COMMANDS

|  |  |
| --- | --- |
| **1)** | **cal**:- Displays a calendar |
|  | **Syntax**:- cal [options] [ month ] [year] |
|  | **Description** :-   * cal displays a simple calendar. If arguments are not specified, the current month is displayed. The switching options are as follows: |
|  | |  |  | | --- | --- | | -1 | Display single (current) month output. (This is the default.) | | -3 | Display prev/current/next month output | | -s | Display Sunday as the first day of the week (This is the default.) | | -m | Display Monday as the first day of the week | | -j | Display Julian dates (days one-based, numbered from January 1) | | -y | Display a calendar for the current year | |
|  | **Example:-**  **$cal**  **or**  **$cal 02 2016**  **Feb 2016**   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Su** | **Mo** | **Tu** | **We** | **Th** | **Fr** | **Sa** | |  | 1 | 2 | 3 | 4 | 5 | 6 | | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 14 | 15 | 16 | 17 | 18 | 19 | 20 | | 21 | 22 | 23 | 24 | 25 | 26 | 27 | | 28 | 29 |  |  |  |  |  | |
|  |  |
| **2)** | **clear** :- It clears the terminal screen. |
|  | **Syntax** :- clear |
|  | **Description** :-   * Clear clears your screen if this is possible, including its scroll back buffer. * Clear ignores any command-line parameters that may be present. |
|  |  |
| **3)** | **pwd :-** Displays path from root to current directory |
|  | **Syntax :-** pwd [options] |
|  | **Example:**  **$ pwd**  /home/kumar/progs |
|  |  |
| **4)** | **cd**:- It is used to change the directory. |
|  | **Syntax** :- cd [directory] |
|  | **Description:-**   * Used to go back one directory on the majority of all UNIX shells. It is important that the space be between the cd and directory name or .. |
|  | **Example:-**  **$ pwd**  /home/kumar  **$ cd progs**  **$ pwd**  /home/kumar/progs  **$ cd ..**  /home/kumar |
|  |  |
| **5)** | **ls**:- Lists the contents of a directory |
|  | **Syntax** :- ls [options] |
|  | **Description** :-   |  |  | | --- | --- | | -a | Shows you all files, even files that are hidden (these files begin with a dot.) | | -A | List all files including the hidden files. However, does not display the working directory (.) or the parent directory (..). | | -d | If an argument is a directory it only lists its name not its contents | | -l | Shows you huge amounts of information (permissions, owners, size, and when last modified.) | | -p | Displays a slash ( / ) in front of all directories | | -r | Reverses the order of how the files are displayed | | -R | Includes the contents of subdirectories | |
|  | **Example:-**  **$ ls – l**  -rw-r----- 1 student student 23 Jan 16 15:27 file1.txt |
|  | **Field Explanation:**   * If first character is – then it is normal file * If it is d then it is directory * **Field 1 – File Permissions**: Next 9 character specifies the files permission. Each 3 characters refers to the read, write, execute permissions for user, group and world In this example, -rw-r—– indicates read-write permission for user, read permission for group, and no permission for others. * **Field 2 – Number of links**: Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file. * **Field 3 – Owner**: Third field specifies owner of the file. In this example, this file is owned by username ‘student’. * **Field 4 – Group**: Fourth field specifies the group of the file. In this example, this file belongs to ”student’ group. * **Field 5 – Size**: Fifth field specifies the size of file. In this example, ’13′ indicates the file size. * **Field 6 – Last modified date & time**: Sixth field specifies the date and time of the last modification of the file. In this example, ‘Jan 16 15:27′ specifies the last modification time of the file. * **Field 7 – File or directory name**: The last field is the name of the file or directory. In this example, the file name is file1.txt |
|  |  |
| **6)** | **exit** :- It is used to terminate a program, shell or log you out of a network normally. |
|  | **Syntax** :- exit |
|  |  |
| **7)** | **echo** :- It prints the given input string to standard output. |
|  | **Syntax** :- echo string  **Example:-**  **$ echo “hi.. hello unix”**  hi.. hello unix |
|  |  |
| **8)** | **who** :- who command can list the names of users currently logged in, their terminal, the time they have been logged in, and the name of the host from which they have logged in. |
|  | **Syntax** :- who [options] [file] |
|  | **Description**:- |
|  | |  |  | | --- | --- | | am i | Print the username of the invoking user, The 'am' and 'i' must be space separated. | | -b | Prints time of last system boot. | | -d | print dead processes. | | -H | Print column headings above the output. | | -l | Include idle time as HOURS:MINUTES. An idle time of . indicates activity within the last minute. | | -m | Same as who am i. | | -q | Prints only the usernames and the user count/total no of users logged in. | |
|  | **Example :-** |
|  | **$ who**  dietstaffpts/1 2016-02-20 22:42 (:0.0)  dietstaffpts/2 2016-02-20 09:30 (:0.0)   * Here first column shows user name, second shows name of the terminal the user is working on. Third& fourth column shows date and time of logging, last column shows machine name. |
|  |  |
| **9)** | **who am i**:- Print effective userid |
|  | **Syntax** :- who am i |
|  | **Description: -** Print the user name associated with the current effective user id. |
|  | **Example :-**  **$ who am i**  dietstaffpts/3 2016-02-10 08:52 (:0.0)   * Here first column shows user name, second shows name of the terminal the user is working on. Third & fourth column shows date and time of logging, last column shows machine name. |
| **10)** | **mkdir**:- This command is used to create a new directory |
|  | **Syntax** :- mkdir [options] directory |
|  | **Description** :-   |  |  | | --- | --- | | -m | Set permission mode (as in chmod) | | -p | No error if existing, make parent directories as needed. | | -v | Print a message for each created directory | | directory | The name of the directory that you wish to create | |
|  | **Example:-**  **$ mkdir aaa**   * The above command will create directory named aaa under the current directory. We can also create number of subdirectories with one mkdir command. |
|  |  |
| **11)** | **rmdir**:- It is used to delete/remove a directory and its subdirectories. |
|  | **Syntax** :- rmdir [options..] Directory |
|  | **Description** :-   * It removes only empty directory.  |  |  | | --- | --- | | -p | Allow users to remove the directory and its parent directories which become empty. | |
|  |  |
| **12)** | **bc**:- bc command is used for command line calculator. It is similar to basic calculator. By using which we can do basic mathematical calculations. |
|  | **Syntax** :- bc [options] |
|  | **Description** :-   * bc is a language that supports arbitrary precision numbers with interactive execution of statements. * bc starts by processing code from all the files listed on the command line in the order listed. After all files have been processed, bc reads from the standard input. All code is executed as it is read.  |  |  | | --- | --- | | -q | To avoid bc welcome message | | -l | To include math library functionalities | |
|  | **Example:-**  **$ bc**  bc 1.06 Copyright 1991-1994,1997,1998,2000 Free Software Foundation,Inc. This is free software with ABSOLUTELY NO WARRANTY. For details type `warranty'. 2\*3 6   * The above command used is for mathematical calculations.   **$ bc -l**  bc 1.06 Copyright 1991-1994,1997,1998,2000 Free Software Foundation,Inc. This is free software with ABSOLUTELY NO WARRANTY.For details type `warranty'.11+2 13   * The above command displays the sum of '11+2'.   **$ bc calc.txt**  bc 1.06 Copyright 1991-1994,1997,1998,2000 Free Software Foundation,Inc. This is free software with ABSOLUTELY NO WARRANTY. For details type `warranty'. 13   * 'calc.txt' file have the following code:11+2. Get the input from file and displays the output. |
