EXPERIMENT 1: Introduction to Unix and its architecture. Introduction to basic Unix/Linux Commands.

What is Unix ?

The Unix operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

* Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
* There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
* Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
* A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture

Here is a basic block diagram of a Unix system −



The main concept that unites all the versions of Unix is the following four basics −

* **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
* **Shell** − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.
* **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
* **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

System Bootup

If you have a computer which has the Unix operating system installed in it, then you simply need to turn on the system to make it live.

As soon as you turn on the system, it starts booting up and finally it prompts you to log into the system, which is an activity to log into the system and use it for your day-to-day activities.

Login Unix

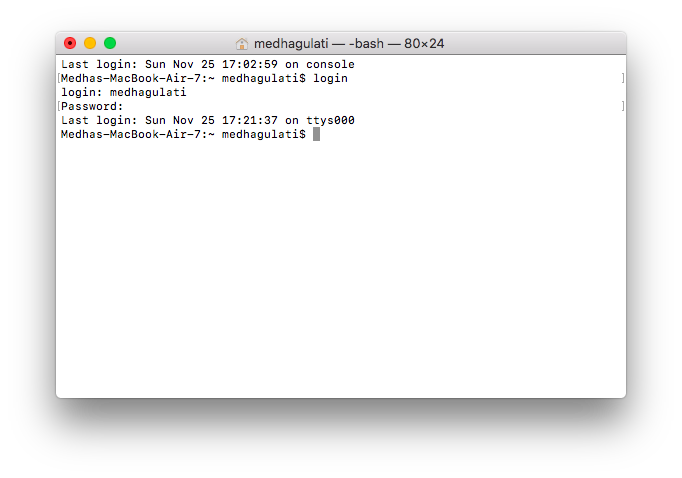
When you first connect to a Unix system, you usually see a prompt such as the following −

login:

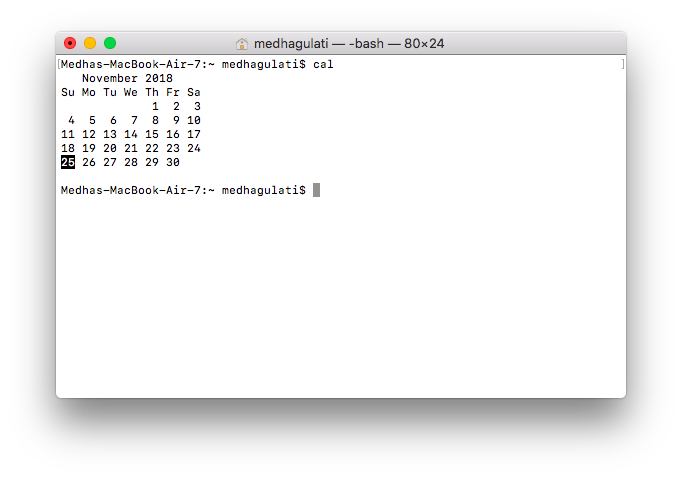
To log in

* Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.
* Type your userid at the login prompt, then press **ENTER**. Your userid is **case-sensitive**, so be sure you type it exactly as your system administrator has instructed.
* Type your password at the password prompt, then press **ENTER**. Your password is also case-sensitive.

If you provide the correct userid and password, then you will be allowed to enter into the system.



You will be provided with a command prompt (sometime called the **$** prompt ) where you type all your commands. For example, to check calendar, you need to type the **cal** command as follows −



### Change Password

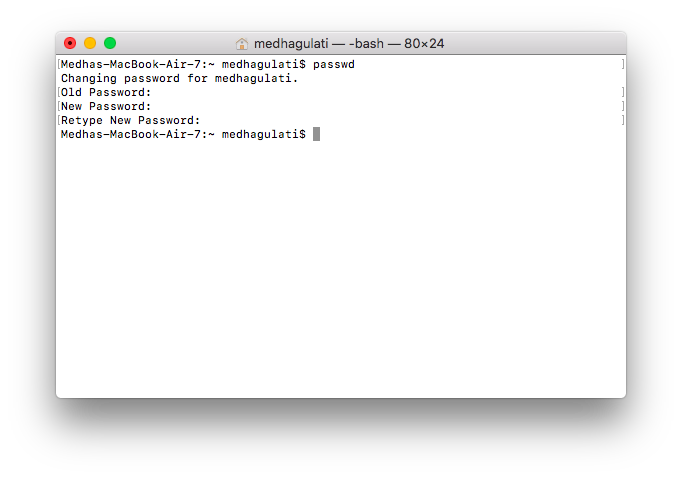
All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers. Following are the steps to change your password −

**Step 1** − To start, type password at the command prompt as shown below.

**Step 2** − Enter your old password, the one you're currently using.

**Step 3** − Type in your new password. Always keep your password complex enough so that nobody can guess it. But make sure, you remember it.

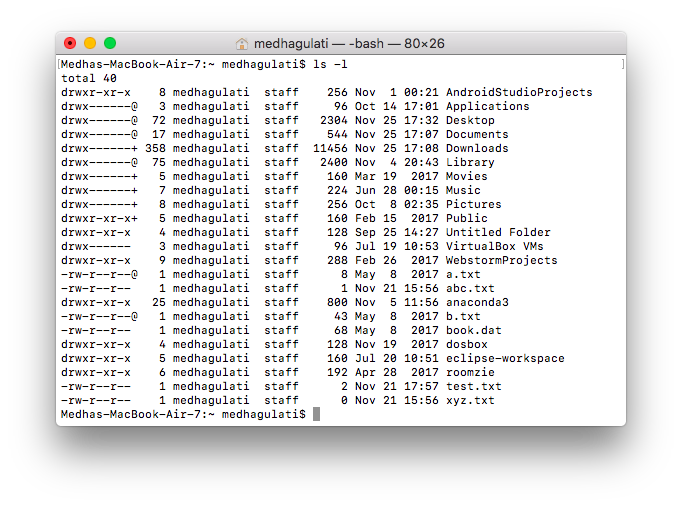
**Step 4** − You must verify the password by typing it again.



