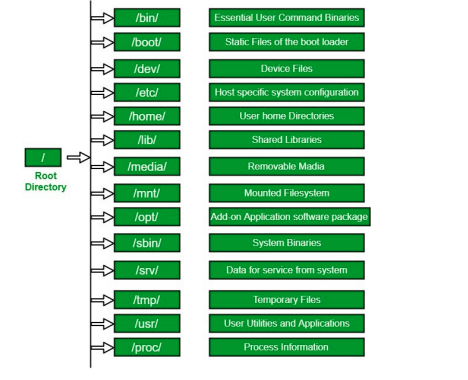
**1 . Write short note on linux file system hierarchy**

**The Linux File Hierarchy Structure or the Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix-like operating systems.It is maintained by the Linux Foundation.**

**● In the FHS, all files and directories appear under the root directory /, even if they are stored on different physical or virtual devices.**

**● Some of these directories only exist on a particular system if certain subsystems, such as the X Window System, are installed.**

**● Most of these directories exist in all UNIX operating systems and are generally used in much the same way; however, the descriptions here are those used specifically for the FHS, and are not considered authoritative for platforms other than Linux.**

**– The Root Directory**

**Everything on your Linux system is located under the / directory, known as the root directory. You can think of the / directory as being similar to the C:\ directory on Windows – but this isn’t strictly true, as Linux doesn’t have drive letters. While another partition would be located at D:\ on Windows, this other partition would appear in another folder under / on Linux.**

**/bin : Essential command binaries that need to be available in single user mode; for all users, e.g., cat, ls, cp.**

**● Contains binary executables**

**● Common linux commands you need to use in single-user modes are located under this directory. ● Commands used by all the users of the system are located here e.g. ps, ls, ping, grep, cp**

**The /bin directory contains the essential user binaries (programs) that must be present when the system is mounted in single-user mode. Applications such as Firefox are stored in /usr/bin, while important system programs and utilities such as the bash shell are located in /bin. The /usr directory may be stored on another partition – placing these files in the /bin directory ensures the system will have these important utilities even if no other file systems are mounted. The /sbin directory is similar – it contains essential system administration binaries.**

**/boot – Static Boot Files**

**The /boot directory contains the files needed to boot the system – for example, the GRUB boot loader’s files and your Linux kernels are stored here. The boot loader’s configuration files aren’t located here, though – they’re in /etc with the other configuration files.**

**/cdrom – Historical Mount Point for CD-ROMs**

**The /cdrom directory isn’t part of the FHS standard, but you’ll still find it on Ubuntu and other operating systems. It’s a temporary location for CD-ROMs inserted in the system. However, the standard location for temporary media is inside the /media directory.**

**/dev – Device Files**

**Linux exposes devices as files, and the /dev directory contains a number of special files that represent devices. These are not actual files as we know them, but they appear as files – for example, /dev/sda represents the first SATA drive in the system. If you wanted to partition it, you could start a partition editor and tell it to edit /dev/sda.**

**This directory also contains pseudo-devices, which are virtual devices that don’t actually correspond to hardware. For example, /dev/random produces random numbers. /dev/null is a special device that produces no output and automatically discards all input – when you pipe the output of a command to /dev/null, you discard it.**

**/etc – Configuration Files**

**The /etc directory contains configuration files, which can generally be edited by hand in a text editor. Note that the /etc/ directory contains system-wide configuration files – user-specific configuration files are located in each user’s home directory.**

**/home – Home Folders**

**The /home directory contains a home folder for each user. For example, if your user name is bob, you have a home folder located at /home/bob. This home folder contains the user’s data files and user-specific configuration files. Each user only has write access to their own home folder and must obtain elevated permissions (become the root user) to modify other files on the system.**

**/lib – Essential Shared Libraries**

**The /lib directory contains libraries needed by the essential binaries in the /bin and /sbin folder. Libraries needed by the binaries in the /usr/bin folder are located in /usr/lib.**

**/lost+found – Recovered Files**

**Each Linux file system has a lost+found directory. If the file system crashes, a file system check will be performed at next boot. Any corrupted files found will be placed in the lost+found directory, so you can attempt to recover as much data as possible.**

**/media – Removable Media**

**The /media directory contains subdirectories where removable media devices inserted into the computer are mounted. For example, when you insert a CD into your Linux system, a directory will automatically be created inside the /media directory. You can access the contents of the CD inside this directory.**

**/mnt – Temporary Mount Points**

**Historically speaking, the /mnt directory is where system administrators mounted temporary file systems while using them. For example, if you’re mounting a Windows partition to perform some file recovery operations, you might mount it at /mnt/windows. However, you can mount other file systems anywhere on the system.**

**/opt – Optional Packages**

**The /opt directory contains subdirectories for optional software packages. It’s commonly used by proprietary software that doesn’t obey the standard file system hierarchy – for example, a proprietary program might dump its files in /opt/application when you install it.**

**/proc – Kernel & Process Files**

**The /proc directory similar to the /dev directory because it doesn’t contain standard files. It contains special files that represent system and process information.**

**/root – Root Home Directory**

**The /root directory is the home directory of the root user. Instead of being located at /home/root, it’s located at /root. This is distinct from /, which is the system root directory.**

**/run – Application State Files**

**The /run directory is fairly new, and gives applications a standard place to store transient files they require like sockets and process IDs. These files can’t be stored in /tmp because files in /tmp may be deleted.**

**/sbin – System Administration Binaries**

**The /sbin directory is similar to the /bin directory. It contains essential binaries that are generally intended to be run by the root user for system administration**

**/selinux – SELinux Virtual File System**

**If your Linux distribution uses SELinux for security (Fedora and Red Hat, for example), the /selinux directory contains special files used by SELinux. It’s similar to /proc. Ubuntu doesn’t use SELinux, so the presence of this folder on Ubuntu appears to be a bug.**

**/srv – Service Data**

**The /srv directory contains “data for services provided by the system.” If you were using the Apache HTTP server to serve a website, you’d likely store your website’s files in a directory inside the /srv directory.**

**/tmp – Temporary Files**

**Applications store temporary files in the /tmp directory. These files are generally deleted whenever your system is restarted and may be deleted at any time by utilities such as tmpwatch.**

**/usr – User Binaries & Read-Only Data**

**The /usr directory contains applications and files used by users, as opposed to applications and files used by the system. For example, non-essential applications are located inside the /usr/bin directory instead of the /bin directory and non-essential system administration binaries are located in the /usr/sbin directory instead of the /sbin directory. Libraries for each are located inside the /usr/lib directory. The /usr directory also contains other directories – for example, architecture-independent files like graphics are located in /usr/share.The /usr/local directory is where locally compiled applications install to by default – this prevents them from mucking up the rest of the system.**

**/var – Variable Data Files**

**The /var directory is the writable counterpart to the /usr directory, which must be read-only in normal operation. Log files and everything else that would normally be written to /usr during normal operation are written to the /var directory. For example, you’ll find log files in /var/log.**

