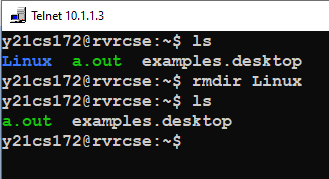
* 1. This command is used to print the hidden files that start with ‘.’ in the directory



1. **cd:** It is used to change the directory to the home directory by default (or) to a specified directory



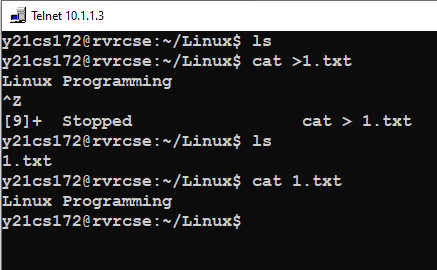
1. **rmdir:** It is used to remove directory(ies) only if they are empty.



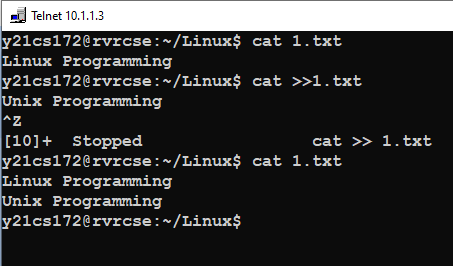
**MODULE – 2**

**FILE HANDLING AND TEXT PROCESSING**

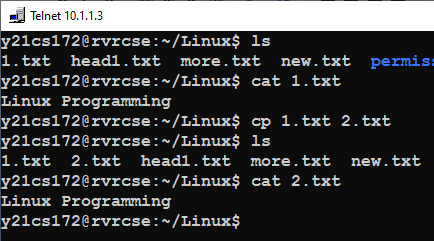
1. **cat:** It is used to concatenate files and print on the standard output
   1. **cat > 1.txt:** Creates a new file and write content to it



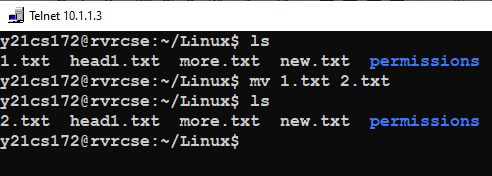
* 1. **cat 2.txt:** Prints the content of the file on standard output
  2. **cat >> 2.txt:** Used to append extra content to the file



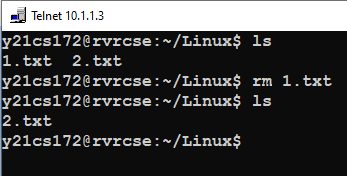
1. **cp:** It is sued to copy files or directories from a source to a destination in the file system.



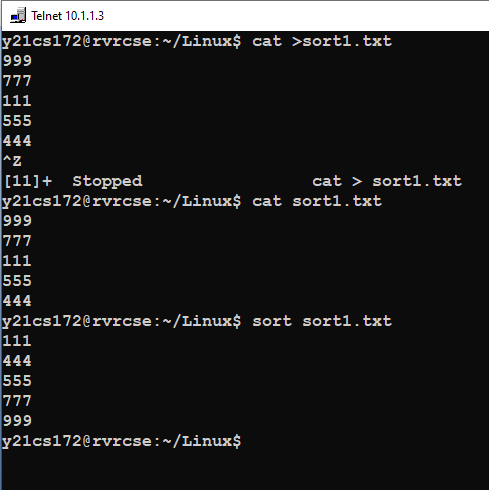
1. **mv:** It is used to move a file from one location to another or to rename a file.



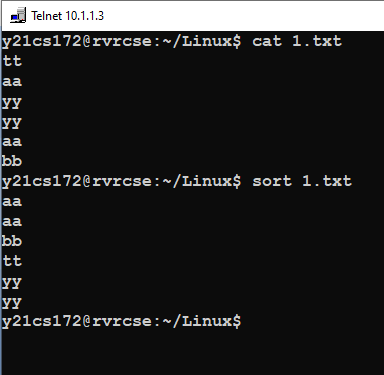
1. **rm:** It is used to remove specified files or directories.