|  |  |
| **13)** | **uname**:- It is used to print system information. |
|  | **Syntax** :- uname [options] |
|  | **Description** :-   * Print certain system information.  |  |  | | --- | --- | | -s | print the kernel name | | -n | print the network node hostname | | -r | print the kernel release | | -v | print the kernel version | | -m | print the machine hardware name | | -o | print the operating system |   **Example:-**  **$ uname**  Linux |
|  |  |
| **14)** | **tty**:- Print the file name of the terminal connected to standard input. |
|  | **Syntax** :- tty |
|  | **Description** :-   * tty writes the name of the terminal that is connected to standard input onto standard output. * Command is very simple and needs no arguments. |
|  | **Example** :-  **$tty**  /student/tty10 |
|  |  |
| **15)** | **stty**:- Change and print terminal line settings. |
|  | **Syntax** :- sty |
|  | **Description** :-   * stty sets certain terminal I/O modes for the device that is the current standard input. * Without arguments, it writes the settings of certain modes to standard output. |
|  | **Example** :-  **$ sty**  speed 19200 baud, 25 rows, 79 columns  kill = ^U |
|  |  |
| **16)** | **cat**:- It is used to create, display and concatenate file contents. |
|  | **Syntax** : - cat [options] [FILE]... |
|  | **Description** :-   |  |  | | --- | --- | | -A | Show all. | | -b | Omits line numbers for blank space in the output. | | -e | A $ character will be printed at the end of each line prior to a new line. | | -E | Displays a $ (dollar sign) at the end of each line. | | -n | Line numbers for all the output lines. | | -s | If the output has multiple empty lines it replaces it with one empty line. | | -T | Displays the tab characters in the output. | | -v | Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly. |  * Two basically three uses of the cat command.  1. Create new files. 2. Display the contents of an existing file. 3. Concatenate the content of multiple files and display.   **Example :-**  **$ cat file1.c**   * Above syntax will display the content of file1.c   **$ cat > file1.c**   * Above syntax creates file1.c and allow us to insert content for this file. * After inserting content you can use ctrl+d to exit the file. * If file with same name exist then it will overwrite that file.   **$ cat file1.c**  process management  memory management  file mgmt  **$ cat file1.c >> file2.c**   * It can concatenate the contents of two files. For this you have to use append output redirection operator. * The contents of file2.cwill be appended to file1.c. |
|  |  |
| **17)** | **cp**:- cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten). |
|  | **Syntax** :- cp [options]... source destination |
|  | **Description:-**   * Here, after cp command contents of both source file and destination file files are the same. * It will copy the content of source file to destination file. * If the destination file doesn’t exist, it will be created. * If it exists then it will be overwritten without any warning. * If there is only one file to be copied then destination can be the ordinary file or the directory file.  |  |  | | --- | --- | | -a | archive files | | -f | force copy by removing the destination file if needed | | -i | interactive - ask before overwrite | | -l | link files instead of copy | | -L | follow symbolic links | | -n | no file overwrite | | -u | update - copy when source is newer than dest | |
|  | **Example:-**  **$ cp file1 file2**   * The above cp command copies the content of file1.php to file2.php.   **Copy folder and subfolders:**  **$ cp-R scripts scripts1**   * The above cp command copy the folder and subfolders from scripts to scripts1 |
| **18)** | **rm**:- It is used to remove/delete the file from the directory. |
|  | **Syntax** :- rm [options..] [file|directory] |
|  | **Description** :-   * Files can be deleted with rm. It can delete more than one file with a single invocation. For deleting a single file we have to use rm command with filename to be deleted. * Deleted file can’t be recovered. rm can’t delete the directories. If we want to remove all the files from the particular directory we can use the \* symbol. |
|  | |  |  | | --- | --- | | -f | Ignore nonexistent files, and never prompt before removing. | | -i | Prompt before every removal. | |
|  | **Examlple :-**  **$ rm myfile.txt**   * Remove the file myfile.txt. If the file is write-protected, you will be prompted to confirm that you really want to delete it.   **$ rm \***   * Remove all files in the working directory. If it is write-protected, you will be prompted before rm removes it.   **$ rm -f myfile.txt**   * Remove the file myfile.txt. You will not be prompted, even if the file is write-protected; if rm can delete the file, it will.   **$ rm -f \***   * Remove all files in the working directory. rm will not prompt you for any reason before deleting them.   **$ rm -i \***   * Attempt to remove every file in the working directory, but prompt before each file to confirm. |
|  |  |
| **19)** | **mv**:- It is used to move/rename file from one directory to another. |
|  | **Syntax** :- mv [options] oldname newname |
|  | **Description** :- |
|  | * mv command which is short for move. * mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another. * mv has two functions: it renames a file and it moves a group of files to a different directory. Mv doesn’t create a copy of the file , it merely renames it. No additional space is consumed on disk during renaming. For example if we rename a file os to os1 and then if we try to read file os we will get error message as it is renamed to os1 there is no existence of file named os.  |  |  | | --- | --- | | -f | mv will move the file(s) without prompting even if it is writing over an existing target. Note that this is the default if the standard input is not a terminal | | -i | Prompts before overwriting another file | |
|  | **Example:-**  **$ cat file1**  Memory  Process  Files  **$ mv file1 file2**   * rename file1 to file2 * If the destination file doesn’t exist it will be created. mv can also be used to rename a directory. A group of files can also be moved to a directory. mv doesn’t prompt for overwriting destination file if it exists. |
| **20)** | **nl :-**  nl numbers the lines in a file. |
|  | **Syntax: -** nl [OPTION] [FILE] |
|  | **Example :-** |
|  | **$cat list.txt**  apples  oranges  potatoes  lemons  garlic  **$nl list.txt**  1 apples  2 oranges  3 potatoes  4 lemons  5 garlic   * In the above example, we use the cat command to display the contents of list.txt. Then we use nl to number each line and display the result to standard output.   **$nl list.txt > nlist.txt**  **$cat nlist.txt**  1 apples  2 oranges  3 potatoes  4 lemons  5 garlic   * In the above example, we run the same nl command, but redirect the output to a new file, nlist.txt. Then we use cat to display the results. |
|  |  |
| **21)** | **cut** :- cut command is used to cut out selected fields of each line of a file. The cut command uses delimiters to determine where to split fields. |
|  | **Syntax** :- cut [options] filename |
|  | **Description :-**   |  |  | | --- | --- | | file | A path name of an input file. If no file operands are specified, or if a file operand is -, the standard input will be used | | -c | The list following -c specifies character positions | | -d | The character following -d is the field delimiter | | -f | select only these fields on each line | | -b | Select only the bytes from each line as specified in LIST | |
