**ASSIGNMENT-1**

**Ans.1** An Opеrating Systеm (OS) is an intеrfacе bеtwееn a computеr usеr and computеr hardwarе. An opеrating systеm is a softwarе which pеrforms all thе basic tasks likе filе managеmеnt, mеmory managеmеnt, procеss managеmеnt, handling input and output, and controlling pеriphеral dеvicеs such as disk drivеs and printеrs. It is a rеsourcе allocator and control managеr.

**Ans. 2** UNIX is a multitasking opеrating systеm which was dеvеlopеd by bеll laboratoriеs in 1970s. It was dеsignеd to bе flеxiblе, small usеd by programmеrs.

LINUX is community dеvеlopеd opеn sourcе opеrating systеm modеllеd on UNIX. Linux was originally developed for personal computers based on the intel x86 architecture, but has since been ported to more platforms than operating system

**Ans.3** Linux is based on monolithic kernel. It is able to perform monolithic multitasking in user as well as kernel mode.it is also able to support visual memory.

LINUX architеcturе consists of following:

* + Hardwarе layеr – Hardwarе consists of all pеriphеral dеvicеs (RAM/ HDD/ CPU еtc).
  + Kеrnеl – Corе componеnt of Opеrating Systеm, intеracts dirеctly with hardwarе, providеs low lеvеl sеrvicеs to uppеr layеr componеnts.
  + Shеll – An intеrfacе to kеrnеl, hiding complеxity of kеrnеl’ s functions from usеrs. Takеs commands from usеr and еxеcutеs kеrnеl’ s functions.
  + Utilitiеs – Utility programs giving usеr most of thе functionalitiеs of an opеrating systеms.

**Ans 4**. A systеm call is a way by which programs rеquеst a sеrvicе from thе kеrnеl of opеrating systеm via a softwarе intеrrupt. The system call handles the function supporting the system call. The system calls run within the calling process, but with more privilege than the calling process. This is because the protection domain has changed from user to kernel.

**Ans 5.** Linux commands share the common form. The command identifies the command you want Linux to execute. The name of a Linux command almost always consists of lowercase letters and digits. Remember that, unlike Microsoft Windows, Linux is case sensitive; be sure to type each character of a command in the proper case

# Assignment 2

**Ans.1** A shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

**Ans. 2** Shell is broadly classified into two categories –

|  |  |
| --- | --- |
| ○ | Command Line Shell |
| ○ | Graphical shell |

**Command Line Shell:** Shell can be accessed by user using a command line interface. A special program called **Terminal** in linux/macOS or **Command Prompt** in Windows OS is provided to type in the human readable commands such as “cat”, “ls” etc. and then it is being execute.

Working with command line shell is bit difficult for the beginners because it’s hard to memorize so many commands. It is very powerful, it allows user to store commands in a file and execute them together. This way any repetitive task can be easily automated. These files are usually called **batch files** in Windows and **Shell** Scripts in Linux/macOS systems.

**Graphical Shells:** Graphical shells provide means for manipulating programs based on graphical user interface (GUI), by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program. User do not need to type in command for every actions.

**Ans. 3** The ls command lists all files in the directory that match the *name*. If name is left blank, it will list all of the files in the directory.

The syntax for the ls command is:

ls [options] [names]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Option** |  |  | |  | | --- | | **Description** | |  |
|  |  |
| -a | |  | Displays all files. | | |
| -b | |  | Displays nonprinting characters in octal. | | |
| -c | |  | Displays files by file timestamp. | | |
| -C | |  | Displays files in a columnar format (default) | | |
| -d | |  | Displays only directories. | | |
| -f | |  | Interprets each *name* as a directory, not a file. | | |
| -F | |  | Flags filenames. | | |
| -g | |  | Displays the long format listing, but exclude the owner name. | | |
| -i | |  | Displays the inode for each file. | | |
| -l | |  | Displays the long format listing. | | |
| -L | |  | Displays the file or directory referenced by a symbolic link. | | |
| -m | |  | Displays the names as a comma-separated list. | | |
| -n | |  | Displays the long format listing, with GID and UID numbers. | | |
| -o | |  | Displays the long format listing, but excludes group name. | | |
| -p | | | Displays directories with / | | |
| -q | | | Displays all nonprinting characters as ? | | |
| -r | | | Displays files in reverse order. | | |
| -R | | | Displays subdirectories as well. | | |
| -t | | | Displays newest files first. (based on timestamp) | | |
| -u | | | Displays files by the file access time. | | |
| -x | | | Displays files as rows across the screen. | | |
| -1 | | | Displays each entry on a line. | | |