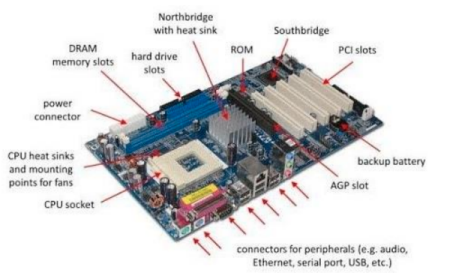
**2. Write short note on motherboard,Ram,Daughter cards,bus slotted,SMPS,internal storage devices,interfacing port**

**Physical identification of major components of a computer system such as mother board, RAM modules, daughter cards, bus slots, SMPS, internal storage devices, interfacing ports. Specifications of desktop and server class computers.**

**Installation of common operating systems for desktop and server use. (Students**

**may be asked to formulate specification for computer to be used as Desktop, Web server)**

**MOTHERBOARD**

****

**A motherboard (also called mainboard, main circuit board, or mobo) is the main printed circuit board (PCB) in general-purpose computers and other expandable systems. It holds and allows communication between many of the crucial electronic components of a system, such as the central processing unit (CPU) and memory, and provides connectors for other peripherals. Unlike a backplane, a motherboard usually contains significant sub-systems, such as the central processor, the chipset's input/output and memory controllers, interface connectors, and other components integrated for general use.**

**Motherboard means specifically a PCB with expansion capabilities. As the name suggests, this board is often referred to as the "mother" of all components attached to it, which often include peripherals, interface cards, and daughterboards: sound cards, video cards, network cards, host bus adapters, TV tuner cards, IEEE 1394 cards; and a variety of other custom components.**

**RAM modules**

****

**In computing, a memory module or RAM (random-access memory) stick is a printed circuit board on which memory integrated circuits are mounted. Memory modules permit easy installation and replacement in electronic systems, especially computers such as personal computers, workstations, and servers. The first memory modules were proprietary designs that were specific to a model of computer from a specific manufacturer. Later, memory modules were standardized by organizations such as JEDEC and could be used in any system designed to use them.**

**Types of memory module include:**

**• TransFlash Memory Module**

**• SIMM, a single in-line memory module**

**• DIMM, dual in-line memory module**

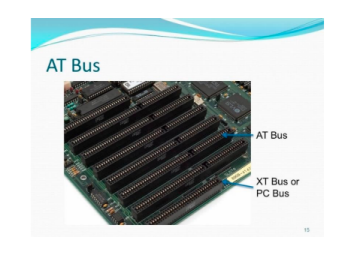
**➢ Rambus memory modules are a subset of DIMMs, but are normally referred to as RIMMs ➢ SO-DIMM, small outline DIMM, a smaller version of the DIMM, used in laptops Distinguishing characteristics of computer memory modules include voltage, capacity, speed (i.e., bit rate), and form factor. For economic reasons, the large (main) memories found in personal computers, workstations, and non-handheld game-consoles (such as PlayStation and Xbox) normally consist of dynamic RAM (DRAM). Other parts of the computer, such as cache memories normally use static RAM (SRAM). Small amounts of SRAM are sometimes used in the same package as DRAM.[2] However, since SRAM has high leakage power and low density, die-stacked DRAM has recently been used for designing multi-megabyte sized processor caches.**

**DAUGHTER BOARD**

****

**The daughter board is a computer hardware. It is also known as the piggyback board, riser card, daughter board, daughtercard or daughter card. A daughter board is a printed circuit board which is connected to the motherboard or expansion card. As compared to the motherboard, it is smaller in size. A daughter board does not act as an expansion card. An expansion card adds extra new functions to the computer. But a daughter board that is connected to the motherboard adds or supports the main functions of the motherboard. Daughter boards are directly connected to the motherboards. You know that expansion cards are connected to the motherboard by using the bus and other serial interfaces. But daughter board is directly connected to the board by soldering. As an update of the motherboard or expansion card, daughter boards are released to extend the features and services of the motherboard or expansion cards.**

**BUS SLOTS**

****

**Alternatively known as a bus slot or expansion port, an expansion slot is a connection or port inside a computer on the motherboard or riser card. It provides an installation point for a hardware expansion card to be connected, which provides additional features to a computer such as video, sound, advanced graphics, Ethernet or memory.**

**The expansion card has an edge connector that fits precisely into the expansion slot as well as a row of contacts that is designed to establish an electrical connection between the motherboard and the electronics on the card, which are mostly integrated circuits. Depending on the form factor of the case and motherboard, a computer system generally can have anywhere from one to seven expansion slots. With a backplane system, up to 19 expansion cards can be installed.**

**Expansion cards can provide various functions including:**

**SMPS**

****

**A switched-mode power supply (SMPS) is an electronic circuit that converts power using switching devices that are turned on and off at high frequencies, and storage components**

**such as inductors or capacitors to supply power when the switching device is in its non conduction state.**

**• Solid-state drive**

**• Power-on self-test**

**• Advanced multirate codec**

**• Basic input/output system**

**(BIOS)**

**• Expansion read-only memory**

**(ROM)**

**• Security devices**

**• RAM memory**

**• Sound**

**• Modems**

**• Network**

**• Interface adapters**

**• TV and radio tuning**

**• Video processing**

**• Host adapting such as redundant**

**array of independent disks or**

**small computer system interface**

**Switching power supplies have high efficiency and are widely used in a variety of electronic equipment, including computers and other sensitive equipment requiring stable and efficient power supply.**

**A switched-mode power supply is also known as a switch-mode power supply or switching-mode power supply.**

**Switched-mode power supplies are classified according to the type of input and output voltages. The four major categories are:**

**• AC to DC**

**• DC to DC**

**• DC to AC**

**• AC to AC**

**A basic isolated AC to DC switched-mode power supply consists of:**

**• Input rectifier and filter**

**• Inverter consisting of switching devices such as MOSFETs**

**• Transformer**

**• Output rectifier and filter**

**• Feedback and control circuit**

**INTERNAL STORAGE DEVICES**

****

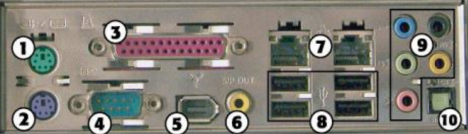
**A storage unit is a part of the computer system which is employed to store the information and instructions to be processed. A storage device is an integral part of the computer hardware which stores information/data to process the result of any computational work. Without a storage device, a computer would not be able to run or even boot up. Or in other words, we can say that a storage device is hardware that is used for storing, porting, or extracting data files. It can also store information/data both temporarily and permanently. Computer storage is of two types:**

**• Primary Storage Devices: It is also known as internal memory and main memory. This is**

**a section of the CPU that holds program instructions, input data, and intermediate results. It is generally smaller in size. RAM (Random Access Memory) and ROM (Read Only Memory) are examples of primary storage.**

**• Secondary Storage Devices: Secondary storage is a memory that is stored external to the computer. It is mainly used for the permanent and long-term storage of programs and data. Hard Disk, CD, DVD, Pen/Flash drive, SSD, etc, are examples of secondary storage.**

**INTERFACING PORTS**

****

**A port is basically a physical docking point which is basically used to connect the external devices to the computer, or we can say that A port act as an interface between the computer and the external devices, e.g., we can connect hard drives, printers to the computer with the help of ports.**