|  | **Example :-**   * For example, let's say you have a file named data.txt which contains the following text:   one two three four five  alpha beta gamma delta epsilon   * In this example, each of these words is separated by a tab character, not spaces. The tab character is the default delimiter of cut, so it will by default consider a field to be anything delimited by a tab. * To "cut" only the third field of each line, use the command:   **$ cut -d’ ‘ -f 3 data.txt**  three  gamma   * let's say you want the third field and every field after it, omitting the first two fields. In this case, you could use the command:   **$ cut -d’ ‘ -f 3- data.txt**  three four five  gamma delta epsilon  **$ cut -c 3 file.txt**  r  m   * For example, to output only the third-through-twelfth character of every line of data.txt, use the command:   **$ cut -c 3-12 data.txt**  e two thre  pha beta g |
|  |  |
| **22)** | **paste**:- paste command is used to paste the content from one file to another file. It is also used to set column format for each line. |
|  | **Syntax** :- paste [option] file |
|  | **Description** :-   * Paste prints lines consisting of sequentially corresponding lines of each specified file. In the output the original lines are separated by TABs. The output line is terminated with a newline. |
|  | |  |  | | --- | --- | | -d | Specify of a list of delimiters. | | -s | Paste one file at a time instead of in parallel. | | File | A path name of an input file. If - is specified for one or more of the file s, the standard input will be used. | |
|  | **Example:-**  **$ cat > a**  Unix  Linux  Windows  **$ cat > b**  Dedicated server  Virtual server  **$ paste a b**  Unix Dedicated server  Linux Virtual server  Windows  **$ paste -d"|" a b**  Unix|Dedicated server  Linux|Virtual server  Windows|  **$ paste –s a b**  Unix Linux Windows  Dedicated server Virtual server  **$ paste –s –d”,” a b**  Unix,Linux,Windows  Dedicated server,Virtual server |
|  |  |
| **23)** | **more**:- Displays text one screen at a time. |
|  | **Syntax** :- more [options] filename |
|  | **Description** :-   * More command displays its output a page at a time. For example we are having a big file with thousands of records and we want to read that file then we should use more command. Consider a file named employees then we will use following command to read its contents a page at a time.  |  |  | | --- | --- | | -c | Clear screen before displaying. | | -e | Exit immediately after writing the last line of the last file in the argument list. | | -n | Specify how many lines are printed in the screen for a given file. | | +n | Starts up the file from the given number. | |
|  | **Example:-**  **$ more -c myfile.txt**   * Clears the screen before printing the file .   **$ more -3 myfile.txt**   * Prints first three lines of the given file. Press Enter to display the file line by line.   **$ more +/"hope" myfile.txt**   * Display the contents of file myfile.txt, beginning at the first line containing the string "hope". |
|  |  |
| **24)** | **cmp**:- It compares two files and tells you which line numbers are different. |
|  | **Syntax** : - cmp [options..] file1 file2 |
|  | **Description** :-   * Let’s create a file named os2. And use cmp command to compare os and os1files.  |  |  | | --- | --- | | - c | Output differing bytes as characters. | | - l | Print the byte number (decimal) and the differing byte values (octal) for each difference. | | - s | Prints nothing for differing files, return exit status only. | | - c | Output differing bytes as characters. | |
|  | **Example:-**  **$ cat file1**  memory  process  files  **$ cat > file2**  memory  process  files mgmt  **$ cmp file1 file2**  File1 file2 differ: char 21, line 3   * The two files are compared byte by byte and the location of the first mismatch is echoed to the screen. cmp doesn’t bother about possible subsequent mismatches. |
|  |  |
|  |  |
| **25)** | **comm**:- compare two sorted files line by line |
|  | **Syntax**:- comm [option]... FILE1 FILE2 |
|  | **Description** :-   * Compare sorted files FILE1 and FILE2 line by line. * Requires two sorted files and lists differing entries in different columns. produces three text columns as output:   **1** Lines only in file1.  **2** Lines only in file2.  **3** Lines in both files.   |  |  | | --- | --- | | -1 | suppress lines unique to FILE1 | | -2 | suppress lines unique to FILE2 | | -3 | suppress lines that appear in both files |  * For example let’s create two files named file1 and file2 with following data.   **$ cat > file1**  c.k.shukla  chanchalsanghvi  s.n.dasgupta  sumit [1] + Stopped  **$ cat > file2**  anilaggarwal  barunsengupta  c.k.shukla  lalit   * Now let’s use comm. Command with these two files.   **$ comm file1 file2**  anilaggarwal  barunsengupta  c.k.shukla  chanchalsanghvi  lalit  s.n.dasgupta  sumit   * In the above output we can see that first column contains two lines unique to the first file and second column contains three lines unique to the second file and the third column contains two lines common to both the files. Comm. Can produce the single column output using 3 options -1,-2 or -3. To drop a particular column, simply use its column number as a prefix. |
|  | **Example :-**  **$ cat file1**  f1.c  f2.c  f3.c  f4.c  f5.c  **$ cat file2**  f1.c  f3.c  f4.c  f6.c  f7.c   * Now, When you run the comm command on these files, this is what you get:   **$ comm file1 file2**  f1.c  f2.c  f3.c  f4.c  f5.c  f6.c  f7.c   * The output is split in 3 columns. Column1 indicates files which are unique in file1, column 2 indicates files unique to file2. Column 3 indicates files common between them. comm command provides some real good options with which you can filter the output better. * Now, say you want to find out only the list of files which were there in the older version but not in the newer version:   **$ comm -23 file1 file2**  f2.c  f5.c   * The option -23 indicates to remove the second and third columns from the comm command output, and hence we are left with only the first column which is the files unique in file1. * Similarly, to find out the list of files which were not there in the old version, but has been added in the new version:   **$ comm -13 file1 file2**  f6.c  f7.c   * As explained above, -13 option tells to remove the first and third columns from the comm output. * Finally, to know the list of files which have been retained, or common in both the versions:   **$ comm -12 file1 file2**  f1.c  f3.c  f4.c   * When you apply comm command on files, the files should be sorted. This command works only on sorted files. |
| **26)** | **diff**:- It is used to find differences between two files. |
|  | **Syntax** :- diff [options..] fileone filetwo |
|  | **Description** :-   * Diff is the third command that can be used to display file differences. Unlike its fellow members ,cmp and comm. , it also tells us which lines in one file have to be changed to make the two files identical.  |  |  | | --- | --- | | -b | Ignore any changes which only change the amount of [whitespace](http://www.computerhope.com/jargon/w/whitspac.htm) (such as spaces or tabs). | | -w | Ignore whitespace entirely. | | -B | Ignore blank lines when calculating differences. | | -y | Display output in two columns. | | -i | Ignore changes in case.consider upper- and lower-case letters equivalent. | |
|  | **Example:-**  **$ cat 1.txt**  aaa  bbb  ccc  ddd  eee  fff  ggg  **$ cat 2.txt**  bbb  c c  ddd  eee  fff  ggg  hhh  **$ diff 1.txt 2.txt**  1d0  < aaa  3c2  < ccc  ---  > c c  7a7  > hhh   * Lines like "1d0" and "3c2" are the coordinates and types of the differences between the two compared files, while lines like "< aaa" and "> hhh" are the differences themselves. * diff change notation includes 2 numbers and a character between them. Characters tell you what kind of change was discovered: * **d – a line was deleted** * **c – a line was changed** * **a – a line was added** * Number to the left of the character gives you the line number in the original (first) file, and the number to the right of the character tells you the line number in the second file used in comparison. * So, looking at the two text files and the diff output above, you can see what happened:   **1d0**  **< aaa**   * This means that 1 line was deleted. < aaa suggests that the aaa line is present only in the original file.   **3c2**  **< ccc**  **---**  **> c c**   * And this means that the line number 3 has changed. You can see how this confirms that in the first file the line was "ccc", and in the second it now is "c c".   **7a7**  **> hhh**   * Finally, this confirms that one new line appeared in the second file, it's "hhh" in the line number 7. |