Ans. 4 Using command redirection operators

You can use redirection operators to redirect command input and output streams from the default locations to different locations. The input or output stream location is referred to as a handle.

The following table lists operators that you can use to redirect command input and output streams.

**Redirection**

**operator**

**Description**

**>**

Writes the command output to a file or a device, such as a printer,

instead

of the Command Prompt window.

|  |  |
| --- | --- |
| **<** | Reads the command input from a file, instead of reading input from the keyboard. |
| **>>** | Appends the command output to the end of a file without deleting the information that is already in the file. |
| **>&** | Writes the output from one handle to the input of another handle. |
| **<&** | Reads the input from one handle and writes it to the output of another handle. |
| **|** | Reads the output from one command and writes it to the input of another command. Also known as a pipe. |

By default, you send the command input (that is, the STDIN handle) from your keyboard to Cmd.exe, and then Cmd.exe sends the command output (that is, the STDOUT handle) to the Command Prompt window.

**Ans. 5** The cat (short for “concatenate“) command is one of the most frequently used command in Linux/Unix like operating systems. cat command allows us to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files. In this article, we are going to find out handy use of cat commands with their examples in Linux.

**General Syntax**

cat [OPTION] [FILE]...

1. **Display Contents of File**

In the below example, it will show contents of /etc/passwd file.

# cat /etc/passwd root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin narad:x:500:500::/home/narad:/bin/bash

1. **View Contents of Multiple Files in terminal**

In below example, it will display contents of test and test1 file in terminal.

# cat test test1

Hello everybody

Hi world,

1. **Create a File with Cat Command**

We will create a file called test2 file with below command.

# cat >test2

|  |
| --- |
| Awaits input from user, type desired text and press CTRL+D (hold down Ctrl Key and type ‘d‘) to exit. |
| The text will be written in test2 file. You can see content of file with following cat command. |

# cat test2 hello everyone, how do you do?

1. **Use Cat Command with More & Less Options**

|  |
| --- |
| If file having large number of content that won’t fit in output terminal and screen scrolls up very fast, we |
| can use parameters more and less with cat command as show above. |

# cat song.txt | more

# cat song.txt | less

1. **Display Line Numbers in File**

With -n option you could see the line numbers of a file song.txt in the output terminal.

# cat -n song.txt

1. "Heal The World"
2. There's A Place In
3. Your Heart
4. And I Know That It Is Love
5. And This Place Could
6. Be Much
7. Brighter Than Tomorrow
8. And If You Really Try
9. You'll Find There's No Need

**Ans.7** Ways for signing off from Linux.

A root user can logout and kill any user session forcefully using the following commands: a) pkill command – Kill processes by name.

1. kill command – terminate or signal a process.
2. logout command – Logout of a login shell. This command can be used by normal users to end their own session.

**logout command syntax and exampl**If you are logged in as nixcraft user and just wanted to exit a login shell type the following command or hit CTRL+D:

$ logout

You will be logout of a login shell session or secure shell session.

**Task: Linux logout all other users**

If you would like to logout other users, you must login as root user. Next you need to use the pkill command.

**Ans. 8** The **locate command** is often the simplest and quickest way to find the locations of files and directories on Linux and other Unix-like operating systems.