**Characteristics of Ports**

**A port has the following characteristics −**

**• External devices are connected to a computer using cables and ports.**

**• Ports are slots on the motherboard into which a cable of an external device is plugged in. • Examples of external devices attached via ports are the mouse, keyboard, monitor, microphone, speakers, etc.**

**Install LAMP in Ubuntu**

**Update your system**

**sudo apt update**

**Install Apache using apt:**

**sudo apt install apache2**

**Confirm that Apache is now running with the following command: sudo systemctl status apache2**

**if it is not working**

**sudo systemctl start apache2**

**Once installed, test by accessing your server’s IP in your browser:**

**http://youripaddress**

**( find out your ip address using iffconfig)**

**Install mariadb**

**(MariaDB is a fork of MySQL from some of the original MySQL team and is a drop-in replacement.)**

**sudo apt install mariadb-server mariadb-client**

**Check mariadb Installation**

**sudo systemctl status mysql**

**( if it is not working sudo systemctl start mysql )**

**Secure your newly installed MariaDB service:**

**sudo mysql\_secure\_installation**

**( This will set password for mariadb, and strengthen the security by asking some questions like disallow root login remotely? Remove test database? Etc)**

**Install PHP and commonly used modules**

**sudo apt install php libapache2-mod-php php-opcache php-cli php-gd php-curl php-mysql**

**Restart apache2**

**sudo systemctl restart apache2**

**Now you can check php installation**

**sudo echo “<?php phpinfo(); ?>” | sudo tee -a /var/www/html/phpinfo.php > /dev/null**

**Open a browser**

**http://127.0.0.1/phpinfo.php**

**Install phpmyadmin**

**sudo apt install phpmyadmin php-mbstring php-zip php-gd php-json php-curl ( It ask for webserver select apache2, select db-configuration and set password )**

**Restart apache2**

**sudo systemctl restart apache2**

**Check phpmyadmin**

**Open a browser**

**http://localhost/phpmyadmin**

**username : root**

**password : yourpassword**

**If any problem for login run the following command**

**sudo mysql**

**ALTER USER root@localhost IDENTIFIED BY “yourpassword”;**

1. **Practice Basic Shell Commands like:- ls, cd, du, pwd, man, cat, more, less, head, tail, mkdir, cp, mv, rm, touch, grep, sort, wc, cut, echo…**

**Pwd:This command is used to display the location of the current working**

**directory.**

**Syntax :-$ pwd**

**Output:**

**Mkdir:This command is used to create a new directory under any directory.**

**Syntax :-$ mkdir<directory name>**

**Output :-**

**ls:This command is used to display a list of content of directory.**

**Syntax :-$ ls**

**Output :-**

**Man:This command is used to display the user manual of any command that we can run on the terminal.**

**Syntax :-$ man <command name>**

**Output:**

**ls –l:This command is used to shows file or directory, size, modified date and time, file or folder name and owner of the file, and its permission.**

**Syntax :-$ ls –l**

**Output:-**

**ls –r:This command is used to display files and directories in reverse order.**

**Syntax :-$ls –r**

**Output :-**

**ls –a:This command is used to list all files including hidden files.**

**Syntax :- $ls –a**

**Output :-**

**ls –al**

**Syntax :-$ ls -al**

**Output :-**

**ls –t:This command is used to display files in the last modified order. Syntax :-$ ls –t**

**Output :-**

**Cd:This command is used to change the current directory. Syntax :-$ cd <directory name>**

**Output :-**

**cd.. :**

**This command is used to move to the parent directory of current directory, or the directory one level up from the current directory. Syntax :- $ cd ..**

**Output :-**

**cd –:This command is used to switch back to previous directory we were working earlier.**

**Syntax :-$ cd –**

**Output :-**

**cat > filename:This command is used to create a file and add contents to that file.**

**Syntax :-$ cat > filename.txt**

**cat filename:This command is used to view the contents in the file. Syntax :-$ cat filename.txt**

**Output :-**

**cat>>filename:This command is used to add contents to an existing file. Syntax :-$ cat >> filename.txt**

**Output :-**

**cat filename1 > filename2:This command is used to copy the content from one file to another file.**

**Syntax :-$ cat filename1 > filename2**

**Output :-**

**read :This command is used to read the content of a line to a variable. Syntax :-$ read variablename**

**Output :-**

**Find:This command is used to display contents of particular directory. Syntax :-$ find filename.txt**

**Output :-**

**grep :This command will let you search through all the text in a given file. Syntax :-$ grep word filename.txt**

**Output:-**

**grep -i :command used for a case insensitive search**

**Syntax: $ grep -i filename.txt**

**Output:**

**grep -v :command used for inverted search.**

**Syntax: $ grep -v filename.txt**

**Output:**

**grep -A1:command used to display line after the result.**

**Syntax: $ grep -A1 filename.txt**

**Output:**

**grep -B1:command used to display line before the result.**

**Syntax: $ grep -B1 filename.txt**

**Output:**

**grep -C1:command used to display line before and after the result. Syntax: $ grep -C1 filename.txt**

**Output:**

**wc -word count:This command is used for counting purpose which is used to find the number of lines,the number of words,the number of characters and the number of bytes.**

**wc -l (count number of lines)**

**wc -w (count number of words)**

**wc -c (count number of characters)**

**wc -m (count number of bytes)**

**Syntax :- $ wc -l filename.txt**

**$ wc -w filename.txt**

**$ wc -c filename.txt**

**$ wc -m filename.txt**

**Output :-**

**df :This command is used to get a report on system disc space usage. Syntax :-$ df filename.txt**

**Output :-**

**df -m :This command is used to see the report in mega bytes.**

**Syntax :$ def -m filename.txt**

**Output :-**

**cut -d:This command is used to cut and display the content based on the delimiter given.**

**Syntax :-$ cut –d delimiter –fieldnumber filename**

**Output :-**

**cut -b:This command is used tocut and display the content based on the specified byte number.**

**Syntax :-$ cut –b bytenumber filename**

**Output :-**

**cut --complement -c:This command is used to erase the specified character and display the remaining content of the file.**

**Syntax :-$ cut --complement –c characternumber filename.txt Output :-**

**Paste:This command is used to paste the contents from the specified file. Syntax :-$ paste filename**

**Output :-**

**paste file1 file2 > file3:This command is used to paste the contents from the specified files to another file.**

**Syntax :-$ paste file1 file2 > file3**

**Output:-**

**paste -s:This command is used to paste the contents sequentially. It reads all the lines from the file and merges all these lines into a single linewith each line separated by tab.**

**Syntax :-paste –s file1 file2**

**Output :-**

**paste -d:This command is used to paste the contents from the given files with the delimiter given.**

**Syntax :-$ paste –d delimiter file1 file2**

**Output :-**

**More:This command is used to view the text files in the command prompt, displaying one screen at a time in case the file is large.**

**Syntax :-$ more filename**

**Output :-**

**more -number:This command is used to display display the lines to the specified number from head.**

**Syntax :-$ more -number**

**Output :-**

**more +number:This command use the line number from where we want to displaying the text content.**

**Syntax :-$ more +number**

**Output :-**

**Cp:This command is used to copy the contents from an existing file to a new file.**

**Syntax :-$ cpexisting\_filenamenew\_filename**

**Output :-**

**cp overwriting:This method is used to overwrite the contents of an existing file from one directory to an existing file with the same name in another directory with the cp command.**