|  |  |
| **27)** | **chmod**:- chmod command allows you to alter / Change access rights to files and directories. |
|  | **Syntax**:- chmod [options] [MODE] FileName |
|  | **Description** :-   * chmod command is used to set the permissions of one or more files for all three categories of users (user,group and others ). It can be run only by the user and super user. Command can be used in two ways. Let’s first take a look at the abbreviations used by chmod command. |
|  | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Category | | Operation | | Permission | | | u | User | + | Assigns permission | r | Read permission | | g | Group | - | Removes permission | w | Write permission | | o | Others | = | Assigns absolute | x | Execute permission |   **File Permission :**   |  |  | | --- | --- | | **#** | **File Permission** | | 0 | none | | 1 | execute only | | 2 | write only | | 3 | write and execute | | 4 | read only | | 5 | read and execute | | 6 | read and write | | 7 | set all permissions | |
|  | **Relative permissions**   * When changing permissions in a relative manner, chmod only changes the permissions specified in the command line and leaves the other permissions unchanged. In this mode it uses the following syntax: * let’s first check the permission of file shortlist. It shows read/write permission. Let’s assign execute permission to this file for owner.   **$ ls –l**  -rw-r--r—1 shortlist None 241 Feb 21 04:02  **$ chmod u+x shortlist**  **$ ls -l**  -rwxr--r-- 1 shortlist None 241 Feb 21 04:02   * Here chmod uses expression u+x means – u shows user category that is user, + shows we need to assign the permission and x shows execute permission. Sameway we can assign other permissions.   **Absolute permissions**   * Sometimes you don’t need to know what a file’s current permissions are, but want to set all nine permission bits explicitly. The expression used by chmod here is a string of three octal numbers. Each type of permission is assigned a number as shown:   Read permission-4  Write permission-2  Execute permission -1   * For each category we add numbers that represent the assigned permission. For instance , 6 represents read and write permission , and 7 represents all permissions.   **Example :**  **$ chmod 666 myfile or(chmod u=rw,g=rw,o=rwmyfile) shortlist**   * Shows read and wrie (4 +2) permissions for all three types of users. |
|  |  |
| **28)** | **chown**:- Command for system V that changes the owner of a file. |
|  | **Syntax** :- chown [options] newowner filename/directoryname |
|  | **Description:-**   |  |  | | --- | --- | | -R | Change the permission on files that are in the subdirectories of the directory that you are currently in. | | -c | Change the permission for each file. | | -f | Prevents chown from displaying error messages when it is unable to change the ownership of a file. | |
|  | **Example :-**  **$ ls -l demo.txt**  -rw-r--r-- 1 root root 0 Jan 31 05:48 demo.txt  **$ chownvivek demo.txt**  -rw-r--r-- 1 vivek root 0 Jan 31 05:48 demo.txt |
|  |  |
| **29)** | **chgrp**:- chgrp command is used to change the group of the file or directory. This is an admin command. Root user only can change the group of the file or directory. |
|  | **Syntax**:- chgrp [options] newgroup filename/directoryname |
|  | **Description:-**   |  |  | | --- | --- | | -R | Change the permission on files that are in the subdirectories of the directory that you are currently in. | | -c | Change the permission for each file. | | -f | Force. Do not report errors. | |
|  | **Example :-**  **$ ls -l demo.txt**  -rw-r--r-- 1 root root 0 Jan 31 05:48 demo.txt  **$ chgrp vivek demo.txt**  -rw-r--r-- 1 root vivek 0 Jan 31 05:48 demo.txt |
|  |  |
| **30)** | **file**:- file command tells you if the object you are looking at is a file or a directory. |
|  | **Syntax**:- file [options] directoryname/filename |
|  | **Description**:-   * File command is used to determine the type of file, especially of an ordinary file. We can use it with one or more filenames as arguments. For example we can use file command to check the type of the os1 file that we have created.   **$ file os1**  os1: short english text |
|  |  |
| **31)** | **finger**:- finger command displays the user's login name, real name, terminal name and write status (as a ''\*'' after the terminal name if write permission is denied), idle time, login time, office location and office phone number. |
|  | **Syntax**:- finger [username] |
|  | **Description :-**   |  |  | | --- | --- | | -l | Force long output format | | -s | Force short output format | |
|  | **Example :-**  **$ finger rahul**  Login: abc Name: (null)  Directory: /home/abc Shell: /bin/bash  On since Mon Nov 1 18:45 (IST) on :0 (messages off)  On since Mon Nov 1 18:46 (IST) on pts/0 from :0.0  New mail received Thu Feb 7 10:33 2015 (IST)  Unread since Mon Feb 8 12:59 2016 (IST)  No Plan. |
|  |  |
| **32)** | **sleep**:- Delay for a specified amount of time |
|  | **Syntax** :- sleep NUMBER[SUFFIX] |
|  | **Description**:-   * The sleep command pauses for an amount of time defined by NUMBER. * SUFFIX may be "s" for seconds (the default), "m" for minutes, "h" for hours, or "d" for days. |
|  | **Example :-**   * To sleep for 5 seconds, use:   **$ sleep 5**   * To sleep for 2 minutes, use:   **$ sleep 2m**   * To sleep for 3 hours, use:   **$ sleep 3h**   * To sleep for 5 days, use:   **$ sleep 5d** |
|  |  |
| **33)** | **kill**:- kill command is used to kill the background process. |
|  | **Syntax**:- kill [options] pid |
|  | **Description** :-   * The command kill sends the specified signal to the specified process or process group. * If no signal is specified, the TERM signal is sent. The TERM signal will kill processes which do not catch this signal. * For other processes, it may be necessary to use the KILL (9) signal, since this signal cannot be caught. |
|  | |  |  | | --- | --- | | pid. | Specify the list of processes that kill should signal | | -s | send the specified signal to the process | | -l | list all the available signals. | | -9 | Force to kill a process. | |
|  |  |
|  |  |
| **34)** | **users** :- users command displays currently logged in users. |
|  | **Syntax:**- users |
|  | **Output**:-  **$ users**  dietstaff dietstaff |
|  |  |
| **35)** | **ps**:- It is used to report the process status. ps is the short name for Process Status. |
|  | **Syntax**:- ps [options] |
|  | **Description** :- |
|  | |  |  | | --- | --- | | -a | List information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal | | -A | List information for all processes. Identical to -e, below | | -f | Generate a full listing | | -j | Print session ID and process group ID | | -l | Generate a long listing | |
|  | **Example :-**  **$ps**  PID TTY TIME CMD  291 console 0:00 bash |
|  |  |
| **36)** | **ln** :- ln command is used to create link to a file (or) directory. It helps to provide soft link for desired files. |
|  | **Syntax**:- ln [options] existingfile(or directory)name newfile(or directory)name |