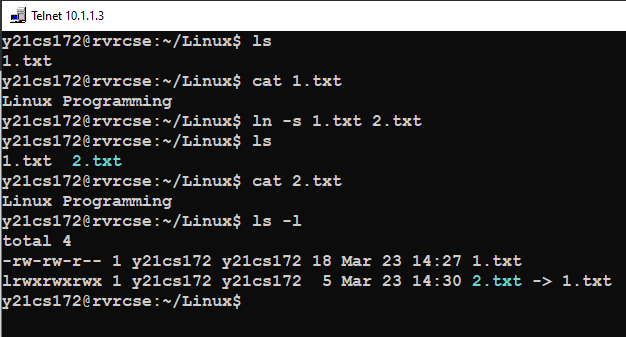
1. **sort:** It is used to sort the content of a given file and prints it on the standard output
   1. Sorting content based on numbers



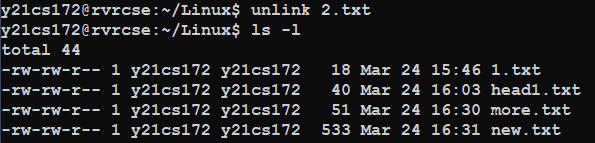
* 1. Sorting content based on characters



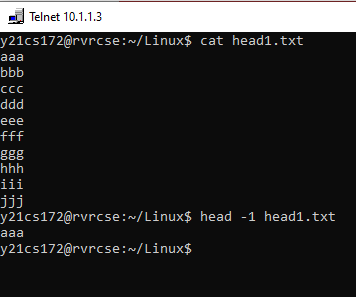
1. **ln –s:** It is used to create symbolic links between two files

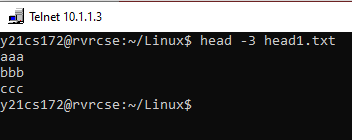


1. **unlink:** It is used to remove the symbolic links between files

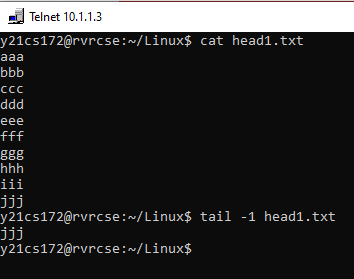
****

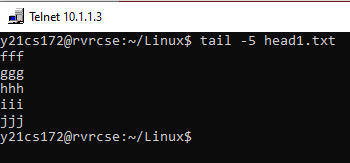
1. **head:** It is used to print the specified number of lines of content of a file from the starting





1. **tail:** It is used to print the specified number of lines of content of a file from the bottom.





1. **find:** It is used to find files or directories in a specified directory and of a specified name expression
2. **more:** It is used for paging through text one screen full at a time.
3. **w:** It is used to display information about the users currently on the machine, and their processes.
4. **nl:** It is used to write each file to standard output with line numbers added.
5. **grep:** It searches for a pattern in the given files
   1. *–wn* options are used to restricts matching to whole words only with line numbers

* 1. To display only those lines in a file that don’t match, use the *-v* option

1. **egrep:** It is used to search for extended regular expression patterns

1. **fgrep:** It is used to search for fixed strings in the specified files

1. **uniq:** The uniq utility displays a file with all of its identical adjacent lines replaced by a singleoccurrence of the repeated line
   1. filter out duplicate adjacent lines
   2. display a count with the lines
   3. ignore first field of each line.
2. **chmod:** It is used to change user’s, group’s and owner’s read, write and execute permissions of a file or directory
   1. Changing user permissions
   2. Changing group permissions
   3. Changing owner permissions
3. **paste:**It is used to parallel merge or join two files by outputing lines consisting of each line separated by **tab**delimiter
4. **cut:** It is used for cutting out the sections from each line and displaying on standard output.
   1. *–b* option to cut specific bytes of each line in a file. We need to mention bytes followed by commands

* 1. *–c* option to specify the columns to print in each line of a file.

* 1. *–f* option is used to print the specified field number and the *–d*  option specifies the delimiter to separate the fields

1. **join:** It is used to join the lines of two files based on a common field.

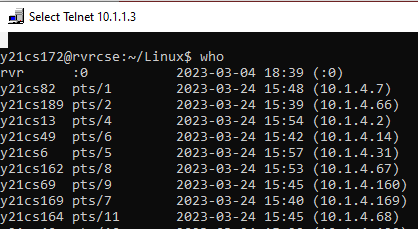
1. **tee:** It is used to read from standard input and write to standard output and files
   1. Without any options
   2. *–a* command it used to append the standard input to a file rather than overwriting

1. **cmp:** It compares two files byte by byte and returns at which byte the files first differ
2. **diff:**It is used to compare two files and display a list of editing changes that would convert the first file into the second file. It displays three kinds of editing changes: ‘a’ adding lines, ‘c’ changing lines and ‘d’ deleting lines

**MODULE – 3**

**DISK UTITLITIES, BACKUP AND OTHER UTILITIES**

1. **who:** It is used to print the information about users who are currently logged in.