**Syntax :-$ cp filename directoryname**

**Output :-**

**cp-i:This command is used to ask the confirmation message once before overwriting the file. We give ‘y’ or ‘n’ as the response.**

**Syntax :- $ cp -i filename destination\_directory**

**Output :-**

**Mv:This command is used to move an existing file or directory from one location to another.**

**Syntax :-$ mv filename directory\_name**

**Output:-**

**mv overwriting:This method is used to overwrite the contents of an existing file from one directory to an existing file with the same name in another directory with the mv command.**

**Syntax :-$ mv filename directory\_name**

**Output :-**

**Head:This command is used to display the first 10 lines of the file by default. Syntax :-$ head filename**

**Output:-**

**head -number:This command is used to display the lines of the file to the specified number from head.**

**Syntax :-$ head -number filename**

**Output :-**

**Tail:This command is used to display the last 10 lines of the file by default. Syntax :-$ tail filename**

**Output :-**

**tail -number:This command is used to display the lines of the file to the specified number from tail.**

**Syntax :-$ tail -number filename**

**Output :-**

**sudo useradd :This command is used to add new user.**

**Syntax :-$ sudo useradd username**

**Output :-**

**sudo passwd :This command is used to add password to the user. Syntax :-$ sudo passwd username**

**Output :-**

**sudo usermod :This command is used to add members.**

**Syntax :-$sudo usermod -G groupname username**

**delete**

**sudo userdel username - used to delete user.**

**sudo groupdel groupname - used to delete group name.**

**Syntax :-$ sudo userdel username**

**$ sudo groupdel groupname**

**Output :-**

**chmod :This command is used change directory permission of files. chmod +rwx**

**chmod -wx**

**chmod -rwx**

**Syntax :- $ chmod +wx filename**

**$ chmod -wx filename**

**$ chmod -rwx filename**

**Output :-**

**chown:This command is used to give ownership to user . Syntax :- $ sudo chown username filename**

**Output :-**

**Ssh:This command is used to provide a secure encrypted connection between two hosts over an insecure network.**

**Syntax :- $ ssh mca@ipaddress**

1. **Write a Shell program to check the given number is even or odd.**

**Code:**

**echo "enter the number"**

**read a**

**if [ $((a%2)) == 0 ]; then**

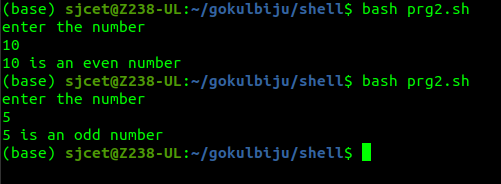
**echo $a" is an even number"**

**else**

**echo $a" is an odd number"**

**fi**

**Output**

****

1. **Write a Shell program to check a leap year.**

**Code:**

**echo "enter the year"**

**read a**

**if [[ $((a%400)) -eq 0 && $((a%100)) -ne 0 || $((a%4)) -eq 0 ]]; then**

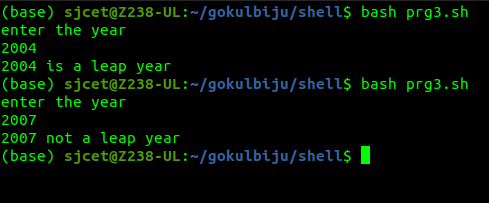
**echo $a "is a leap year"**

**else**

**echo $a "not a leap year"**

**fi**

**Output**

****

1. **Write a Shell program to find the area and circumference of a circle.**

**Code:**

**echo "enter the radious"**

**read r**

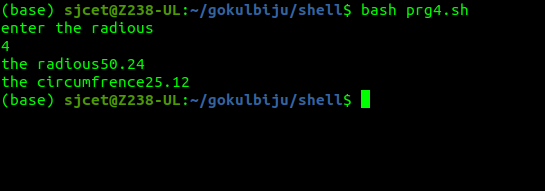
**area=$(echo "3.14\*$r\*$r" | bc )**

**circum=$(echo "3.14\*2\*$r" | bc)**

**echo "the radious"$area**

**echo "the circumfrence"$circum**

**Output**

****

1. **Write a Shell program to check the given number and its reverse are same.**

**Code:**

**echo "Enter a number"**

**read r**

**reverse=$(echo $r | rev )**

**echo "the reveres of the number "$reverse**

**if [ $r == $reverse ]; then**

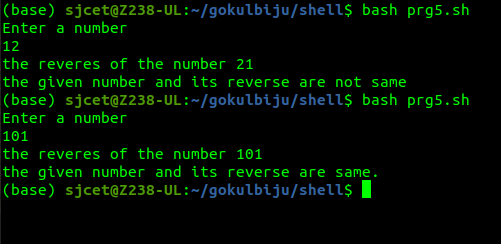
**echo "the given number and its reverse are same."**

**else**

**echo "the given number and its reverse are not same"**

**fi**

**Output**

****

1. **Write a Shell program to check the given string is palindrome or not.**

**Code:**

**echo "enter the string"**

**read r**

**reve=$(echo $r | rev )**

**if [ $r = $reve ]; then**

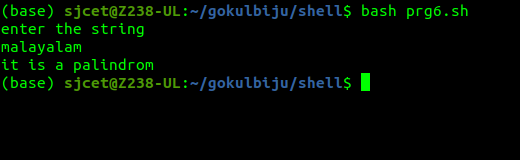
**echo "it is a palindrom"**

**else**

**echo "it is not a palindrom"**

**fi**

**Output**

****

1. **Write a Shell program to find the sum of odd and even numbers from a set of numbers.**

**Code:**

**echo "Enter a set of numbers (separated by spaces):"**

**read numbers**

**IFS=" " read -a array <<< "$numbers"**

**sum\_even=0**

**sum\_odd=0**

**for num in "${array[@]}"**

**do**

**if [ $((num % 2)) -eq 0 ]**

**then**

**sum\_even=$((sum\_even + num))**

**else**

**sum\_odd=$((sum\_odd + num))**

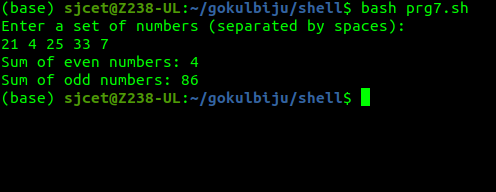
**fi**

**done**

**echo "Sum of even numbers: $sum\_even"**

**echo "Sum of odd numbers: $sum\_odd"**

**Output**

****

1. **Write a Shell program to find the roots of a quadratic equation.**

**Code:**

**echo Enter the coefficient of x^2:**

**read a**

**echo Enter the coefficient of x:**

**read b**

**echo Enter the constant term:**

**read c**

**discriminant=$((b\*b - 4\*a\*c))**

**if [ $discriminant -lt 0 ]**

**then**

**echo "The quadratic equation has no real roots."**

**else**

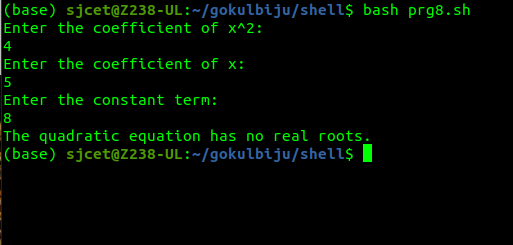
**root1=$(echo "scale=2; (-$b + sqrt($discriminant)) / (2\*$a)" | bc)**