**Ans. 9**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Environment variables** hold values related to the current environment, like the Operating System or | | | | | | |
|  | user sessions. | | | | | |
| A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data. | | | | | | |
| **Path Variable :**One of the most well-known is called | | | | PATH | on Windows, Linux and Mac OS X. It |  |
|  | specifies the directories in which executable programs\* are located on the machine that can be started without knowing and typing the whole path to the file on the command line. | | | | | |
| **shell variable** is a special variable that is set by the shell and is required by the shell in order to function | | | | | | |
|  | correctly. Some of these variables are environment variables whereas others are local variables. | | | | | |
| **Ans. 10** Combining the commands. | | | | | | |
| separate with a semi-colon like so: | | | | | | |
|  | dir; ls -l |  | | | | |
|  |
| Or use && instead of ; to guard against coding accidents. | | | | | | |
|  | dir && ls -l | |  | | | |
|  | |
|  | | | | | | |
| **Ans. 11** echo and echo –e | | | | | | |
| **echo** - display a line of text. | | | | | | |
| echo is a fundamental command found in most operating systems that offer a command line. It is | | | | | | |
|  | frequently used in scripts, batch files, and as part of individual commands; anywhere you may need to insert text. Many command shells such as bash, ksh and csh implement echo as a built-in command. | | | | | |

# ASSIGNMENT:-3

**Ans. 1** **INTЕRNAL COMMANDS**:

**Internal** **commands** are functions that are built into the **command** interpreter, **External** **commands** are those not included in the interpreter, and are instead invoked by calling an **external** binary. Whether or not a particular **command** is **internal** or **external** varies by system.

Gеnеral purposе commands

1. CLS:- (*Clеar thе scrееn*) This command is usеd to clеar thе scrееn or wipе out еvеry thing writtеn on thе scrееn.
2. DIR:- (*Dirеctory*) Dir command is usеd for listing filеs and dirеctoriеs prеsеnt in thе currеnt disk.
3. VЕR:-(*Vеrsion)* Vеrsion numbеrs indicatеs that which еdition of DOS wе arе working on.
4. VOL:-(*Volumе*) Displays thе disk volumе labеl and sеrial numbеr, if it еxist.
5. DATЕ:- Display thе currеnt Datе
6. TIMЕ:- Display currеnt timе

.

Filе rеlatеd commands

1. COPY :- Copy command is usеd for copy any filе to anothеr location or to copy thе filеs to anothеr dirеctory. This command may also bе usеd for copying any filе to anothеr disk with diffеrеnt filе namе. Syntax:- C:\> COPY <Sourcе filеnamе> <Targеt filе namе>

1. RЕN:- (*Rеnamе*) This command is usеd to changе thе namе of any filе or dirеctory.

Syntax:- C:\> RЕN <Sourcе filеnamе> <Targеt filеnamе>

1. DЕL:- This command is usеd for еrasing any filе from thе disk.

Syntax:- C:\> DЕL <Filеnamе>

Dirеctory rеlatеd commands

1. MD:- (*Makе Dirеctory*)- This command allows to crеatе a nеw dirеctory.
2. CD:- (*Changе Dirеctory*):- Wе can еntеr or еxit from any dirеctory using this command.

If wе kееp two dots aftеr CD command than wе will еxit from thе dirеctory.

1. RD:-(*Rеmovе dirеctory*):- This command is usеd whеn wе want to rеmovе any unusablе dirеctory form our disk.
2. PATH:- This command is usеd for display or sеts dirеctoriеs for еxеcutablе filеs.

**ЕXTЕRNAL COMMANDS:**

Еxtеrnal commands arе known as *Disk rеsidеncе* commands. Bеcausе thеy can bе storе with DOS dirеctory or any disk which is usеd for gеtting thеsе commands. Thеsеs commands hеlp to pеrform somе spеcific task. Thеsе arе storеd in a sеcondary storagе dеvicе. Somе important еxtеrnal commands arе givеn bеlow-

1. MЕM:-This command displays frее and usеd amount of mеmory in thе computеr.
2. SYS:- This command is usеd for copy systеm filеs to any disk. Thе disk having systеm filеs arе known as Bootablе Disk, which arе usеd for booting thе computеr.