|  | **Description**:-  **What Is A Link?**   * A link is an entry in your file system which connects a filename to the actual bytes of data on the disk. More than one filename can "link" to the same data. Here's an example. Let's create a file named file1.txt:   **$ echo "This is a file." > file1.txt**   * This command echoes the string "This is a file". Normally this would simply echo to our terminal, but the > operator redirects the string's text to a file, in this case file1.txt * When this file was created, the operating system wrote the bytes to a location on the disk and also linked that data to a filename, file1.txt so that we can refer to the file in commands and arguments. * If you rename the file, the contents of the file are not altered; only the information that points to it. * The filename and the file's data are two separate entities.   File1.txt  “This is a file””   * What the link command does is allow us to manually create a link to file data that already exists. * So, let's use link to create our own link to the file data we just created. In essence, we'll create another file name for the data that already exists.   **$ link file1.txt file2.txt**   * The important thing to realize is that we did not make a copy of this data. Both filenames point to the same bytes of data on the disk. Here's an illustration to help you visualize it:   File2.txt  File1.txt  “This is a file”   * If we change the contents of the data pointed to by either one of these files, the other file's contents are changed as well. Let's append a line to one of them using the >>operator:   **$ echo "It points to data on the disk." >> file1.txt**   * Now let's look at the contents of file1.txt:   **$ cat file1.txt**  This is a file.   * now let's look at the second file, the one we created with the link command.   **$ cat file2.txt**  This is a file.   * ln, by default, creates a hard link just like link does. So this ln command:   **$ ln file1.txt file2.txt**   * It is the same as the following link command. Because, both commands create a hard link named file2.txt which links to the data offile1.txt.   **$ link file1.txt file2.txt**   * However, we can also use ln to create symbolic links with the -s option. So the command:   **$ ln -s file1.txt file2.txt**   * It will create a symbolic link to file1.txt named file2.txt. In contrast to our hard link example, here's an illustration to help you visualize our symbolic link:   File2.txt  File1.txt  “This is a file””   * You should also be aware that, unlike hard links, removing the file (or directory) that a symlink(symbolic linkl) points to will break the link. So if we create file1.txt:   **$ echo "This is a file." > file1.txt**   * Now, create a symbolic link to it:   **$ ln -s file1.txt file2.txt**   * we can cat either one of these to see the contents:   **$ cat file1.txt**  This is a file.  **$ cat file2.txt**  This is a file.   * But, if we remove file1.txt:   **$ rm file1.txt**   * we can no longer access the data it contained with our symlink:   **$ cat file2.txt**  cat: file2.txt: No such file or directory |
|  | |  |  | | --- | --- | | -s | Makes it so that it creates a symbolic link | | -f | If the destination file or files already exist, overwrite them. | |
|  |  |
| **37)** | **head**:- head command is used to display the first ten lines of a file, and also specifies how many lines to display. |
|  | **Syntax**:- head [options] filename |
|  | **Description**:-   * Head command displays the top of the file. When used without an option , it displays the first ten lines of the file.  |  |  | | --- | --- | | -n | To specify how many lines you want to display. | | -n number | The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines. | | -c number | The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes. | |
|  | **Example :-**  **$ head myfile.txt**   * Display the first ten lines of myfile.txt.   **$ head -15 myfile.txt**   * Display the first fifteen lines of myfile.txt.   **$ head -c 20 myfile.txt**   * Will output only the first twenty bytes (characters) of myfile.txt. Newlines count as a single character, so if head prints out a newline, it will count it as a byte.   **$ head -n 5K myfile.txt**   * Displays the first 5,000 lines of myfile.txt. |
|  |  |
| **38)** | **tail**:- tail command is used to display the last or bottom part of the file. By default it displays last 10 lines of a file. |
|  | **Syntax** :- tail [options] filename |
|  | **Description**:-   |  |  | | --- | --- | | -l | To specify the units of lines. | | -b | To specify the units of blocks. | | -n | To specify how many lines you want to display. | | -c number | The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes. | | -n number | The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines. | |
|  | **Example : -**  **$ tail index.txt**   * It displays the last 10 lines of 'index.txt'.   **$ tail -2 index.txt**   * It displays the last 2 lines of 'index.txt'.   **$ tail -n 5 index.txt**   * It displays the last 5 lines of 'index.txt'.   **$ tail -c 5 index.txt**   * It displays the last 5 characters of 'index.txt'. |
|  |  |
| **39)** | **sort**:- It is used to sort the lines in a text file. |
|  | **Syntax**:- sort [options] filename |
|  | |  |  | | --- | --- | | -b | Ignores spaces at beginning of the line | | -c | Check whether input is sorted; do not sort | | -r | Sorts in reverse order | | -u | If line is duplicated only display once | | -r | Reverse the result of comparisons. | |
|  | **Example:-** |
|  | **$ cat > data.txt**  apples  oranges  pears  kiwis  bananas  **$ sort data.txt**  apples  bananas  kiwis  oranges  pears   * Note that this command does not actually change the input file, data.txt. If you want to write the output to a new file, output.txt, redirect the output like this:   **$ sort data.txt > output.txt**   * it will not display any output, but will create the file output.txt with the same sorted data from the previous command. To check the output, use the cat command:   **$ cat output.txt**  apples  bananas  kiwis  oranges  pears   * You can also use the built-in sort option -o, which allows you to specify an output file:   **$ sort -o output.txt data.txt**   * Using the -o option is functionally the same as redirecting the output to a file; neither one has an advantage over the other.   **Sorting In Reverse Order :**   * You can perform a reverse-order sort using the -r flag. For example, the following command:   **$ sort -r data.txt**  pears  oranges  kiwis  bananas  apples  **Checking For Sorted Order**   * If you just want to check to see if your input file is already sorted, use the -c option:   **$ sort -c data.txt**   * If your data is unsorted, you will receive an informational message reporting the line number of the first unsorted data, and what the unsorted data is. |
|  |  |
| **40)** | **find**:- Finds one or more files assuming that you know their approximate path. |
|  | **Syntax** :- find [options] path |
|  | **Description** :-   * Find is one of the powerful utility of Unix (or Linux) used for searching the files in a directory hierarchy |
|  | |  |  | | --- | --- | | path | A path name of a starting point in the directory hierarchy | | -maxdepth | Descend at most levels (a non-negative integer) levels of directories below the command line arguments. | | -i | ignore the case in the current directory and sub-directories. | | -size | Find file based on size | |
|  | **Example:-**  **$ find -name "sum.java"**  ./bkp/sum.java  ./sum.java  **$ find -iname "sum.java"**  ./SUM.java  ./bkp/sum.java  ./sum.java  **$ find -maxdepth 1 -name "sum.java"**  ./sum.java   * Find Files Under Home Directory * Find all the files under /home directory with name myfile.txt.   **$ find /home -name myfile.txt**  /home/myfile.txt  **$ find . -size 10M**   * Display files whose size is exactly 10M   **$ find . -size +10M**   * Display files larger than 10M size   **$ find -newer "sum.java"**   * Display the files which are modified after the modification of a give file.   **$ find . -perm 777**   * display the files based on the file permissions. * This will display the files which have read, write, and execute permissions. To know the permissions of files and directories use the command **"ls -l".**   **$ find -name '\*' -size +1000k**   * The system would search for any file in the list that is larger than 1000k. |