**root2=$(echo "scale=2; (-$b - sqrt($discriminant)) / (2\*$a)" | bc)**

**echo "The roots of the quadratic equation are: $root1 and $root2"**

**fi**

**Output**

****

1. **Write a Shell program to check the given integer is Armstrong number or not.**

**Code:**

**echo "Enter a number: "**

**read c**

**x=$c**

**sum=0**

**r=0**

**n=0**

**while [ $x -gt 0 ]**

**do**

**r=`expr $x % 10`**

**n=`expr $r \\* $r \\* $r \\* $r`**

**sum=`expr $sum + $n`**

**x=`expr $x / 10`**

**done**

**if [ $sum -eq $c ]**

**then**

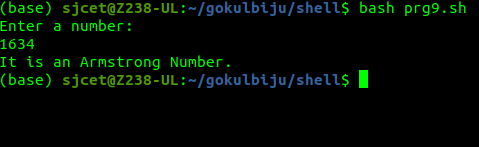
**echo "It is an Armstrong Number."**

**else**

**echo "It is not an Armstrong Number."**

**fi**

**Output**

****

1. **Write a Shell program to check the given integer is prime or not.**

**Code:**

**echo -e "Enter Number : \c"**

**read n**

**while [ $n -lt 2 ]**

**do**

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

**do**

**ans=$(( n%i ))**

**if [ $ans -eq 0 ]**

**then**

**echo "$n is not a prime number."**

**exit 0**

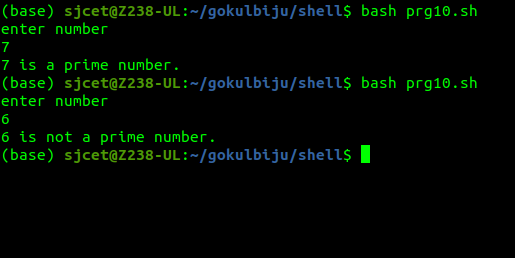
**fi**

**done**

**done**

**echo "$n is a prime number."**

**Output**

****

1. **Write a Shell program to generate prime numbers between 1 and 50.**

**Code:**

**echo "Enter a limit"**

**read limit**

**echo "prime numbers upto $limit are :"**

**echo "1"**

**i=2**

**while [ $i -le $limit ]**

**do**

**flag=1**

**j=2**

**while [ $j -lt $i ]**

**do**

**rem=$(( $i % $j ))**

**if [ $rem -eq 0 ]**

**then**

**flag=0**

**break**

**fi**

**j=$(( $j+1 ))**

**done**

**if [ $flag -eq 1 ]**

**then**

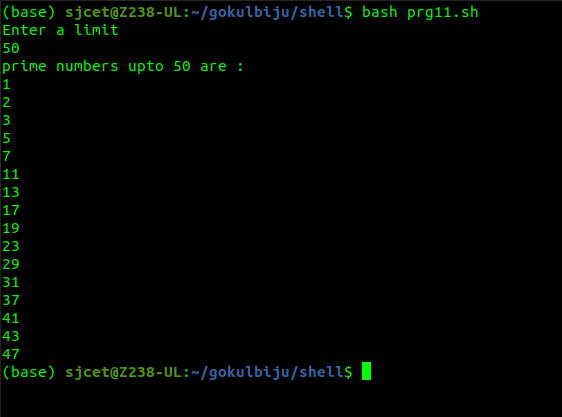
**echo "$i"**

**fi**

**i=$(( $i+1 ))**

**done**

**Output**

****

1. **Write a Shell program to find the sum of square of individual digits of a number.**