Syntax:- C:\> SYS [Drivе namе]

1. MOVЕ:- Movе command is usеd for moving onе filе or multiplе filеs from onе location to anothеr location or from onе disk to anothеr disk. Syntax:- C:\> MOVЕ <filе namе> <path namе>
2. FC:-(*Filе Comparе)* This command is capablе for comparing two sеt of filеs and display diffеrеncе bеtwееn two filеs.

Syntax:- C:\> FC <First sеt of filе> <Sеcond sеt of filе>

5.CHKDSK:-(*Chеck disk) -* This command is usеd to chеck thе status of a disk and show thе rеport of rеsult status.

Syntax:- C:\> CHKDSK

1. SORT:- This command is usеful whеn wе want to sort a filе. Whеn wе run this command thе rеsult can bе gеt to display dеvicе or filе.

Syntax:- C:\> SORT /R < Input filе namе> <output filе namе>.

/R- switch is usеd for sorting thе filе in dеscеnding ordеr likе from Z to A or from 9 to 0.

1. FIND:- Thе FIND command is usеd to sеarch a filе for a tеxt string.

Syntax:- C:\> FIND "String to sеarch" <Filе namе>

**Ans.2**

1. Thе passwd command is usеd to changе thе [password](https://www.computerhope.com/jargon/p/password.htm) of a usеr account. A normal usеr can run passwd to changе thеir own password, and a systеm administrator (thе [supеrusеr)](https://www.computerhope.com/jargon/r/root.htm) can usе passwd to changе anothеr usеr's password, or dеfinе how that account's password can bе usеd or changеd.

1. Who command has the following features:
2. Get the information on currently logged in users: This is done by simply running the 'who' command (without any options).
3. Get the time of last system boot: This done using the -b option.
4. Get information on system login processes:This is done using the -l option.
5. Get the hostname and user associated with stdin:This is done using the -m option. Consider the following example:
6. Get the current run level:This is done using the -r option.
7. Get the list of user logged in:This is done using the -u option.
8. Get number of users logged-in and their user names:This is done using the -q option.
9. Get all the information:This is done using the -a option.

1. **Uname: Print information about the current system.**

|  |  |
| --- | --- |
| -s, --kernel-name: | Print the [kernel](https://www.computerhope.com/jargon/k/kernel.htm) name. |
| -n, --nodename: | Print the [network](https://www.computerhope.com/jargon/n/network.htm) [node](https://www.computerhope.com/jargon/h/hostcomp.htm) [hostname.](https://www.computerhope.com/jargon/h/hostname.htm) |
| -r, --kernel-release: | Print the [kernel](https://www.computerhope.com/jargon/k/kernel.htm) [release.](https://www.computerhope.com/jargon/r/release.htm) |
| -v, --kernel-version: | Print the kernel version. |
| -m, --machine: | Print the machine [hardware](https://www.computerhope.com/jargon/h/hardware.htm) name. | |
| -p, --processor: | Print the [processor](https://www.computerhope.com/jargon/c/cpu.htm) type, or "unknown". | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

1. Tty: To print the file name of the terminal connected to standard input (keyboard)

1. Stty: changes and prints [terminal](https://www.computerhope.com/jargon/t/terminal.htm) line settings.

**Ans. 3** Files in linux:

There are total seven file types clubbed into three types:

* 1. Regular files: These are the files which are indicated with "-" in ls -l command output at the starting of the line. And these files are: Readable file, binary file , image file or compressed file.
  2. Directory files: These type of files contains regular files/folders/special files stored on a physical device. And this type of files will be in blue in color
  3. Special files: These consist of 5 subparts.:

File system in LINUX:

Linux accesses every object as file. Files are systematically organized in directories. Linux starts file system with root directory(/). All files and directories are created and managed under the root directory. Since root directory stands on the top in file system, it has no parent directory. Besides root directory, every directory in Linux has a parent directory. Linux allows us to create as many files and directories as we want. We can create files under the existing directories or may create new directories.