|  |  |
| **41)** | **uniq**:- Report or filter out repeated lines in a file. |
|  | **Syntax**:- uniq [option] filename |
|  | **Description : -** |
|  | |  |  | | --- | --- | | -c | Precede each output line with a count of the number of times the line occurred in the input | | -d | Suppress the writing of lines that are not repeated in the input | | -D | Print all duplicate lines | | -f | Avoid comparing first N fields | | -i | Ignore case when comparing | | -s | Avoid comparing first N characters. | | -u | Prints only unique lines | | -w | Compare no more than N characters in lines | |
|  | **Example:-**  **$ cat example.txt**  Unix operating system  unix operating system  unix dedicated server  linux dedicated server   * The default behaviour of the uniq command is to suppress the duplicate line. Note that, you have to pass sorted input to the uniq, as it compares only successive lines.   **$ uniq example.txt**  unix operating system  unix dedicated server  linux dedicated server   * The -c option is used to find how many times each line occurs in the file. It prefixes each line with the count.   **$ uniq -c example.txt**  2 unix operating system  1 unix dedicated server  1 linux dedicated server   * You can print only the lines that occur more than once in a file using the -d option.   **$ uniq -d example.txt**  unix operating system   * The -D option prints all the duplicate lines.   **$ uniq -D example.txt**  unix operating system  unix operating system |
|  |  |
| **42)** | **tr**:- Translate characters. |
|  | **Syntax**:- tr [options] set1 [set2] |
|  | **Description**:-   |  |  | | --- | --- | | -c | Complement the set of characters specified by string1 | | -d | Delete all occurrences of input characters that are specified by string1 | | -s | replaces repeated characters listed in the string1 with single occurrence | | Set1 | First string or character to be changed | | set2 | Second string or character to change the string1 | |
|  | **Example:-**   * Convert lower case letters to upper case * The following tr command translates the lower case letters to capital letters in the give string:   **$ echo "linux dedicated server" | tr "[:lower:]" "[:upper:]"**  LINUX DEDICATED SERVER  **$ echo "linux dedicated server" | tr "[a-z]" "[A-Z]"**  LINUX DEDICATED SERVER   * The -c option is used to replace the non-matching characters with another set of characters.   **$ echo "unix" | tr -c "u" "a"**  uaaa   * You can squeeze more than one occurrence of continuous characters with single occurrence. The following example squeezes two or more successive blank spaces into a single space.   **$ echo "linux server" | tr -s " "**  linux server   * The following example removes the word linux from the string.   **$ echo "linuxserver" | tr -d "linux"**  Server |
|  |  |
| **43)** | **history**:- history command is used to list out the recently executed commands in the number line order. |
|  | **Syntax**:- history [options] |
|  | **Description**:-   * The history command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an ``event''.  |  |  | | --- | --- | | -c | clear the history list by deleting all of the entries. | |
|  | **Example :-**   * To list the recently executed commands:   **$ history**   * This command is used to list the history.   **To find specific command in history list:**  **$ history | grep cd**  33 cd Pictures/  37 cd ..  39 cd Desktop/  61 cd /usr/bin/  68 cd   * This command is used to list only cd commands from history list.   **To copy history list to file**  **$ history -a history.txt**   * This command is used to copy history list to history.txt file .   **To Clear all history:**  **$ history -c**   * This command can be used to clear all history from history list. |
| **44)** | **write:-** Send a message to another user. |
|  | **Syntax**:- write person [ttyname] |
|  | **Description**:-   * The write utility allows you to communicate with other users, by copying lines from your terminal to theirs. * When you run the write command, the user you are writing to gets a message of the format:   Message from yourname@yourhost on yourtty at hh:mm ...   * Any further lines you enter will be copied to the specified user's terminal. If the other user wants to reply, they must run write as well. * When you are done, type an end-of-file or interrupt character. The other user will see the message ‘EOF’ indicating that the conversation is over. |
|  | |  |  | | --- | --- | | person | If you wish to talk to someone on your own machine, then person is just the person's login name. If you wish to talk to a user on another host, then person is of the form 'user@host'. | | ttyname | If you wish to talk to a user who is logged in more than once, the ttyname argument may be used to indicate the appropriate terminal name, where ttyname is of the form 'ttyXX' or 'pts/X'. | |
|  |  |
| **45)** | **grep**:- It selects and prints the lines from a file which matches a given string or pattern. |
|  | **Syntax**:- grep [options] pattern [file] |
|  | **Description:-** |
|  | * This command searches the specified input fully for a match with the supplied pattern and displays it. * While forming the patterns to be searched we can use shell match characters, or regular expressions. * Let us begin with the simplest example of usage of grep.  |  |  | | --- | --- | | -i | Ignore case distinctions | | -v | Invert the sense of matching, to select non-matching lines. | | -w | Select only those lines containing matches that form whole words. | | -x | Select only matches that exactly match the whole line. | | -c | print a count of matching lines for each input file. | |
|  | **Example :-** |
|  | **$ grep "Error" logfile.txt**   * This searches for the string "Error" in the log file and prints all the lines that has the word "Error".   **$ grep "string" file1 file2**   * Searching for a string in multiple files.   **$ grep -i "UNix" file.txt**   * The -i option enables to search for a string case insensitively in the give file. It matches the words like "UNIX", "Unix", "unix".   **$ grep -w "world" file.txt**   * By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.   **$ grep -c "sting" file.txt**   * We can find the number of lines that matches the given string/pattern   **$ grep -l "string" \***   * We can just display the files that contain the given string/pattern.   **$ grep -n "string" file.txt**   * We can make the grep command to display the position of the line which contains the matched string in a file using the -n option |
|  |  |
| **46)** | **pwd:-**Displaying your current directory name (Print working directory). |
|  | **Syntax:-**pwd [options] |
|  | **Description:-**   * At the time of logging in user is placed in the specific directory of the file system. You can move around from one directory to another, but any point of time, you are located in only one directory. This directory is known as your current directory. pwd command tells your current directory. |
|  | **Example:-**  **$ pwd**  /home/abc |
|  |  |
| **47)** | **wc:-** Word Count (wc) command counts and displays the number of lines, words, character and number of bytes enclosed in a file. |
|  | **Syntax: -** wc [*options*] [filename] |
|  | **Description:-**   * This command counts lines, words and characters depending on the options used. It takes one or more filenames as its arguments and displays four-columnar output. For example let’s read our os1 file. And we use wc command with that filename.  |  |  | | --- | --- | | -l | print the newline counts. | | -w | print the word counts. | | -c | print the byte counts. | | -m | print the character counts. | | -L | print the length of the longest line. | |