**Code:**

**echo "Enter a number: "**

**read number**

**sum=0**

**while [ $number -ne 0 ]**

**do**

**digit=$((number % 10))**

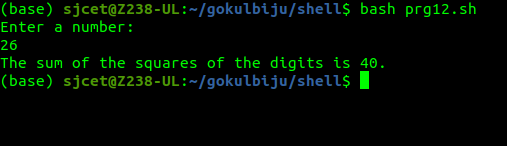
**sum=$((sum + digit \* digit))**

**number=$((number / 10))**

**done**

**echo "The sum of the squares of the digits is $sum."**

**Output**

****

1. **Write a Shell program to count the number of vowels in a line of text.**

**Code:**

**echo "Enter a line of text: "**

**read line**

**# Initialize the vowel count to 0**

**count=0**

**# Loop through each character of the line and check if it is a vowel**

**for (( i=0; i<${#line}; i++ ))**

**do**

**char=${line:$i:1}**

**if [[ $char == [aeiouAEIOU] ]]**

**then**

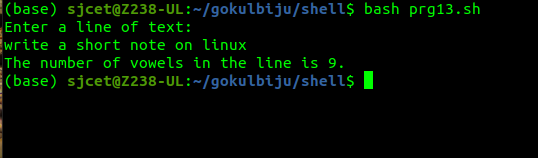
**count=$((count + 1))**

**fi**

**done**

**echo "The number of vowels in the line is $count."**

**Output**

****

1. **Write a Shell program to display student grades.**

**Code:**

**echo Enter the Student name**

**read name**

**echo Enter the Register number**

**read rno**

**echo Enter the Mark1**

**read m1**

**echo Enter the Mark2**

**read m2**

**echo Enter the Mark3**

**read m3**

**echo Enter the Mark4**

**read m4**

**echo Enter the Mark5**

**read m5**

**tot=$(expr $m1 + $m2 + $m3 + $m4 + $m5)**

**avg=$(expr $tot / 5)**

**echo -----------------------------------**

**echo '/t Student Mark List'**

**echo -----------------------------------**

**echo "Student Name : $name"**

**echo "Register Number : $rno"**

**echo "Mark1 : $m1"**

**echo "Mark2 : $m2"**

**echo "Mark3 : $m3"**

**echo "Mark4 : $m4"**

**echo "Mark5 : $m5"**

**echo "Total : $tot"**

**echo "Average : $avg"**

**if [ $m1 -ge 35 ] && [ $m2 -ge 35 ] && [ $m3 -ge 35 ] && [ $m4 -ge 35 ] && [ $m5 -ge 35 ]**

**then**

**echo "Result : Pass"**

**if [ $avg -ge 90 ]**

**then**

**echo "Grade : S"**

**elif [ $avg -ge 80 ]**

**then**

**echo "Grade : A"**

**elif [ $avg -ge 70 ]**

**then**

**echo "Grade : B"**

**elif [ $avg -ge 60 ]**

**then**

**echo "Grade : C"**

**elif [ $avg -ge 50 ]**

**then**

**echo "Grade : D"**

**elif [ $avg -ge 35 ]**

**then**

**echo "Grade : E"**

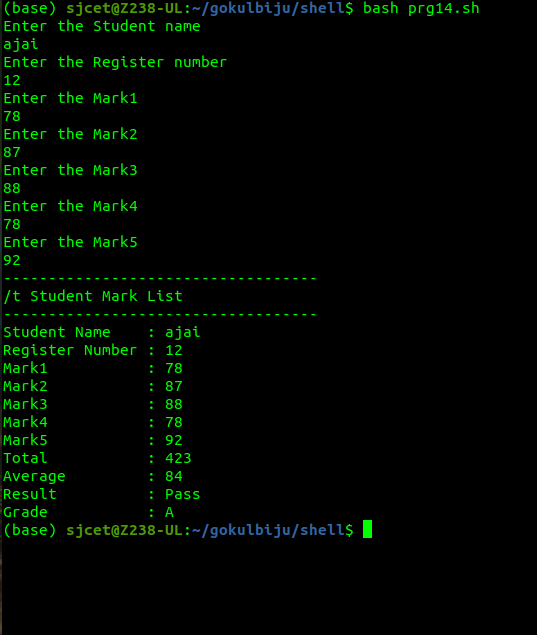
**fi**

**else**

**echo "Result : Fail"**

**fi**

**Output**

****

1. **Write a Shell program to find the smallest and largest numbers from a set of numbers.**

**Code:**

**echo "enter size of an array"**

**read n**

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

**do**

**echo " enter $((i+1)) number"**

**read nos[$i]**

**done**

**echo "number entered are"**

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

**do**

**echo ${nos[$i]}**

**done**

**small=${nos[0]}**

**greatest=${nos[0]}**

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

**do**

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

**small=${nos[$i]}**

**elif [ ${nos[$i]} -gt $greatest ]; then**

**greatest=${nos[$i]}**

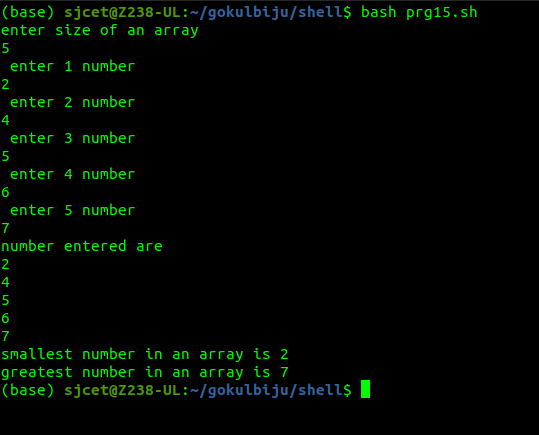
**fi**

**done**

**echo "smallest number in an array is $small"**

**echo "greatest number in an array is $greatest"**

**Output**

****

1. **Write a Shell program to find the smallest digit from a number.**

**Code:**

**echo "Enter a number with spacing: "**

**read num**

**min=${num:0:1}**

**for (( i=1; i<${#num}; i++ ))**

**do**

**digit=${num:$i:1}**

**if (( digit < min )); then**

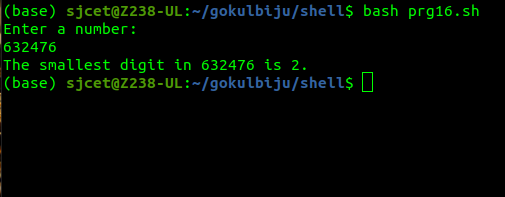
**min=$digit**

**fi**

**done**

**echo "The smallest digit in $num is $min."**

**Output**

****

1. **Write a Shell program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.**

**Code:**

**for((i = 50 ; i<= 100 ; i++))**

**do**

**if [ `expr $i % 3` = 0 -a `expr $i % 5` != 0 ]**

**then**

**echo $i**

**fi**

**done**

**Output**

****

1. **Write a Shell program to find the second highest number from a set of numbers.**

**Code:**

**echo "Enter a set of numbers separated by spaces: "**

**read numbers**

**# Convert the space-separated string to an array**

**arr=($numbers)**

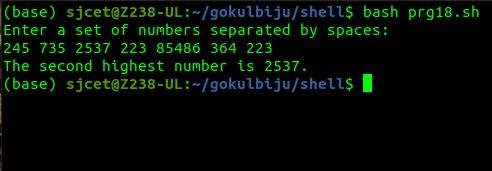
**# Sort the array in descending order**

**sorted\_arr=($(echo "${arr[@]}" | tr " " "\n" | sort -rn))**

**# Output the second highest number**

**echo "The second highest number is ${sorted\_arr[1]}."**

**Output**

****

1. **Write a Shell program to find the sum of digits of a number using function.**

**Code:**

**function sum\_of\_digits {**

**local number=$1**

**local sum=0**

**while (( number > 0 )); do**

**sum=$(( sum + number % 10 ))**

**number=$(( number / 10 ))**

**done**

**echo "$sum"**

**}**

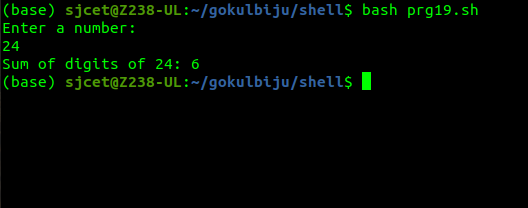
**echo "Enter a number:"**

**read number**

**result=$(sum\_of\_digits $number)**

**echo "Sum of digits of $number: $result"**

**Output**

****

1. **Write a Shell program to print the reverse of a number using function.**

**Code:**

**function reverse\_number {**

**local number=$1**

**local reverse=0**

**while (( number > 0 )); do**

**reverse=$(( reverse \* 10 + number % 10 ))**

**number=$(( number / 10 ))**

**done**

**echo "$reverse"**

**}**

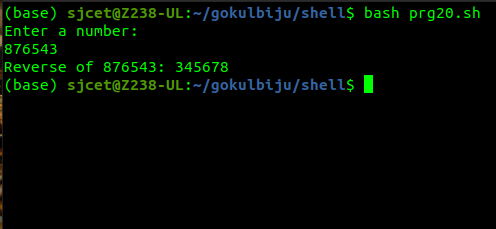
**echo "Enter a number:"**

**read number**

**result=$(reverse\_number $number)**

**echo "Reverse of $number: $result"**

**Output**

****

1. **Write a Shell program to find the factorial of a number using for loop.**

**Code:**

**echo "Enter a number:"**

**read number**

**factorial=1**

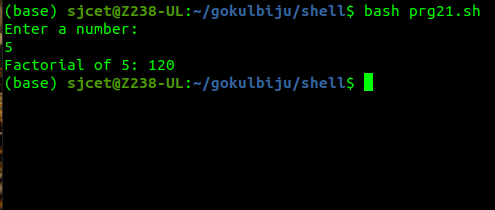
**for (( i=1; i<=number; i++ )); do**

**factorial=$(( factorial \* i ))**

**done**

**echo "Factorial of $number: $factorial"**

**Output**

****

1. **Write a Shell program to generate Fibonacci series.**

**Code:**

**echo "Enter the length of the Fibonacci series:"**

**read length**

**num1=0**

**num2=1**

**echo -n "$num1 $num2 "**

**for (( i=2; i<length; i++ )); do**

**next=$(( num1 + num2 ))**

**echo -n "$next "**

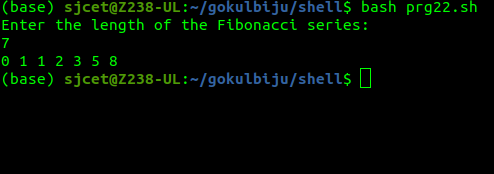
**num1=$num2**

**num2=$next**

**done**

**echo ""**

**Output**



1. **Write a shell script, which receives two filenames as arguments. It checks whether the two files contents are same or not. If they are same then second file is deleted.**