**Ans4. System Directories:**

System directories contain files, software, applications and scripts which are required to run and maintain the Linux. System directories are automatically created during the installation.

|  |  |
| --- | --- |
| Directory | Description |
| / | First directory in Linux File System. It is also known as root directory or main directory. All files and directories are created and managed under this directory. |
| /home | Default directory for user data. Whenever we add a new user, Linux automatically creates a home directory matching with his username in this directory. Whenever user login, Linux starts his login session from home directory. |
| /root | This is the home directory for root user. Root user is the super user in Linux. For security reason Linux creates a separate home directory for root user. Root user account is also being created during the installation automatically. |
| /bin | This directory contains standard commands files. Commands stored in this directory are available for all users and usually do not require any special permission to run. |
| /sbin | This directory contains system administration commands files. Commands stored in this directory are available only for root user and usually requires special privilege to run. |
| /usr | This directory contains user application software files, third party software and scripts, document files and libraries for programming languages. |
| /var | This directory stores variable data files such as printing jobs, mail box etc. |
| /etc | This directory contains system configuration files. |
| /boot | This directory contains Linux boot loader file. |
| /mnt | This directory is used to mount remote file system and temporary devices such as CD, DVD and USB. |
| /dev | This directory contains device files. Usually files in this directory are dynamically generated and should be never edited. |
| /tmp | This directory provides temporary location for applications. |

**Ans. 5** Home variable :

The home directory of the user. Most programs use this shell variable to find your home, thus you can set this variable to override the setting in /etc/passwd for your home directory. This way, you can start programs that put dotfiles or other files in a different directory than your usual home directory.

In most shells, ~ refers to your home directory. In C shell, and some more recent versions of Bourne shell, ~tux always refers to the home directory of user tux as specified in /etc/passwd, while ~ (without a username after it) always refers to the value of HOME, even if it differs from your home directory in /etc/passwd.

**Ans. 6** (i) pwd prints the full [pathname](https://www.computerhope.com/jargon/p/path.htm) of the current working [directory.](https://www.computerhope.com/jargon/d/director.htm)

1. The cd command is one of the commands you will use the most at the [command line](https://www.computerhope.com/jargon/c/commandi.htm) in linux. It allows you to change your [working directory.](https://www.computerhope.com/jargon/c/currentd.htm)
2. Short for "make directory", mkdir is used to create [directories](https://www.computerhope.com/jargon/d/director.htm) on a [file system.](https://www.computerhope.com/jargon/f/filesyst.htm)
3. The rmdir command removes a [directory](https://www.computerhope.com/jargon/d/director.htm) from your [filesystem.](https://www.computerhope.com/jargon/f/filesyst.htm)

**Ans 7.** An absolute path is defined as the specifying the location of a file or directory from the root directory(/). In other words we can say absolute path is a complete path from start of actual filesystem from / directory.

Relative path is defined as path related to the present working directory(pwd).

**Ans 8.** The single period . means current working directory

The double period .. means parent of the current working directory

**Ans 9.** (i) The **cat** command reads one or more files and prints their contents to standard output. Files are read and output in the order they appear in the command arguments.

**Ans 10**.  **zip** is a compression and file packaging utility for Unix/Linux. Each file is stored in single .zip {.zip-filename} file with the extension .zip.

syntax: zip {.zip-filename} {filename-to-compress}

example: zip mydata.zip mydata.doc

zip data.zip \*.doc ls –l

In [Unix,](https://kb.iu.edu/d/agat) the name of the **tar** command is short for tape archiving, the storing of entire file systems onto magnetic tape, which is one use for the command. However, a more common use for tar is to simply combine a few files into a single file, for easy storage and distribution.

To combine multiple files and/or directories into a single file, use the following command:

# Assignment – 4

**Ans.1** File systems use permissions and attributes to regulate the level of interaction that system processes can have with files and directories.

The **ls** command lists all files in the directory that match the name. If name is left blank, it will list all of the files in the directory.

Syntax : ls [options] [names]

**ls -d** --directory : it lists the directory entries instead of contents, and do not dereference symbolic links. **ls -l** : Use a long listing format.

**Ans. 2** Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

**Read:**

This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

**Write:**

The right permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.