1. **du:** It is used to summarize disk usage of the set of files, recursively for directories.
2. **df:** It displays the amount of disk space available on the file system containing each file name argument. If not file name is give, the space available on all currently mounted file systems is shown.
3. **sed:** The *s*tream *ed*itor utility sed scans one or more files and performs an editing action on all of the lines that match a particular condition.
   1. Substituting text: Substituting first character with a space in each line
   2. Deleting text: Deleting only those lines that contain the word ‘a’
   3. Inserting text:
   4. Replacing text:

**MODULE – 4**

**PROGRAMMABLE TEXT PROCESSING**

**awk**is a programmable text-processing utility that scans the lines of its input and performs actions on every line that matches a particular criterion

1. **Accessing individual files:**
2. **Begin and End**
3. **Operators**
4. **Variables**
5. **Control Structures**
6. **Extended regular expressions**
7. **Condition Ranges**
8. **Field Separators**
9. **Built-In functions**

**MODULE - 5**

**SHELL SCRIPTING**

**Write a shell script program for the following:**

1. To create a directory and list all the directory files in a directory.

**echo 'Enter the Directory name to create:'**

**read Dir\_name**

**if [ -e $Dir\_name ]**

**then**

**echo "$Dir\_name directory already exists!"**

**else**

**mkdir $Dir\_name**

**echo "$Dir\_name directory created."**

**fi**

**echo 'Enter a Directory name to Display Files: '**

**read Dir\_name**

**cd $Dir\_name**

**echo 'List of Directory files in the directory: '$Dir\_name**

**for i in \*/**

**do**

**echo "$i"**

**done**

1. To display a list of all the files in the current directory

**echo "Current directory:"**

**pwd**

**echo "Files in current directory:"**

**ls -a**

1. To count no of lines, words, and characters of an input file.

**read -p "Enter the file name: " filename**

**if [ ! -f "$filename" ]; then**

**echo "File not found!"**

**exit 1**

**fi**

**line\_count=$(wc -l < "$filename")**

**word\_count=$(wc -w < "$filename")**

**char\_count=$(wc -m < "$filename")**

**echo "Number of lines: $line\_count"**

**echo "Number of words: $word\_count"**

**echo "Number of characters: $char\_count"**

1. To accept a file name starting and ending line numbers as arguments and display all the lines between given line numbers.

**read -p "Enter file name: " filename**

**if [ ! -f "$filename" ]; then**

**echo "File not found!"**

**exit 1**

**fi**

**read -p "Enter starting line:" start\_line**

**read -p "Enter ending line:" end\_line**

**if [ "$start\_line" -gt "$end\_line" ]; then**

**echo "Start line cannot be greater than end line!"**

**exit 1**

**fi**

**sed -n $start\_line,$end\_line\p $filename**

1. To deletes all lines containing the specified word in one or more files supplied as arguments to it.
2. To test whether the given file is existing or not.