|  | **Example :-**  **$cat myfile.txt**  Red Hat  CentOS  Fedora  Debian  Scientific Linux  OpenSuse  Ubuntu  Xubuntu  Linux Mint  Pearl Linux  Slackware  Mandriva   * The ‘wc‘ command without passing any parameter will display a basic result of ”myfile.txt‘ file. The three numbers shown below are 12 (number of lines), 16 (number of words) and 112 (number of bytes) of the file.   **$wc myfile.txt**  12 16 112 myfile.txt   * To count number of newlines in a file use the option ‘-l‘, which prints the number of lines from a given file. Say, the following command will display the count of newlines in a file. In the output the first filed assigned as count and second field is the name of file.   **$wc -l myfile.txt**  12 myfile.txt   * Using ‘-w‘ argument with ‘wc‘ command prints the number of words in a file. Type the following command to count the words in a file.   **$wc -w myfile.txt**  16 myfile.txt   * When using options ‘-c‘ and ‘-m‘ with ‘wc‘ command will print the total number of bytes and characters respectively in a file.   **$wc -c myfile.txt**  112 myfile.txt   * The ‘wc‘command allow an argument ‘-L‘, it can be used to print out the length of longest (number of characters) line in a file. So, we have the longest character line (‘Scientific Linux‘) in a file.   **$wc -L myfile.txt**  16 myfile.txt |
|  |  |
| **48)** | **man**:- man command which is short for manual, provides in depth information about the requested command (or) allows users to search for commands related to a particular keyword. |
|  | **Syntax**:- man commandname [options] |
|  | **Description :-**   |  |  | | --- | --- | | -a | Print a one-line help message and exit. | | -k | Searches for keywords in all of the manuals available. | |
|  | **Example:-**  **$ man mkdir**   * Display the information about mkdir command |
| **49)** | **| (Pipeline command)**:- Used to combine more than one command. Takes output of 1st command as input of 2nd command. |
|  | **Syntax:- commmand1| command2|……** |
|  | **Example:-** |
|  | **$ls -l | grep "Feb"**  -rw-r--r-- 1 dietstaffdietstaff 336 Feb 19 14:41 calc1.sh  -rw-r--r—1 dietstaffdietstaff 410 Feb 19 14:28 calc.sh  drwxr-xr-x 7 dietstaffdietstaff 4096 Feb 24 09:04 Desktop  drwxr-xr-x 7 dietstaffdietstaff 12288Feb 26 07:44 Downloads  drwxr-xr-x 3 dietstaffdietstaff 4096 Feb 26 08:55 lab  drwxr-xr-x 2 dietstaffdietstaff 4096 Feb 24 08:00 shellscript  -rw-r--r—1 dietstaffdietstaff 81 Feb 24 12:34 temp3.sh |
|  | **Explain the basic system administration commands of unix** |
| **1)** | **date** : - Prints or sets the date and time. |
|  | **Syntax** :- date[options] [+format] [date] |
|  | **Description** :-   * Display the current date with current time, time zone. * The command can also be used with suitable format specifies as arguments. Each format is preceded by a + symbol, followed by the % operator, and a single character describing the format. |
|  | **Format**   |  |  | | --- | --- | | %a | Abbreviated weekday(Tue). | | %A | Full weekday(Tuesday). | | %b | Abbreviated month name(Jan). | | %B | Full month name(January). | | %c | Country-specific date and time format.. | | %D | Date in the format %m/%d/%y. | | %j | Julian day of year (001-366). | | %p | String to indicate a.m. or p.m. | | %T | Time in the format %H:%M:%S. | | %t | Tab space. | | %V | Week number in year (01-52); start week on Monday. | |
|  | **Example:-** |
|  | **$ date**  Fri Feb 19 09:55:15 IST 2016  **$ date +%m**  02  **$ date +%h**  Feb  **$ date +%y**  16  **$ date +"Date is %D %t Time is %T"**  date is 19/02/16 Time is 10:52:34   * To know the week number of the year,   **$ date -V**  11   * To set the date,   **$ date -s "10/08/2016 11:37:23"**   * The above command will print Wed Oct 08 11:37:23 IST 2016 |
| **2)** | **wall :-** send a message to everybody's terminal. |
|  | **Syntax** :- wall [ message ] |
|  | * Wall sends a message to everybody logged in with their mesg(1) permission set to yes. The message can be given as an argument to *wall*, or it can be sent to *wall*'s standard input. When using the standard input from a terminal, the message should be terminated with the EOF key (usually Control-D). * The length of the message is limited to 20 lines. |
|  | **Example : -**  **$sudo wall message.txt**   * Using the sudo command to run wall as the superuser, sends the contents ofmessage.txt to all users. |
|  | **ulimit :** Setting limits on file size |
|  | **Syntax: -** ulimit [limit] |
|  | **Description** :-   * The ulimit command imposes a restriction on the maximum size of the file that a user is permitted to create , so that there is less wastage of disk space. |
| **3)** | **passwd**:- It is used to change your password. |
|  | **Syntax**:- passwd |
|  | **Description** :-   * Passwd changes the password or shell associated with the user given by name or the current user if name is omitted. * First user has to insert current password. Then new password will be asked followed by confirm new password field. * passwd command can also be used to change the home directory where the path stands for the home directory. |
| **4)** | **suid : -** set user id. |
|  | **Description :-**   * suid (Set owner User ID up on execution) is a special type of file permissions given to a file. * Normally in Linux/Unix when a program runs, it inherits access permissions from the logged in user. * suid is defined as giving temporary permissions to a user to run a program/file with the permissions of the file owner rather that the user who runs it. * In simple words users will get file owner’s permissions as well as owner UID and GID when executing a file/program/command. |
|  | **Example:-**  **$ chmod u+s a.out**  **$ ls –l a.out**  -rw**s**r-xr-rw 1 dietstaff dietstaff 121 Feb 21 11:18 a.out |

SHELL SCRIPTS

|  |  |
| --- | --- |
|  |  |
| **1)** | **Write a shell script to scans the name of the command and executes it.** |
|  | **Program :-** |
|  | echo "enter command name"  read cmd  $cmd |
|  | **Output :-** |
|  | enter command name  cal  February 2016  Su Mo Tu We Th Fr Sa  1 2 3 4 5 6  7 8 9 10 11 12 13  14 15 16 17 18 19 20  21 22 23 24 25 26 27  28 29 |
| **2)** | **Write a shell script Which works like calculator and performs below operations Addition , Subtract ,Division ,Multiplication** |
|  | **Program :-** |
|  | **i) using if..elif statement** |
|  | echo " Enter one no."  read n1  echo "Enter second no."  read n2  echo "1.Addition"  echo "2.Subtraction"  echo "3.Multiplication"  echo "4.Division"  echo "Enter your choice"  read ch  if [ $ch = "1" ]  then  sum=`expr $n1 + $n2`  echo "Sum ="$sum  elif [ $ch = "2" ]  then  sum=`expr $n1 - $n2`  echo "Sub = "$sum  elif [ $ch = "3" ]  then  sum=`expr $n1 \\* $n2`  echo "Mul = "$sum  elif [ $ch = "4" ]  then  sum=`expr $n1 / $n2`  echo "Div = "$sum  fi |
|  | **Output :-** |
|  | Enter one no.  32  Enter second no.  12  1.Addition  2.Subtraction  3.Multiplication  4.Division  Enter your choice  2  Sub = 20 |
|  | **ii) using while loop and switch statement** |
|  | i="y"  while [ $i = "y" ]  do  echo " Enter one no."  read n1  echo "Enter second no."  read n2  echo "1.Addition"  echo "2.Subtraction"  echo "3.Multiplication"  echo "4.Division"  echo "Enter your choice"  read ch  case $ch in  1)sum=`expr $n1 + $n2`  echo "Sum ="$sum;;  2)sum=`expr $n1 - $n2`  echo "Sub = "$sum;;  3)sum=`expr $n1 \\* $n2`  echo "Mul = "$sum;;  4)sum=`expr $n1 / $n2`  echo "Div = "$sum;;  \*)echo "Invalid choice";;  esac  echo "Do u want to continue ? y/n"  read i  if [ $i != "y" ]  then  exit  fi  done |
|  | **Output :-** |
|  | Enter one no.  32  Enter second no.  22  1.Addition  2.Subtraction  3.Multiplication  4.Division  Enter your choice  2  Sub = 10  Do u want to continue ? y/n  N |
| **3)** | **Write a shell script to print the pyramid structure for the given number.** |
|  | **Program :-** |
|  | echo "enter the number"  read n  printf "\n"  for((i=1;i<=$n;i++))  do  for((j=1;j<=$i;j++))  do  printf "$j"  done  printf "\n"  done |
|  | **Output :-** |
|  | enter the number  4  1  12  123  1234 |
| **4)** | **Write a shell script to find the largest among the 3 given numbers.** |
|  | **Program :-** |