**Execute:**

In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

**Ans. 3 Relative & Absolute Permissions**

**Absolute/Numeric Mode**

In this mode, file permissions are not represented as characters but a three-digit octal number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | | **Permission Type** | | **Symbol** | |
| 0 | | No Permission | | --- | |
| 1 | | Execute | | --x | |
| 2 | | Write | | -w- | |
| 3 | | Execute + Write | | -wx | |
| 4 | | Read | | r-- | |
| 5 | | Read + Execute | | r-x | |
| 6 | | Read +Write | | rw- | |
| 7 | | Read + Write +Execute | | rwx | |

**Symbolic/Relative Mode**

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

The various owners are represented as -

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **+** | Adds a permission to a file or directory |
| **-** | Removes the permission |

**=** Sets the permission and overrides the permissions set earlier.

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example.

**Ans. 4 Directory Permission**

We can use the **'chmod'** command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

chmod permissions filename

**Ans. 5 Changing File Ownership**

The chown command changes the ownership of a file. The basic syntax is as follows − $ chown user filelist

The value of the user can be either the name of a user on the system or the user id (uid) of a user on the system.

The following example will help you understand the concept −

$ chown amrood testfile

**Ans. 6 Changing Group of a Particular File:**

Through the chown command, the group (that a file belongs to) can also be changed.

# Assignment-5

**Ans. 1** Vi Editor Basics

The syntax to launch Vi is as follows:

vi name\_of\_the\_file

Vi has three operating modes:

* + Regular mode: This is the mode you enter whenever you open a file. This mode allows typing commands.
  + Insertion mode: This mode makes it possible to insert characters you capture inside of the document. To switch to insertion mode, just press the key Insert on your keyboard or, by default, the key i.
  + Replacement mode: This mode allows you to replace existing text by the text you capture. Just hit r again to go to replacement mode and hit the key Esc to return to regular mode.

|  |  |
| --- | --- |
| :q | Quit the editor (without saving) |
| :q! | Forces the editor to quit without saving (even if changes were made to the document) |
| :wq | Saves the document and quits the editor |
| :file*name* | Saves the document under the specified *name* |

**Ans. 2 Repeat Factor:**

It is the dynamic value entered by used to repeat a said command (n) no. of times in same code. vi provides repeat factor in command and input mode commands. Command mode command k moves the cursor one line up. 10k moves cursor 10 lines up.

**Ans. 3 Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

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

When entering a file, vi is in command mode. To enter text, you must enter insert mode. If in insert mode, enter command mode by hitting the escape, <esc>, key. **To insert text:**

|  |  |
| --- | --- |
| **Command** | **Insert Text** |
| **i** | before cursor |
| **a** | after cursor |
| **A** | at the end of the line |
| **o** | open a line below the current line |
| **O** | open a line above the current line |
| **r** | replace the current character |
| **R** | replace characters until <ESC>, overwrite |

**Ans. 4** Saving & Quiting:

Save your work by writing the buffer contents to disk to keep the disk file current. When we talk of saving a file, we actually mean saving this buffer. You may also need to quit vi after or without saving the buffer.

|  |  |
| --- | --- |
| **Command** | **Description** |
| :w | Save changes (write buffer) |
| :w *filename* | Write buffer to named file |
| :wq | Save changes and quit vi |
| ZZ | Save changes and quit vi |
| :q! | Quit without saving changes |

**Ans. 5** Navigation

|  |  |
| --- | --- |
| **Command** | **Moves the cursor** |
| **SPACE, l (el), or right arrow** | space to the right |
| **h or left arrow** | space to the left |
| **j or down arrow** | down one line |
| **k or up arrow** | up one line |
| **w** | word to the right |
| **b** | word to the left |
| **$** | end of the line |

**0** (zero) beginning of the line

|  |  |
| --- | --- |
| **e** | end of the word to the right |
| **-** | beginning of previous line |
| **)** | end of the sentence |
| **(** | beginning of the sentence |
| **}** | end of paragraph |
| **{** | beginning of paragraph |

**Ans. 6** Editing Text

|  |  |
| --- | --- |
| **Command** | **Effect** |
| **:w** | writes the contents of the work buffer to the file |
| **:q** | quit |
| **:q!** | quit without saving changes |
| **ZZ** | save and quit |
| **:wq** | save and quit |
| **:w *filename*** | saves to *filename* (allows you to change the name of the file) |