**Code:**

**if [ $# -ne 2 ]; then**

**echo "Usage: $0 file1 file2"**

**exit 1**

**fi**

**if cmp -s "$1" "$2"; then**

**echo "The contents of $1 and $2 are the same. Deleting $2..."**

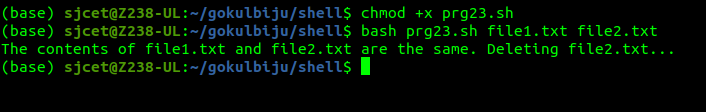
**rm "$2"**

**else**

**echo "The contents of $1 and $2 are different."**

**fi**

**Output**

****

1. **Write a Menu driven Shell script that Lists current directory, Prints Working Directory, displays Date and displays Users logged in**

**Code:**

**while true**

**do**

**clear**

**echo "Menu:"**

**echo "1. List current directory"**

**echo "2. Print working directory"**

**echo "3. Display date"**

**echo "4. Display users logged in"**

**echo "5. Exit"**

**read -p "Enter your choice: " choice**

**case $choice in**

**1)**

**ls -l**

**read -p "Press enter to continue"**

**;;**

**2)**

**pwd**

**read -p "Press enter to continue"**

**;;**

**3)**

**date**

**read -p "Press enter to continue"**

**;;**

**4)**

**who**

**read -p "Press enter to continue"**

**;;**

**5)**

**exit 0**

**;;**

**\*)**

**echo "Invalid choice. Press enter to try again"**

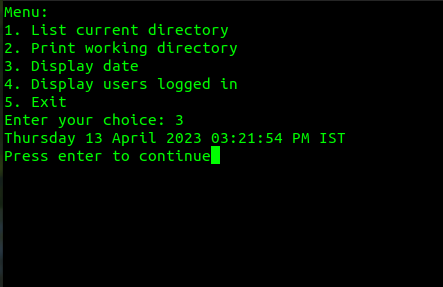
**read**

**;;**

**esac**

**done**

**Output**

****

1. **Shell script to check executable rights for all files in the current directory, if a file does not have the execute permission then make it executable.**

**Code:**

**echo -n "Enter file name : "**

**read file**

**[ -w $file ] && W="Write = yes" || W="Write = No"**

**[ -x $file ] && X="Execute = yes" || X="Execute = No"**

**[ -r $file ] && R="Read = yes" || R="Read = No"**

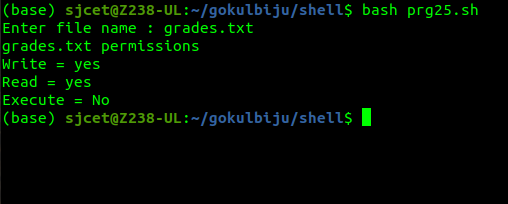
**echo "$file permissions"**

**echo "$W"**

**echo "$R"**

**echo "$X"**

**Output**

****

1. **Write a Shell program to generate all combinations of 1, 2, and 3 using loop.**

**Code:**

**for i in 1 2 3**

**do**

**for j in 1 2 3**

**do**

**for k in 1 2 3**

**do**

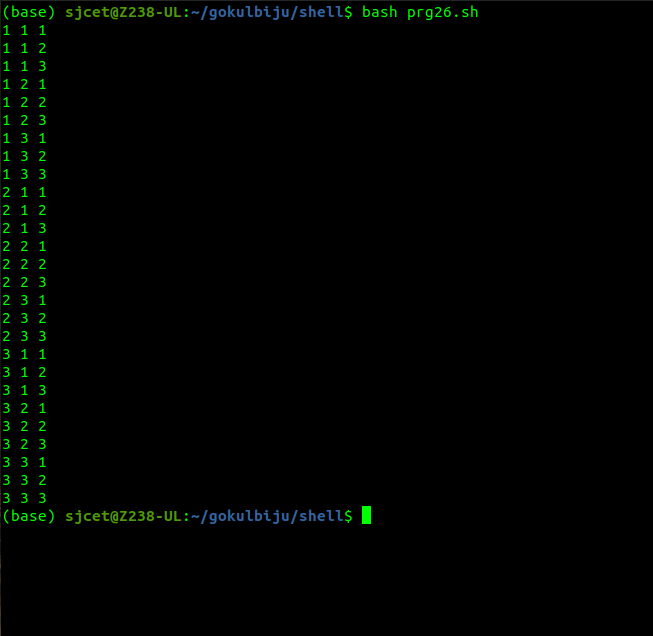
**echo "$i $j $k"**

**done**

**done**

**done**

**output**

****

1. **Write a Shell program to create the number series.**

1

2 3

4 5 6

7 8 9 10

**Code:**

count=1

for (( i=1; i<=4; i++ ))

do

for (( j=1; j<=i; j++ ))

do

echo -n "$count "

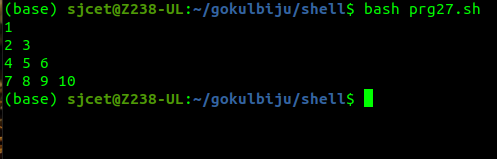
count=$((count+1))

done

echo ""