|  | clear  echo "Enter first number: "  read a  echo "Enter second number: "  read b  echo "Enter third number: "  read c  if [ $a -ge $b -a $a -ge $c ]  then  echo "$a is largest integer"  elif [ $b -ge $a -a $b -ge $c ]  then  echo "$b is largest integer"  elif [ $c -ge $a -a $c -ge $b ]  then  echo "$c is largest integer"  fi |
|  | **Output :-** |
|  | Enter first number:  22  Enter second number:  33  Enter third number:  42  44 is largest integer |
| **5)** | **Write a shell script to find factorial of given number n.** |
|  | **Program :-** |
|  | clear  fact=1  echo "Enter number to find factorial : "  read n  a=$n  #if enter value less than 0  if [ $n -le 0 ]  then  echo "invalid number"  exit  fi  #factorial logic  while [ $n -ge 1 ]  do  fact=`expr $fact \\* $n`  n=`expr $n - 1`  done  echo "Factorial for $a is $fact" |
|  | **Output :-** |
|  | Enter number to find factorial :  5  Factorial for 5 is 120 |
| **6)** | **Write a shell script to print all prime numbers from 1 to n.** |
|  | **Program :-**  clear |
|  | echo "enter the range"  read n  echo "the prime no are:"  m=2  while [ $m -le $n ]  do  i=2  flag=0  while [ $i -le `expr $m / 2` ]  do  if [ `expr $m % $i` -eq 0 ]  then  flag=1  break  fi  i=`expr $i + 1`  done  if [ $flag -eq 0 ]  then  echo $m  fi  m=`expr $m + 1`  done |
|  | **Output :-** |
|  | enter the range  10  the prime no are  2  3  5  7 |
| **7)** | **Write a shell script toreverse a number supplied by a user.** |
|  | **Program :-** |
|  | if [ $# -eq 1 ]  then  if [ $1 -gt 0 ]  then  num=$1  sumi=0  while [ $num -ne 0 ]  do  lnum=`expr $num % 10`  sumi=`expr $sumi \\* 10 + $lnum`  num=`expr $num / 10`  done  echo "Reverse of digits is $sumi of $1"  else  echo " Number is less than 0"  fi  else  echo "Insert only one parameter "  fi |
|  | **Output :-** |
|  | bash pr81.sh 123  Reverse of digits is 321 of 123 |
| **8)** | **Write a shell script to find first n Fibonacci numbers like: 0 1, 1, 2, 3, 5, 13,…** |
|  | **Program :-** |
|  | clear  echo "How many number of terms to be generated ?"  read n  x=0  y=1  i=2  echo "Fibonacci Series up to $n terms :"  echo "$x"  echo "$y"  while [ $i -lt $n ]  do  i=`expr $i + 1 `  z=`expr $x + $y `  echo "$z"  x=$y  y=$z  done |
|  | **Output :-** |
|  | How many number of terms to be generated?  5  Fibonacci Series up to 5 terms :  0  1  1  2  3 |
| **9)** | **Write a shell script to check whether the given number is Perfect or not.** |
|  | **Program :-** |
|  | echo Enter a number  read no  i=1  ans=0  while [ $i -le `expr $no / 2` ]  do  if [ `expr $no % $i` -eq 0 ]  then  ans=`expr $ans + $i`  fi  i=`expr $i + 1`  done  if [ $no -eq $ans ]  then  echo $no is perfect  else  echo $no is NOT perfect  fi |
|  | **Output :-** |
|  | Enter a number  6  6 is perfect  Enter a number  10  10 is NOT perfect |
| **10)** | **Write a shell script which displays a list of all files in the current directory to which you have read, write and execute permissions** |
|  | **Program :-** |
|  | for File in \*  do  if [ -r $File -a -w $File -a -x $File ]  then  echo $File  fi  done |
|  | **Output :-** |
|  | Desktop  Documents  Downloads  lab  Music  Pictures  Public  shellscript  Templates  Videos |
| **11)** | **Write a shell script that deletes all the files in the current directory which are 0 bytes in length.** |
|  | **Program :-** |
|  | clear  find . -name "\*" -size -1k –delete  echo “files deleted” |
|  | **Output :-** |
|  | files deleted |
| **12)** | **Write a shell script to check whether the given string is Palindrome or not.** |
|  | **Program :-** |
|  | clear  echo "Enter the string:"  read str  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 the string:  abba  String is palindrome  Enter the string:  abc  String is not palindrome |
| **13)** | **Write a shell script to display the digits which are in odd position in a given 5 digit number** |
|  | **Program :-** |
|  | echo "Enter a 5 digit number"  read num  n=1  while [ $n -le 5 ]  do  a=`echo $num | cut -c $n`  echo $a  n=`expr $n + 2`  done |
|  | **Output :-** |
|  | Enter a 5 digit number  12345  1  3  5 |
| **14)** | **Write a shell script tocheck given year is leap year or not.** |
|  | **Program :-** |
|  | clear  echo "enter any year"  read num  if [ `expr $num % 4` -eq 0 ]  then  if [ `expr $num % 100` -eq 0 -a `expr $num % 400` -ne 0 ]  then  echo "Not a leap year"  else  echo "Leap year "  fi  else  echo "Not a leap year"  fi |
|  | **Output :-** |
|  | enter any year  2016  Leap year  enter any year  2100  Not a leap year |
| **15)** | **Write a shell script tofind the value of one number raised to the power of another.** |
|  | **Program :-** |
|  | echo "Input number"  read no  echo "Input power"  read power  counter=0  ans=1  while [ $power -ne $counter ]  do  ans=`echo $ans \\* $no | bc`  counter=`echo $counter + 1 | bc`  done  echo "$no power of $power is $ans" |
|  | **Output :-** |
|  | Input number  5  Input power  3  5 power of 3 is 125 |
| **16)** | **Write a shell script to display the following details in a pay list Pay slip details, House rent allowance, Dearness allowance, Provident fund. HRA is to be calculated at the rate of 20% of basic, DA at the rate of 40% of basic and PF at the rate of 10% of basic.** |
|  | **Program :-** |
|  | i="y"  while [ $i = "y" ]  do  echo "Please enter your Basic:"  read basic  echo "PAY SLIP DETAILS"  echo "1. HOUSE RENT ALLOWANCE"  echo "2. DEARNESS ALLOWANCE"  echo "3. PROVIDENT FUND"  echo "your choice:"  read ch  case $ch in  1) hra=`expr $basic \\* 20 / 100`  echo Your HOUSE RENT ALLOWANCE is Rs. $hra;;  2) da=`expr $basic \\* 40 / 100`  echo Your DEARNESS ALLOWANCE is Rs. $da;;  3) pf=`expr $basic \\* 10 / 100`  echo Your PPOVIDENT FUND is Rs. $pf;;  \*) echo "Not a valid choice";;  esac  echo "Do u want to continue ?"  read i  if [ $i != "y" ]  then  exit  fi  done |
|  | **Output :-** |
|  | Please enter your Basic:  1000  PAY SLIP DETAILS  1. HOUSE RENT ALLOWANCE  2. DEARNESS ALLOWANCE  3. PROVIDENT FUND  your choice:  1  Your HOUSE RENT ALLOWANCE is Rs. 200  Do u want to continue ? |
|  |  |
| **17)** | **Write a shell script to find sum of digits of a number.** |
|  | **Program :-** |
|  | clear  echo "enter the number"  read n  sum=0  a=$n  while(($n >0))  do  x=`expr $n % 10`  sum=`expr $sum + $x`  n=`expr $n / 10`  done  echo "the sum of $a is $sum" |
|  | **Output :-** |
|  | enter the number  3355  the sum of 3355 is 16 |
| **18)** | **Write a shell script that greets the user by saying Good Morning, Good Afternoon, and Good Evening according to the system time.** |
|  | **Program :-** |
|  | clear  #hours=`date|cut -c 12-13`  hours=`date +%H`  if [ $hours -le 12 ]  then  echo "Good Morning"  elif [ $hours -le 16 ]  then  echo "Good Afternoon"  elif [ $hours -le 20 ]  then  echo "Good Evening"  else  echo "Good Night"  fi |
|  | **Output :-** |
|  | Good Afternoon |
| **19)** | **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.** |
|  | **Program :-** |
|  | Clear  echo "Enter the five subject marks for the student"  read s1 s2 s3  sum1=`expr $s1 + $s2 + $s3`  echo "Sum of 5 subjects are: " $sum1  per=`expr $sum1 / 3`  echo " Percentage: " $per  if [ $per -ge 60 ]  then  echo "You get Distinction"  elif [ $per -ge 50 ]  then  echo â€œYou get First classâ€  elif [ $per -ge 40 ]  then  echo "You get Second class"  else  echo "You get Fail"  fi |
|  | **Output :-** |
|  | Enter the five subject marks for the student  78 88 92  Sum of 5 subjects are: 258  Percentage: 86  You get Distinction |
| **20)** | **Write a shell script that finds total no. of users and finds out how many of them are currently logged in.** |
|  | **Program :-** |
|  | cat /etc/passwd>user.txt  set `wc -l user.txt`  log=`who|wc -l`  echo "There are $1 users in network "  echo "There are $log user loged in right now" |
|  | **Output :-** |
|  | There are 49 users in network  There are 2 user loged in right now |