### Listing Directories and Files

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

You can use the **ls** command to list out all the files or directories available in a directory. Following is the example of using **ls** command with **-l** option.

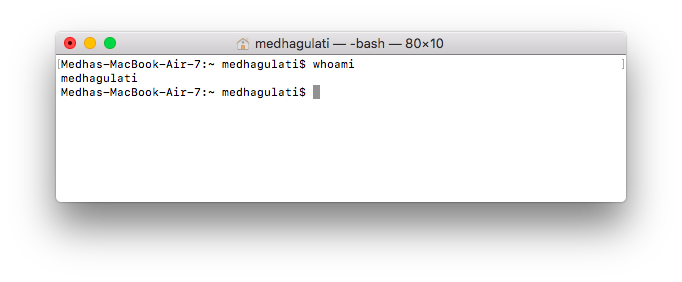


Here entries starting with **d.....** represent directories.

### Who Are You?

While you're logged into the system, you might be willing to know : **Who am I**?

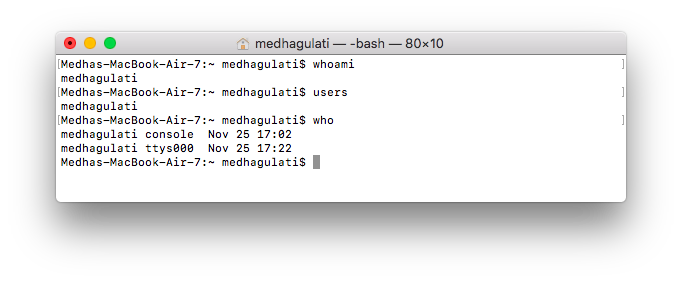
The easiest way to find out "who you are" is to enter the **whoami** command

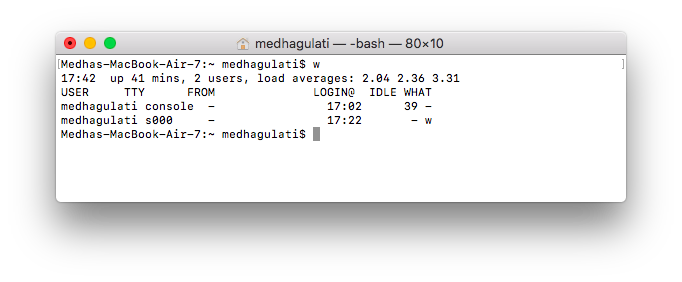


### Who is Logged in?

Sometime you might be interested to know who is logged in to the computer at the same time.

There are three commands available to get you this information, based on how much you wish to know about the other users: **users**, **who**, and **w**.





This lists down information associated with the users logged in the system.

### Logging Out

When you finish your session, you need to log out of the system. This is to ensure that nobody else accesses your files.

**To log out**

* Just type the **logout** command at the command prompt, and the system will clean up everything and break the connection.

### System Shutdown

The most consistent way to shut down a Unix system properly via the command line is to use one of the following commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **halt**  Brings the system down immediately |
| 2 | **init 0**  Powers off the system using predefined scripts to synchronize and clean up the system prior to shutting down |
| 3 | **init 6**  Reboots the system by shutting it down completely and then restarting it |
| 4 | **poweroff**  Shuts down the system by powering off |
| 5 | **reboot**  Reboots the system |
| 6 | **shutdown**  Shuts down the system |

You typically need to be the super user or root (the most privileged account on a Unix system) to shut down the system. However, on some standalone or personally-owned Unix boxes, an administrative user and sometimes regular users can do so.



EXPERIMENT 2: Discuss the Unix/Linux File Management and its associated commands.

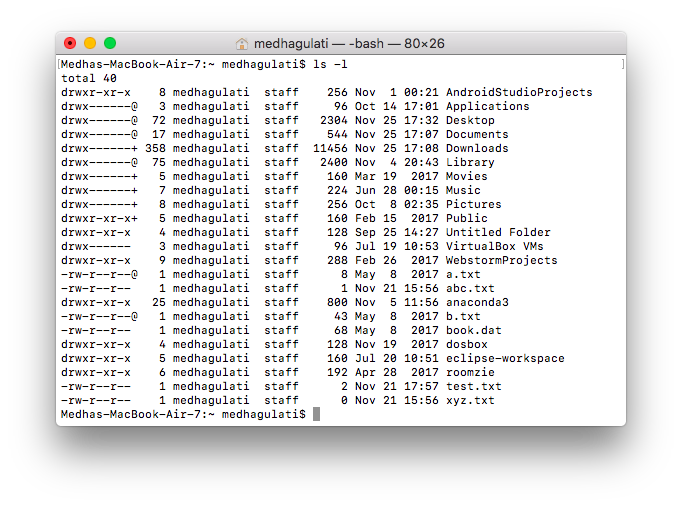
All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

In Unix, there are three basic types of files −

* **Ordinary Files** − An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
* **Directories** − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
* **Special Files** − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

## Listing Files

To list the files and directories stored in the current directory, use the following command –



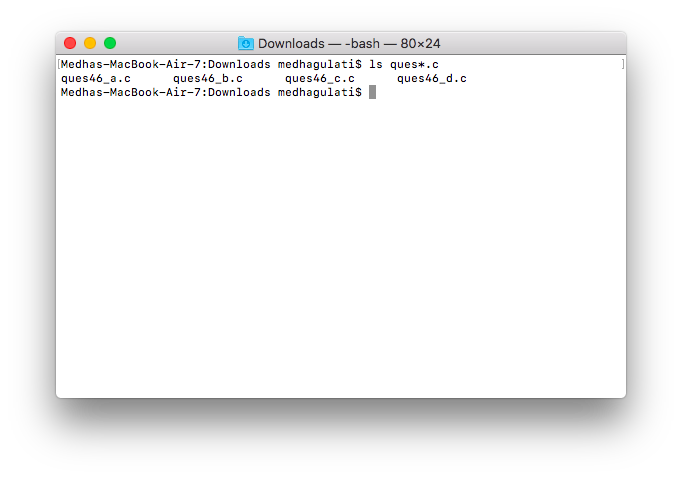
Here is the information about all the listed columns −