done

**output**



1. **Write a Shell program to create Pascal’s triangle.**

**Code:**

**echo "Enter the number of rows to generate for Pascal's triangle:"**

**read rows**

**row=1**

**echo $row**

**for ((i=1; i<$rows; i++)); do**

**prev\_row=($row)**

**row=${prev\_row[0]}**

**for ((j=1; j<=i; j++)); do**

**current=$((prev\_row[j-1] + prev\_row[j]))**

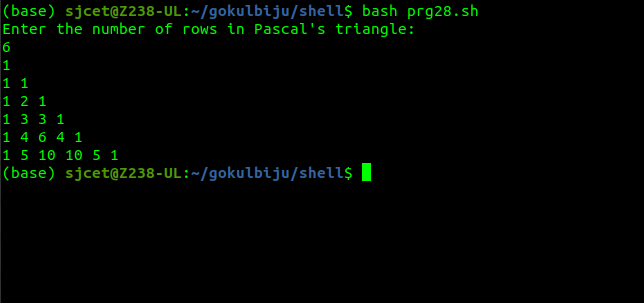
**row="$row $current"**

**done**

**echo $row**

**done**

**Output**

****

1. **Write a Decimal to Binary Conversion Shell Script**

**Code:**

**# Prompt the user for the decimal number to convert**

**echo "Enter a decimal number: "**

**read decimal**

**# Convert the decimal number to binary**

**binary=""**

**while [ $decimal -gt 0 ]; do**

**remainder=$((decimal % 2))**

**binary="$remainder$binary"**

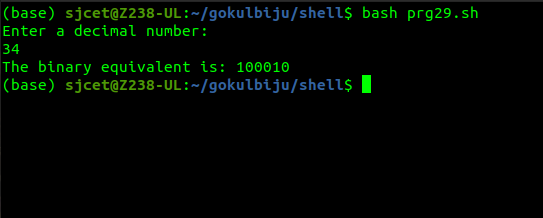
**decimal=$((decimal / 2))**

**done**

**# Print the binary number**

**echo "The binary equivalent is: $binary"**

**Output**

****

1. **Write a Shell Script to Check Whether a String is Palindrome or not**

**Code:**

**# Prompt the user for the string to check**

**echo "Enter a string: "**

**read string**

**# Reverse the string**

**reverse=$(echo $string | rev)**

**# Check if the string is equal to its reverse**

**if [ "$string" == "$reverse" ]; then**

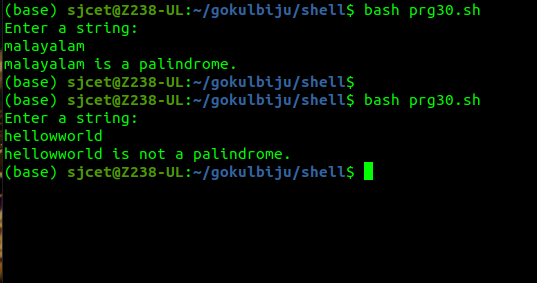
**echo "$string is a palindrome."**

**else**

**echo "$string is not a palindrome."**

**fi**

**Output**

****

1. **Write a shell script to find out the unique words in a file and also count the occurrence of each of these words.**

**Code:**

**# Prompt the user for the file name**

**echo "Enter the file name: "**

**read file**

**# Check if the file exists**

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

**echo "File not found."**

**exit 1**

**fi**

**# Convert the contents of the file to lowercase and replace all non-alphanumeric characters with spaces**

**contents=$(tr '[:upper:]' '[:lower:]' < $file | sed 's/[^a-z0-9]/ /g')**

**# Create an array of words from the file contents**

**words=($contents)**

**# Loop through the array of words and count their occurrences**

**declare -A count**

**for word in "${words[@]}"; do**

**if [ -n "$word" ]; then**

**((count[$word]++))**

**fi**

**done**

**# Print the unique words and their counts**

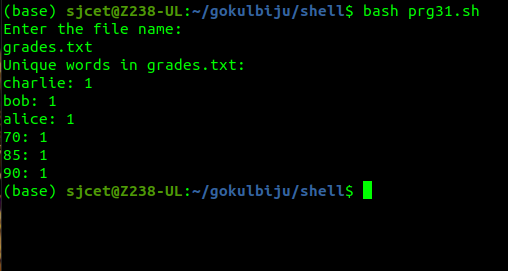
**echo "Unique words in $file:"**

**for word in "${!count[@]}"; do**

**echo "$word: ${count[$word]}"**

**done**

**Output**

****

1. **Write a shell script to get the total count of the word “Linux” in all the “.txt” files and also across files present in subdirectories.**

**Code:**

**search\_dir="."**

**files=$(find "$search\_dir" -type f -name "\*.txt")**

**count=0**

**for file in $files; do**

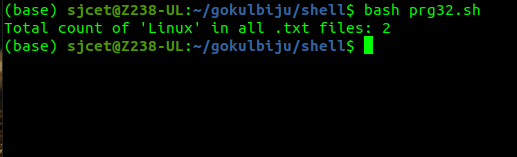
**occurrences=$(grep -o "Linux" "$file" | wc -l)**

**count=$((count + occurrences))**

**done**

**echo "Total count of 'Linux' in all .txt files: $count"**

**Output**

****

1. **Write a shell script to validate password strength. Here are a few assumptions for the password string.**

**Length – minimum of 8 characters.**

**Contain both alphabet and number.**

**Include both the small and capital case letters.**

**Code:**

**read -p "Enter your password: " password**

**# Check if password is at least 8 characters long**

**if [[ ${#password} -lt 8 ]]; then**

**echo "Password length must be at least 8 characters."**

**exit 1**

**fi**

**# Check if password contains both alphabet and number**

**if ! [[ "$password" =~ [A-Za-z]+[0-9]+ ]]; then**

**echo "Password must contain both alphabet and number."**

**exit 1**

**fi**

**# Check if password includes both small and capital case letters**

**if ! [[ "$password" =~ [a-z]+ ]] || ! [[ "$password" =~ [A-Z]+ ]]; then**

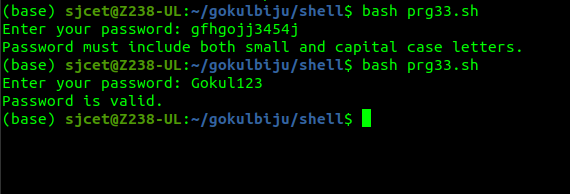
**echo "Password must include both small and capital case letters."**

**exit 1**

**fi**

**echo "Password is valid."**

**Output**

****

1. **Write a shell script to print the count of files and subdirectories in the specified directory.**

**Code:**

**if [ $# -eq 0 ]; then**

**echo "Usage: $0 directory"**

**exit 1**

**fi**

**directory=$1**

**if [ ! -d $directory ]; then**

**echo "Error: $directory is not a directory"**

**exit 1**

**fi**

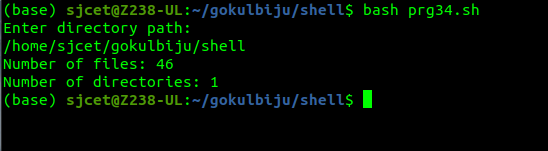
**num\_files=$(find $directory -maxdepth 1 -type f | wc -l)**

**num\_dirs=$(find $directory -maxdepth 1 -type d | wc -l)**

**echo "Number of files in $directory: $num\_files"**

**echo "Number of directories in $directory: $((num\_dirs - 1))"**

**Output**

****

1. **Write a shell script to reverse the list of strings and reverse each string further in the list.**

**Code:**

**strings="string1**

**string2**

**string3**

**string4**

**string5"**

**readarray -t strings\_array <<< "$strings"**

**for (( i=${#strings\_array[@]}-1; i>=0; i-- )); do**

**# reverse each string in the array**

**reversed\_string=$(echo "${strings\_array[i]}" | rev)**

**# replace the original string with the reversed string in the array**

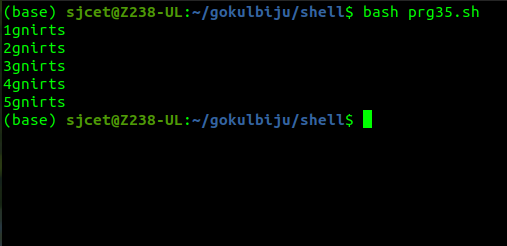
**strings\_array[i]=$reversed\_string**

**done**

**reversed\_strings=$(printf '%s\n' "${strings\_array[@]}")**

**echo "$reversed\_strings"**

**Output**

****