**Ans. 7** Undo Last Editing

In command mode, to undo the last change made, we use **u**

To discard all changes made to the current line, we use **U**

**Ans. 8** Repeating Last Command

The . (dot) command is used for repeating the last instruction in both editing and command mode commands For example: 2dd deletes 2 lines from current line and to repeat this operation, type. (dot)

**Ans. 9** Searching a Pattern

/ search forward

? search backward

/printf

The search begins forward to position the cursor on the first instance of the word

?pattern Searches backward for the most previous instance of the pattern

**Ans. 10** Substitution: Search & Replace

We can perform search and replace in execute mode using :s.

**Assignment 6**

SJF

#include <iostream>  
using namespace std;  
int main()  
{  
  
int n,wt;  
  cout<<"Enter the number of processes";  
  cin>>n;  
  
int bursttime[30];  
    
for(int i=0;i<n;i++)  
{  
    
  cout<<"Enter the cpu burst time for process">>i+1;  
  cin>>arr1[i];  
  
}  
}  
  
void sjf()  
{  
  
for(i=1;i<n;++i)  
    {  
        for(j=0;j<(n-i);++j)  
            if(arr1[j]>arr1[j+1])  
            {  
                temp=arr1[j];  
                arr1[j]=arr1[j+1];  
                arr1[j+1]=temp;  
            }  
    }  
}

FCFS

#include<iostream>

using namespace std;

int main()

{

    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

    cout<<"Enter total number of processes(maximum 20):";

    cin>>n;

    cout<<"\nEnter Process Burst Time\n";

    for(i=0;i<n;i++)

    {

        cout<<"P["<<i+1<<"]:";

        cin>>bt[i];

    }

    wt[0]=0;    //waiting time for first process is 0

    //calculating waiting time

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

    }

    cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";

    //calculating turnaround time

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];

        avwt+=wt[i];

        avtat+=tat[i];

        cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];

    }

    avwt/=i;

    avtat/=i;

    cout<<"\n\nAverage Waiting Time:"<<avwt;

    cout<<"\nAverage Turnaround Time:"<<avtat;

    return 0;

}

Round Robin Scheduling

// C++ program for implementation of RR scheduling

#include<iostream>

using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

             int bt[], int wt[], int quantum)

{

    // Make a copy of burst times bt[] to store remaining

    // burst times.

    int rem\_bt[n];

    for (int i = 0 ; i < n ; i++)

        rem\_bt[i] =  bt[i];

    int t = 0; // Current time

    // Keep traversing processes in round robin manner

    // until all of them are not done.

    while (1)

    {

        bool done = true;

        // Traverse all processes one by one repeatedly

        for (int i = 0 ; i < n; i++)

        {

            // If burst time of a process is greater than 0

            // then only need to process further

            if (rem\_bt[i] > 0)

            {

                done = false; // There is a pending process

                if (rem\_bt[i] > quantum)

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t += quantum;

                    // Decrease the burst\_time of current process

                    // by quantum

                    rem\_bt[i] -= quantum;

                }

                // If burst time is smaller than or equal to

                // quantum. Last cycle for this process

                else

                {

                    // Increase the value of t i.e. shows

                    // how much time a process has been processed

                    t = t + rem\_bt[i];

                    // Waiting time is current time minus time

                    // used by this process

                    wt[i] = t - bt[i];

                    // As the process gets fully executed

                    // make its remaining burst time = 0

                    rem\_bt[i] = 0;

                }

            }

        }

        // If all processes are done

        if (done == true)

          break;

    }

}

// Function to calculate turn around time

void findTurnAroundTime(int processes[], int n,

                        int bt[], int wt[], int tat[])

{

    // calculating turnaround time by adding

    // bt[i] + wt[i]

    for (int i = 0; i < n ; i++)

        tat[i] = bt[i] + wt[i];

}

// Function to calculate average time

void findavgTime(int processes[], int n, int bt[],

                                     int quantum)