* **First Column** − Represents the file type and the permission given on the file. Below is the description of all type of files.
* **Second Column** − Represents the number of memory blocks taken by the file or directory.
* **Third Column** − Represents the owner of the file. This is the Unix user who created this file.
* **Fourth Column** − Represents the group of the owner. Every Unix user will have an associated group.
* **Fifth Column** − Represents the file size in bytes.
* **Sixth Column** − Represents the date and the time when this file was created or modified for the last time.
* **Seventh Column** − Represents the file or the directory name.

|  |  |
| --- | --- |
| **Sr.No.** | **Prefix & Description** |
| 1 | **-**  Regular file, such as an ASCII text file, binary executable, or hard link. |
| 2 | **b**  Block special file. Block input/output device file such as a physical hard drive. |
| 3 | **c**  Character special file. Raw input/output device file such as a physical hard drive. |
| 4 | **d**  Directory file that contains a listing of other files and directories. |
| 5 | **l**  Symbolic link file. Links on any regular file. |
| 6 | **p**  Named pipe. A mechanism for interprocess communications. |
| 7 | **s**  Socket used for interprocess communication. |

## Metacharacters

Metacharacters have a special meaning in Unix. For example, **\*** and **?** are metacharacters. We use **\*** to match 0 or more characters, a question mark (**?**) matches with a single character.

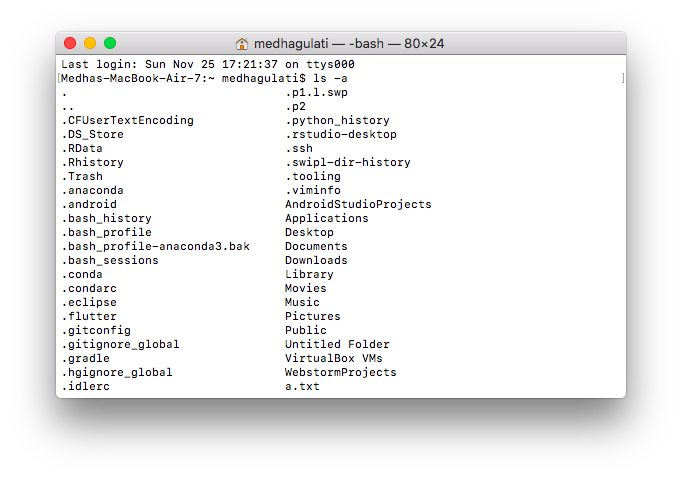


## Hidden Files

An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.

Some common examples of the hidden files include the files −

* **.profile** − The Bourne shell ( sh) initialization script
* **.kshrc** − The Korn shell ( ksh) initialization script
* **.cshrc** − The C shell ( csh) initialization script
* **.rhosts** − The remote shell configuration file



* **Single dot (.)** − This represents the current directory.
* **Double dot (..)** − This represents the parent directory.

## Creating Files

You can use the **vi** editor to create ordinary files on any Unix system. You simply need to give the following command −

$ vi filename

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode.

Once you are done with the program, follow these steps −

* Press the key **esc** to come out of the edit mode.
* Press two keys **Shift + ZZ** together to come out of the file completely.

## Editing Files

You can edit an existing file using the **vi** editor.

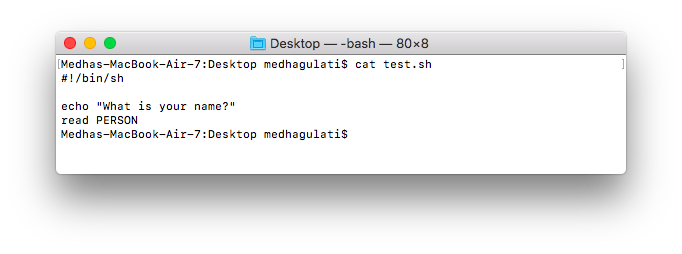
Once the file is opened, you can come in the edit mode by pressing the key **i**and then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file −

* **l** key to move to the right side.
* **h** key to move to the left side.
* **k** key to move upside in the file.
* **j** key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

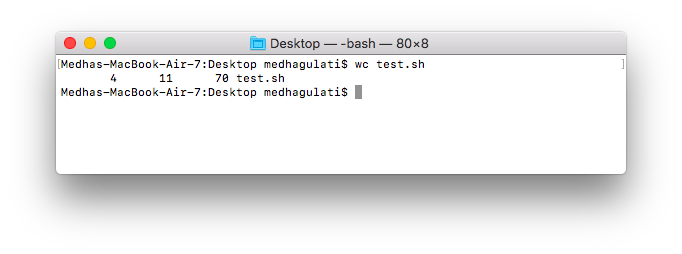
## Display Content of a File

You can use the **cat** command to see the content of a file.



## Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file.



* **First Column** − Represents the total number of lines in the file.
* **Second Column** − Represents the total number of words in the file.
* **Third Column** − Represents the total number of bytes in the file. This is the actual size of the file.
* **Fourth Column** − Represents the file name.

## Copying Files

To make a copy of a file use the **cp** command.

$ cp source\_file destination\_file



## Renaming Files

To change the name of a file, use the **mv** command.

$ mv old\_file new\_file

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

## Deleting Files

To delete an existing file, use the **rm** command. Following is the basic syntax −

$ rm filename

You can remove multiple files at a time with the command given below −

$ rm filename1 filename2 filename3 $

## Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for it when it starts up −

* **stdin** − This is referred to as the *standard input* and the associated file descriptor is 0. This is also represented as STDIN. The Unix program will read the default input from STDIN.
* **stdout** − This is referred to as the *standard output* and the associated file descriptor is 1. This is also represented as STDOUT. The Unix program will write the default output at STDOUT
* **stderr** − This is referred to as the *standard error* and the associated file descriptor is 2. This is also represented as STDERR. The Unix program will write all the error messages at STDERR.