**read -p "Enter filname: " filename**

**if [ -f $filename ]**

**then**

**echo "$filename exists!"**

**else**

**echo "$filename doesn't exist!"**

**fi**

1. To read, delete and append a file.

**read -p "Enter the file name: " fname**

**if [ -f $fname ]**

**then**

**echo "Contents of file: "**

**cat $fname**

**else**

**echo "File does not exist"**

**fi**

**read -p "Enter the file name to delete: " fname**

**if [ -f $fname ]**

**then**

**rm $fname**

**echo "File deleted"**

**else**

**echo "File does not exist"**

**fi**

**read -p "Enter the file name to append: " fname**

**if [ -f $fname ]**

**then**

**echo "Enter text to append: "**

**cat >> $fname**

**else**

**echo "File does not exist"**

**fi**

1. To store all command line arguments to an array and print.

**arr=$@**

**echo "The arguments are: "**

**for i in $arr**

**do**

**echo $i**

**done**

1. To print the calendar month by default

**month = $1**

**year = $2**

**echo "Calender of the given month and year:"**

**cal $month $year**

**MODULE-6**

**SYSTEM CALLS**

1. Write a program on File management System Calls: open (), read (), write (), close ().

**#include<stdio.h>**

**#include<fcntl.h>**

**#include<stdlib.h>**

**#include<string.h>**

**void main()**

**{**

**int fd1,fd2;**

**fd1 = open("sample.txt", O\_RDONLY);**

**fd2 = open("test.txt", O\_CREAT | O\_RDWR, 0700);**

**printf("fd1 = %d\n",fd1);**

**printf("fd2 = %d\n",fd2);**

**char \*c = (char\*)malloc(20\*sizeof(char));**

**int s = read(fd1,c,10);**

**c[s] = '\0';**

**printf("Contents of first %d bytes of fd1: %s\n",s,c);**

**write(fd2, "RVR & JC", 8);**

**close(fd1);**

**close(fd2);**

**}**

1. Write a program on File handling system call: perror ().

**#include<stdio.h>**

**#include<errno.h>**

**#include<sys/file.h>**

**void main()**

**{**

**int fd1, fd2;**

**fd1 = open("nonexist1.txt",O\_RDONLY);**

**if(fd1 == -1)**

**{**

**printf("errno = %d\n",errno);**

**perror("Could not open the file to read");**

**}**

**fd2 = open("nonexist2.txt",O\_WRONLY);**

**if(fd2 == -1)**

**{**

**printf("errno = %d\n",errno);**

**perror("Could not open the file to write");**

**}**

**}**

1. Write a program for demonstrating dup () and dup2() system calls.

**#include<stdio.h>**

**#include<unistd.h>**

**#include<fcntl.h>**

**void main()**

**{**

**int old\_fd, new\_fd1, new\_fd2;**

**old\_fd = open("test.txt", O\_RDWR);**

**printf("The Old File Descriptor is: %d\n",old\_fd);**

**new\_fd1 = dup(old\_fd);**

**printf("The First New File Descriptor is: %d\n",new\_fd1);**

**new\_fd2 = dup2(old\_fd, 7);**

**printf("The Second New File Descriptor is: %d\n",new\_fd2);**

**close(old\_fd);**

**}**

1. Write a program to create two processes, to run a loop in which one process adds all even numbers and other process adds all odd numbers (use fork () system call).

**#include<stdio.h>**

**#include<sys/types.h>**

**#define Max\_count 10**

**void EvenSum();**

**void OddSum();**

**void main()**

**{**

**pid\_t pid = fork();**

**if(pid==0)**

**EvenSum();**

**else**

**OddSum();**

**}**

**void OddSum()**

**{**

**int sum=0;**

**for(int i=1;i<=Max\_count;i++)**

**if(i%2 != 0)**

**sum += i;**

**printf("Sum of Odd Numbers upto %d: %d\n",Max\_count,sum);**

**}**

**void EvenSum()**

**{**

**int sum=0;**

**for(int i=1;i<=Max\_count;i++)**

**if(i%2 == 0)**

**sum += i;**

**printf("Sum of Even Numbers upto %d: %d\n",Max\_count,sum);**

**}**

1. Write a Program to create orphan process.

**#include<stdio.h>**

**#include<sys/types.h>**

**#include<unistd.h>**

**void main()**

**{**

**int pid = fork();**

**if(pid>0)**

**printf("In Parent Process");**

**else if(pid==0)**

**{**

**sleep(5);**

**printf("In Cild Process");**

**}**

**}**

1. Write a Program to create a zombie process and how to avoid Zombie using wait ().

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<unistd.h>**

**#include<sys/wait.h>**

**#include<sys/types.h>**

**void main()**

**{**

**pid\_t child = fork();**

**if(child>0)**

**{**

**printf("Parent process start\n");**

**wait(&child);**

**printf("Parent process end\n");**

**}**

**else if(child==0)**

**{**

**printf("Child process start\n");**

**sleep(10);**

**printf("Child process end\n");**

**exit(0);**

**}**

**}**

1. Write a program for Requesting an alarm signal to execute user defined alarm handler.

**#include<stdio.h>**

**#include<signal.h>**

**int alarm\_flag = 0;**

**void alarm\_handler();**

**void main()**

**{**

**signal(SIGALRM, alarm\_handler);**

**alarm(5);**

**printf("Looping...\n");**

**while(!alarm\_flag)**

**{**

**printf("Inside the Loop\n");**

**pause();**

**}**

**printf("Loop ends due to alarm signal\n");**

**}**

**void alarm\_handler()**

**{**

**printf("An alarm clock signal was recieved\n");**

**alarm\_flag = 1;**

**}**

1. Write a program to demonstrate Suspending and Resuming Processes.

**#include<stdio.h>**

**#include<signal.h>**

**void main()**

**{**

**int pid1, pid2;**

**pid1 = fork();**

**if(pid1==0)**

**{**

**while(1)**

**{**

**printf("Process 1 is alive\n");**

**sleep(1);**

**}**

**}**

**pid2 = fork();**

**if(pid2==0)**

**{**

**while(1)**

**{**

**printf("Process 2 is alive\n");**

**sleep(1);**

**}**

**}**

**sleep(3);**

**kill(pid1, SIGSTOP);**

**sleep(3);**

**kill(pid1, SIGCONT);**

**sleep(3);**

**kill(pid1, SIGINT);**

**kill(pid2, SIGINT);**

**}**

**MODULE – 7**

**INTER PROCESS COMMUNICATION**

1. Write a program to implement the concept of pipes.
2. Write a program to implement the concept of shared memory.
3. Write a program to implement the concept of semaphores.