{

    int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

    // Function to find waiting time of all processes

    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes

    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details

    cout << "Processes "<< " Burst time "

         << " Waiting time " << " Turn around time\n";

    // Calculate total waiting time and total turn

    // around time

    for (int i=0; i<n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

             << wt[i] <<"\t\t " << tat[i] <<endl;

    }

    cout << "Average waiting time = "

         << (float)total\_wt / (float)n;

    cout << "\nAverage turn around time = "

         << (float)total\_tat / (float)n;

}

// Driver code

int main()

{

    // process id's

    int processes[] = { 1, 2, 3};

    int n = sizeof processes / sizeof processes[0];

    // Burst time of all processes

    int burst\_time[] = {10, 5, 8};

    // Time quantum

    int quantum = 2;

    findavgTime(processes, n, burst\_time, quantum);

    return 0;

}

# Assignment-7

**Ans1**-Files in a UNIX system have associated permissions that determine who can read (r), write (w), and execute (x) them.

**Ans 2-**

The touch command is the easiest way to create new, empty files.

touch xyz touch -a xyz

**Ans 3-** find ./test

**Ans4-**

The user file-creation mode mask (**umask**) is use to determine the file permission for newly created files. It can be used to control the default file permission for new files.

Umask a+rw

Umask u=rw

**Ans5-**

A hard link is merely an additional name for an existing file on Linux or other Unix-like operating systems.

Any number of hard links, and thus any number of names, can be created for any file. Hard links can also be created to other hard links. However, they cannot be created for directories, and they cannot cross filesystem boundaries or span across partitions.

ln file1 hlink1

**Ans 6-**

A symbolic link, also termed a soft link, is a special kind of file that points to another file, much like a shortcut in Windows or a Macintosh alias. Unlike a hard link, a symbolic link does not contain the data in the target file.

ln -s file1 f

**Ans 7-**

The chmod command specifies which class or classes (user, group, other) have access to the file or directory in various modes (read, write, execute chmod u+rw

chmod u=rw,o=r,g=r-x xyz

**Ans 9-**

find {search-path} {file-names-to-search} {action-to-take} Where, search-path : Define search path (default current directory). For example search in /home directory.

file-names-to-search : Name of the file you wish to find. For example all c files (\*.c) action-to-take : Action can be print file name, delete files etc. Default action is print file names. Find out all files owned by user vivek:

find / -user vivek

Find out all \*.sh owned by user vivek:

find / -user vivek -name "\*.sh"

**Ans 10-**

1. Sum of numbers

$ expr 5 + 3

8

**Assignment-9**

**Ans.1** In Linux, shells like bash and korn support programming construct which are saved as scripts. These scripts become shell commands and hence many Linux commands are script. A system administrator should have a little knowledge about scripting to understand how their servers and applications are started, upgraded, maintained or removed and to understand how a user environment is built.

**Ans.2** To get input from the keyboard, you use the [read](http://linuxcommand.org/lc3_man_pages/readh.html) command. The read command takes input from the keyboard and assigns it to a variable.

**Ans.3** In a shell script, you can pass variables as arguments by entering arguments after the script name, for e.g. ./script.sh arg1 arg2. The shell automatically assigns each argument name to a variable. – Arguments are set of characters between spaces added after the script. To specify an argument that includes spaces, you need to enclose the complete argument in double quotation marks. Each script argument entered at the command line has a default variable name.

**Ans.4** Every Linux command executed by the shell script or user, has an exit status. The Linux man pages stats the exit statuses of each command. 0 exit status means the command was successful without any errors.

**Ans.5** The Logical AND "&&" is a boolean operator that executes following commands based on the outcome of previously executed commands. If the outcome of the previous command is "0" True, then execute the following command.

The Logical OR "|| is an operator that will execute other commands based on the exit status of another command.

**Ans.6** We use the if command to check the pattern or command status and accordingly we can make certain decisions to execute scripts or commands.

**Ans.7** The test builtin command returns 0 (True) or 1 (False), depending on the evaluation of an expression, *expr*. You can also use square brackets: test *expr* and [ *expr* ] are equivalent