**EXPERIMENT 3: Discuss the Unix/Linux Directory Management module and the associated commands.**

A directory is a file the solo job of which is to store the file names and the related information. All the files, whether ordinary, special, or directory, are contained in directories.

Unix uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character (**/**), and all other directories are contained below it.

## Home Directory

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command −

$cd ~

$

Here **~** indicates the home directory. Suppose you have to go in any other user's home directory, use the following command −

$cd ~username

$

To go in your last directory, you can use the following command −

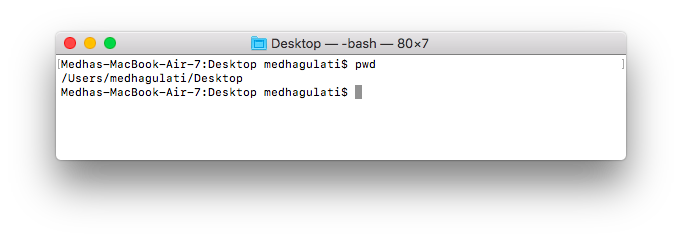
$cd -

## Absolute/Relative Pathnames

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory −



## Listing Directories

To list the files in a directory, you can use the following syntax −

$ls dirname

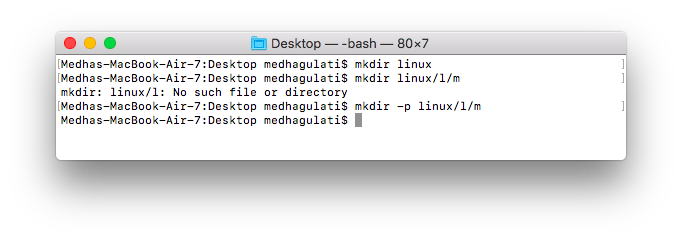
## Creating Directories

We will now understand how to create directories. Directories are created by the following command −

$mkdir dirname

## Creating Parent Directories

We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist.



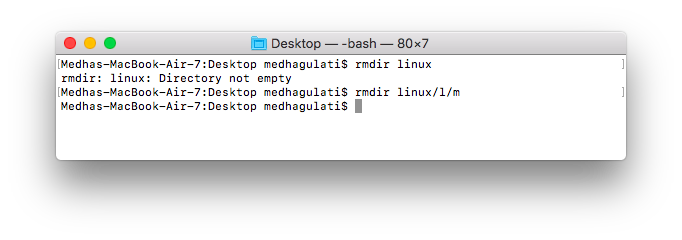
## Removing Directories

Directories can be deleted using the **rmdir** command as follows −

$rmdir dirname

You can remove multiple directories at a time as follows −

$rmdir dirname1 dirname2 dirname3



## Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below −

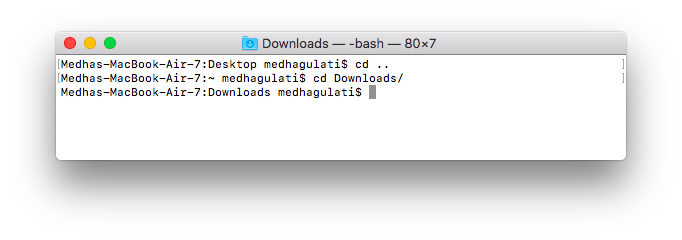
$cd dirname $

Here, **dirname** is the name of the directory that you want to change to. For example, the command −

$cd /usr/local/bin $

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path −

$cd ../../home/amrood





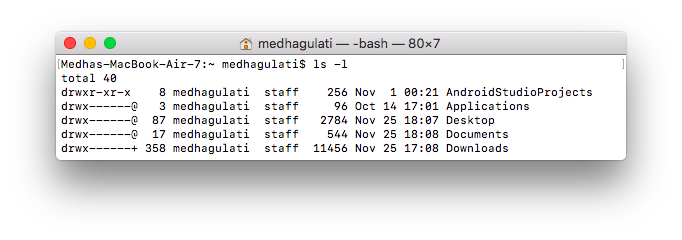
**EXPERIMENT 4:Discuss the Unix/Linux File Permission/Access Modes and the commands related to this**.

File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

* **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
* **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
* **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows –



Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

* The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
* The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--**represents that the group has read (r) and execute (x) permission, but no write permission.
* The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

## File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

### Read

Grants the capability to read, i.e., view the contents of the file.

### Write

Grants the capability to modify, or remove the content of the file.

### Execute

User with execute permissions can run a file as a program.

## Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

### Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

### Write

Access means that the user can add or delete files from the directory.

### Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

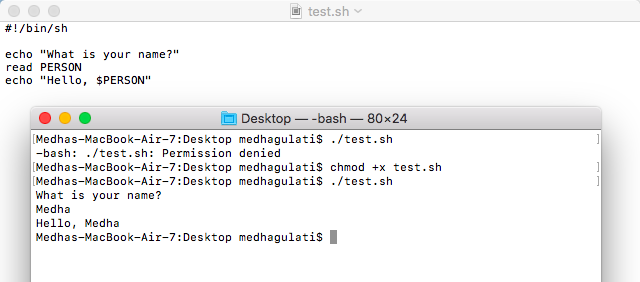
## Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

### Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

|  |  |
| --- | --- |
| **Sr.No.** | **Chmod operator & Description** |
| 1 | **+**  Adds the designated permission(s) to a file or directory. |
| 2 | **-**  Removes the designated permission(s) from a file or directory. |
| 3 | **=**  Sets the designated permission(s). |



## Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

|  |  |  |
| --- | --- | --- |
| **Number** | **Octal Permission Representation** | **Ref** |
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

## Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

* **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
* **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

$ chown user filelist

$ chgrp group filelist

## SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners’ **execute** permission normally resides.

$ ls -l /usr/bin/passwd

-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd\*

$

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

* The owner of the sticky directory
* The owner of the file being removed
* The super user, root

To set the SUID and SGID bits for any directory, the following command −

$ chmod ug+s dirname

$ ls -l

drwsr-sr-x 2 root root 4096 Jun 19 06:45 dirname



EXPERIMENT 5: Discuss the concept of Unix/Linux Environment and the commands associated with the environment and basic utilities.

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.

first we set a variable TEST and then we access its value using the **echo** command −

$TEST="Unix Programming" $echo $TEST

It produces the following result.

Unix Programming

Note that the environment variables are set without using the **$** sign but while accessing them we use the $ sign as prefix. These variables retain their values until we come out of the shell.

When you log in to the system, the shell undergoes a phase called **initialization** to set up the environment. This is usually a two-step process that involves the shell reading the following files −

* /etc/profile
* profile

The process is as follows −

* The shell checks to see whether the file **/etc/profile** exists.
* If it exists, the shell reads it. Otherwise, this file is skipped. No error message is displayed.
* The shell checks to see whether the file **.profile** exists in your home directory. Your home directory is the directory that you start out in after you log in.
* If it exists, the shell reads it; otherwise, the shell skips it. No error message is displayed.

## The .profile File

The file **/etc/profile** is maintained by the system administrator of your Unix machine and contains shell initialization information required by all users on a system.

The file **.profile** is under your control. You can add as much shell customization information as you want to this file. The minimum set of information that you need to configure includes −

* The type of terminal you are using.
* A list of directories in which to locate the commands.
* A list of variables affecting the look and feel of your terminal.

You can check your **.profile** available in your home directory. Open it using the vi editor and check all the variables set for your environment.

## Setting the Terminal Type

Usually, the type of terminal you are using is automatically configured by either the **login** or **getty** programs. Sometimes, the auto configuration process guesses your terminal incorrectly.

If your terminal is set incorrectly, the output of the commands might look strange, or you might not be able to interact with the shell properly.

To make sure that this is not the case, most users set their terminal to the lowest common denominator in the following way −

$TERM=vt100

## Setting the PATH

When you type any command on the command prompt, the shell has to locate the command before it can be executed.

The PATH variable specifies the locations in which the shell should look for commands. Usually the Path variable is set as follows −

$PATH=/bin:/usr/bin

Here, each of the individual entries separated by the colon character **(:)** are directories. If you request the shell to execute a command and it cannot find it in any of the directories given in the PATH variable, a message similar to the following appears −

$hello hello: not found

There are variables like PS1 and PS2 which are discussed in the next section.

## PS1 and PS2 Variables

The characters that the shell displays as your command prompt are stored in the variable PS1. You can change this variable to be anything you want. As soon as you change it, it'll be used by the shell from that point on.

For example, if you issued the command −

$PS1='=>' => => =>

Your prompt will become =>. To set the value of **PS1** so that it shows the working directory, issue the command −

=>PS1="[\u@\h \w]\$" [root@ip-72-167-112-17 /var/www/tutorialspoint/unix]

$ [root@ip-72-167-112-17 /var/www/tutorialspoint/unix]

$

The result of this command is that the prompt displays the user's username, the machine's name (hostname), and the working directory.

There are quite a few **escape sequences** that can be used as value arguments for PS1; try to limit yourself to the most critical so that the prompt does not overwhelm you with information.

|  |  |
| --- | --- |
| **Sr.No.** | **Escape Sequence & Description** |
| 1 | **\t**  Current time, expressed as HH:MM:SS |
| 2 | **\d**  Current date, expressed as Weekday Month Date |
| 3 | **\n**  Newline |
| 4 | **\s**  Current shell environment |
| 5 | **\W**  Working directory |
| 6 | **\w**  Full path of the working directory |
| 7 | **\u**  Current user’s username |
| 8 | **\h**  Hostname of the current machine |
| 9 | **\#**  Command number of the current command. Increases when a new command is entered |
| 10 | **\$**  If the effective UID is 0 (that is, if you are logged in as root), end the prompt with the # character; otherwise, use the $ sign |

You can make the change yourself every time you log in, or you can have the change made automatically in PS1 by adding it to your **.profile** file.

When you issue a command that is incomplete, the shell will display a secondary prompt and wait for you to complete the command and hit **Enter**again.

The default secondary prompt is **>** (the greater than sign), but can be changed by re-defining the **PS2** shell variable −

Following is the example which uses the default secondary prompt −

$ echo "this is a > test" this is a test

$

The example given below re-defines PS2 with a customized prompt −

$ PS2="secondary prompt->"

$ echo "this is a secondary prompt->test" this is a test

$

## Environment Variables

Following is the partial list of important environment variables. These variables are set and accessed as mentioned below −

|  |  |
| --- | --- |
| **Sr.No.** | **Variable & Description** |
| 1 | **DISPLAY**  Contains the identifier for the display that **X11** programs should use by default. |
| 2 | **HOME**  Indicates the home directory of the current user: the default argument for the cd **built-in** command. |
| 3 | **IFS**  Indicates the **Internal Field Separator** that is used by the parser for word splitting after expansion. |
| 4 | **LANG**  LANG expands to the default system locale; LC\_ALL can be used to override this. For example, if its value is **pt\_BR**, then the language is set to (Brazilian) Portuguese and the locale to Brazil. |
| 5 | **LD\_LIBRARY\_PATH**  A Unix system with a dynamic linker, contains a colonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories. |
| 6 | **PATH**  Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands. |
| 7 | **PWD**  Indicates the current working directory as set by the cd command. |
| 8 | **RANDOM**  Generates a random integer between 0 and 32,767 each time it is referenced. |
| 9 | **SHLVL**  Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session. |
| 10 | **TERM**  Refers to the display type. |
| 11 | **TZ**  Refers to Time zone. It can take values like GMT, AST, etc. |
| 12 | **UID**  Expands to the numeric user ID of the current user, initialized at the shell startup. |

Following is the sample example showing few environment variables −

$ echo $HOME /root ]

$ echo $DISPLAY

$ echo $TERM xterm

$ echo $PATH /usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin

## Printing Files

Before you print a file on a Unix system, you may want to reformat it to adjust the margins, highlight some words, and so on. Most files can also be printed without reformatting, but the raw printout may not be that appealing.

Many versions of Unix include two powerful text formatters, **nroff** and **troff**.

### The pr Command

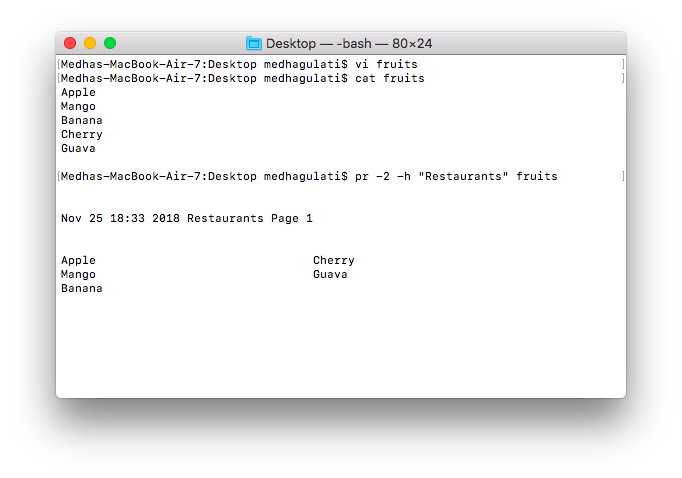
The **pr** command does minor formatting of files on the terminal screen or for a printer. For example, if you have a long list of names in a file, you can format it onscreen into two or more columns.

Following is the syntax for the **pr** command −

pr option(s) filename(s)

The **pr** changes the format of the file only on the screen or on the printed copy; it doesn't modify the original file. Following table lists some **pr** options −

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **-k**  Produces **k** columns of output |
| 2 | **-d**  Double-spaces the output (not on all **pr** versions) |
| 3 | **-h "header"**  Takes the next item as a report header |
| 4 | **-t**  Eliminates the printing of header and the top/bottom margins |
| 5 | **-l PAGE\_LENGTH**  Sets the page length to PAGE\_LENGTH (66) lines. The default number of lines of text is 56 |
| 6 | **-o MARGIN**  Offsets each line with MARGIN (zero) spaces |
| 7 | **-w PAGE\_WIDTH**  Sets the page width to PAGE\_WIDTH (72) characters for multiple text-column output only |



### The lp and lpr Commands

The command **lp** or **lpr** prints a file onto paper as opposed to the screen display. Once you are ready with formatting using the **pr** command, you can use any of these commands to print your file on the printer connected to your computer.

Your system administrator has probably set up a default printer at your site.

The **lp** command shows an ID that you can use to cancel the print job or check its status.

* If you are using the **lp** command, you can use the -n**Num** option to print Num number of copies. Along with the command **lpr**, you can use -**Num** for the same.
* If there are multiple printers connected with the shared network, then you can choose a printer using -d**printer** option along with lp command and for the same purpose you can use -P**printer** option along with lpr command. Here printer is the printer name.

### The lpstat and lpq Commands

The **lpstat** command shows what's in the printer queue: request IDs, owners, file sizes, when the jobs were sent for printing, and the status of the requests.

Use **lpstat -o** if you want to see all output requests other than just your own.

### The cancel and lprm Commands

The **cancel** command terminates a printing request from the **lp command**. The **lprm** command terminates all **lpr requests**. You can specify either the ID of the request (displayed by lp or lpq) or the name of the printer.

$cancel laserp-575 request "laserp-575" cancelled $

To cancel whatever request is currently printing, regardless of its ID, simply enter cancel and the printer name −

$cancel laserp request "laserp-573" cancelled $

The **lprm** command will cancel the active job if it belongs to you. Otherwise, you can give job numbers as arguments, or use a **dash (-)** to remove all of your jobs −

$lprm 575 dfA575diamond dequeued cfA575diamond dequeued $

The **lprm** command tells you the actual filenames removed from the printer queue.

## Sending Email

You use the Unix mail command to send and receive mail. Here is the syntax to send an email −

$mail [-s subject] [-c cc-addr] [-b bcc-addr] to-addr

Here are important options related to mail command −s

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **-s**  Specifies subject on the command line. |
| 2 | **-c**  Sends carbon copies to the list of users. List should be a commaseparated list of names. |
| 3 | **-b**  Sends blind carbon copies to list. List should be a commaseparated list of names. |

Following is an example to send a test message to admin@yahoo.com.

$mail -s "Test Message" admin@yahoo.com

To check incoming email at your Unix system, you simply type email as follows −

$mail no email



**6. Discuss the concept of Unix/Linux Pipes and Filters and the commands associated with this.**

We can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, we put a vertical bar (**|**) on the command line between two commands.

When a program takes its input from another program, it performs some operation on that input, and writes the result to the standard output. It is referred to as a ***filter***.

The grep Command

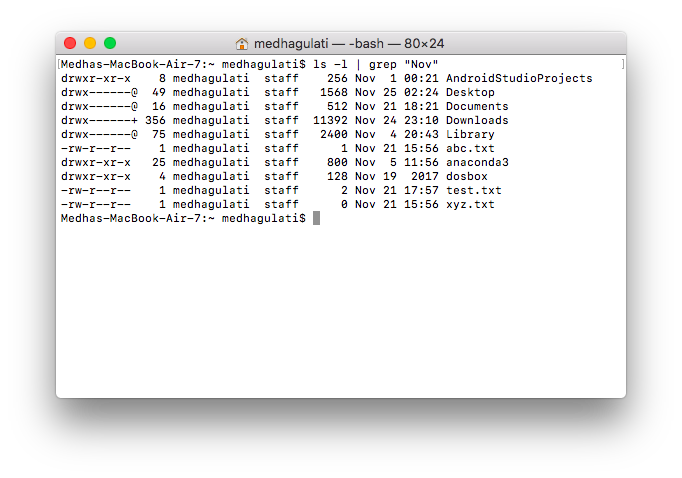
The grep command searches a file or files for lines that have a certain pattern. The syntax is −

$grep pattern file(s)

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

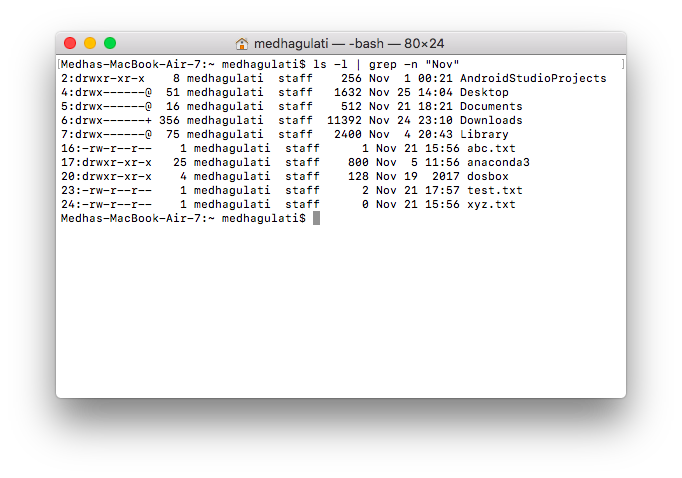
A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

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



There are various options which you can use along with the **grep** command −

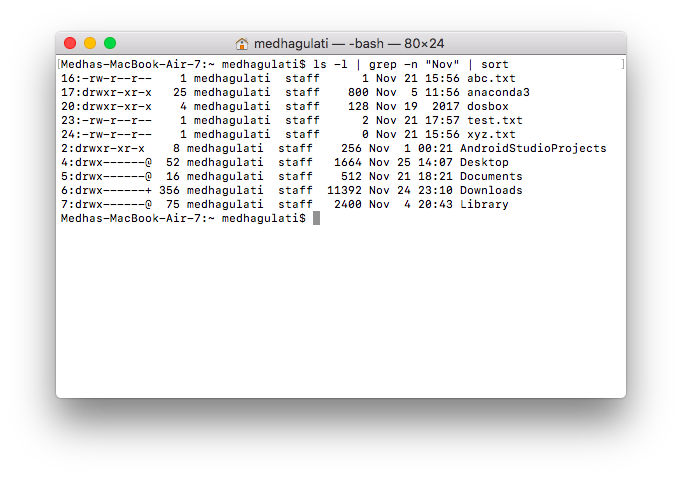
|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **-v**  Prints all lines that do not match pattern. |
| 2 | **-n**  Prints the matched line and its line number. |
| 3 | **-l**  Prints only the names of files with matching lines (letter "l") |
| 4 | **-c**  Prints only the count of matching lines. |
| 5 | **-i**  Matches either upper or lowercase. |



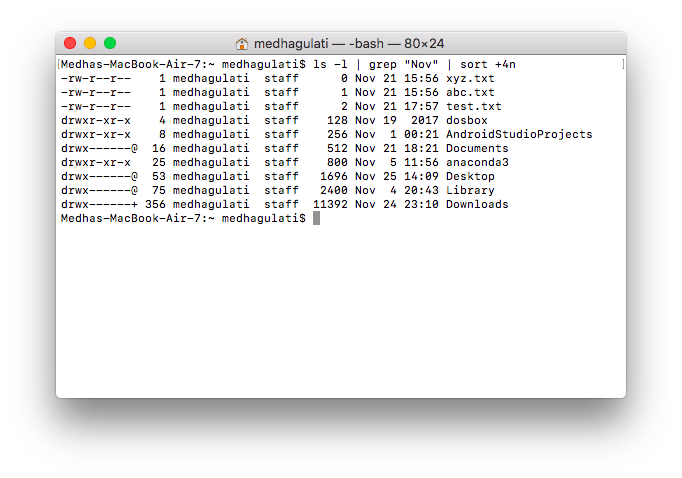
## The sort Command

The **sort** command arranges lines of text alphabetically or numerically.

More than two commands may be linked up into a pipe.



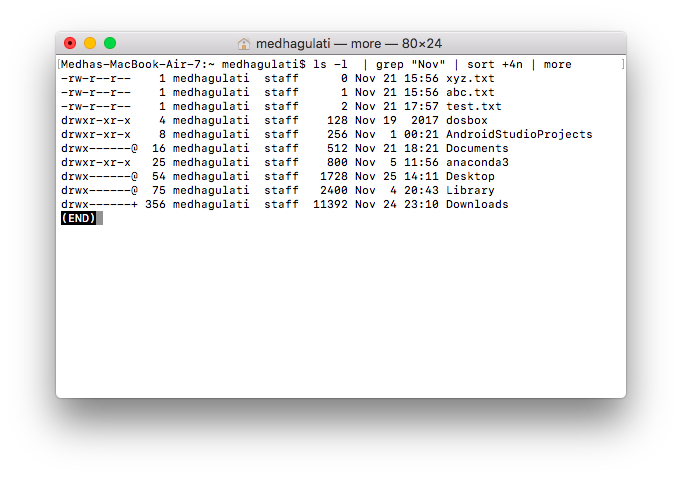
|  |  |
| --- | --- |
| **Sr.No.** | **Description** |
| 1 | **-n**  Sorts numerically (example: 10 will sort after 2), ignores blanks and tabs. |
| 2 | **-r**  Reverses the order of sort. |
| 3 | **-f**  Sorts upper and lowercase together. |
| 4 | **+x**  Ignores first **x** fields when sorting. |



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

## The pg and more Commands

A long output can normally be zipped on the screen, but if we run text through more or use the **pg** command as a filter; the display stops once the screen is full of text.





**7. Discuss the importance of Unix/Linux Process Management and the associated commands.**

When we execute a program on our Unix system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

Whenever we issue a command in Unix, it creates, or starts, a new process. When we tried out the **ls** command to list the directory contents, we started a process. A process, in simple terms, is an instance of a running program.

The operating system tracks processes through a five-digit ID number known as the **pid** or the **process ID**. Each process in the system has a unique **pid**.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

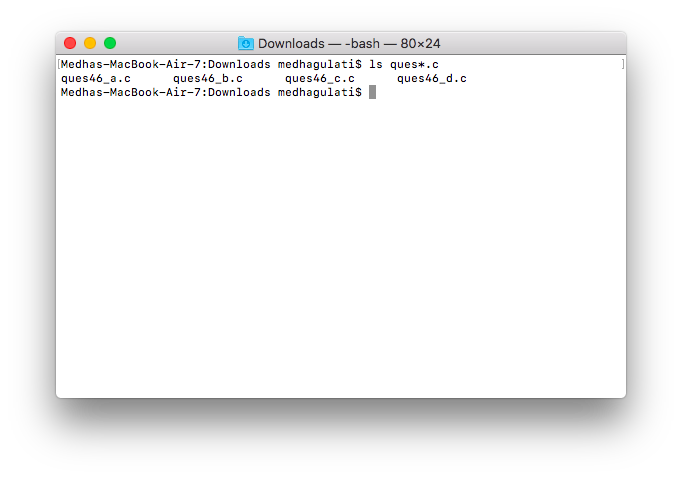
## Starting a Process

When you start a process (run a command), there are two ways you can run it −

* Foreground Processes
* Background Processes

### Foreground Processes

By default, every process that we start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.



The process runs in the foreground, the output is directed to my screen, and if the **ls** command wants any input (which it does not), it waits for it from the keyboard.

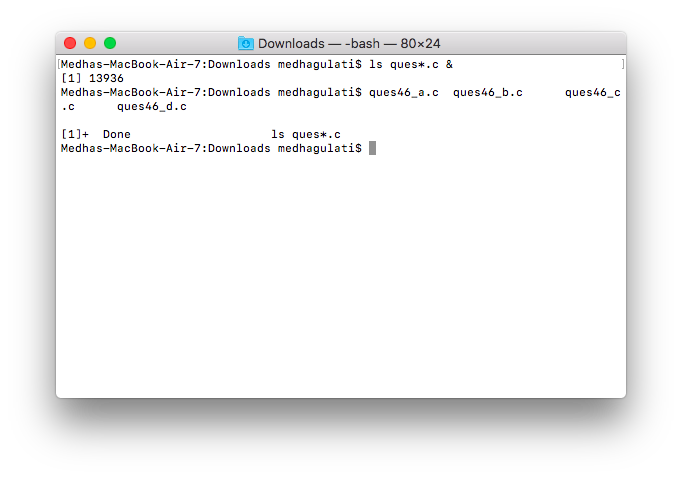
While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

### Background Processes

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand (**&**) at the end of the command.



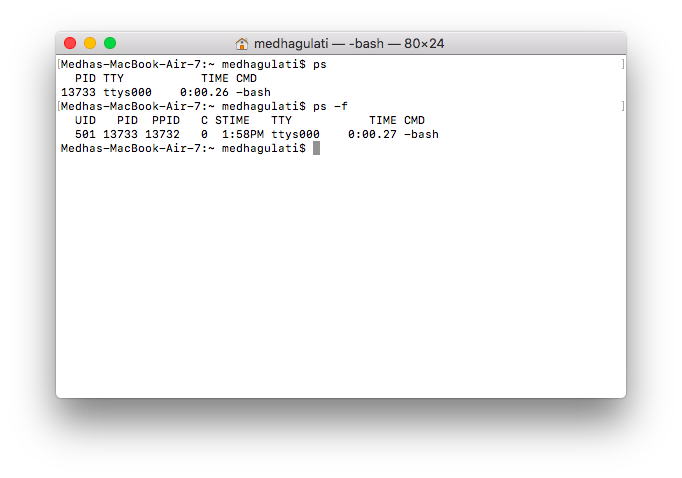
Here, if the **ls** command wants any input (which it does not), it goes into a stop state until we move it into the foreground and give it the data from the keyboard.

That first line contains information about the background process - the job number and the process ID. You need to know the job number to manipulate it between the background and the foreground.

## Listing Running Processes

It is easy to see your own processes by running the **ps** (process status) command

One of the most commonly used flags for ps is the **-f** ( f for full) option, which provides more information



|  |  |
| --- | --- |
| **Sr.No.** | **Column & Description** |
| 1 | **UID**  User ID that this process belongs to (the person running it) |
| 2 | **PID**  Process ID |
| 3 | **PPID**  Parent process ID (the ID of the process that started it) |
| 4 | **C**  CPU utilization of process |
| 5 | **STIME**  Process start time |
| 6 | **TTY**  Terminal type associated with the process |
| 7 | **TIME**  CPU time taken by the process |
| 8 | **CMD**  The command that started this process |

There are other options which can be used along with **ps** command −

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **-a**  Shows information about all users |
| 2 | **-x**  Shows information about processes without terminals |
| 3 | **-u**  Shows additional information like -f option |
| 4 | **-e**  Displays extended information |

## Stopping Processes

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when the process is running in the foreground mode.

If a process is running in the background, you should get its Job ID using the **ps** command. After that, you can use the **kill** command to kill the process.

## Parent and Child Processes

Each unix process has two ID numbers assigned to it: The Process ID (pid) and the Parent process ID (ppid). Each user process in the system has a parent process.

Most of the commands that you run have the shell as their parent. Check the **ps -f** example where this command listed both the process ID and the parent process ID.

## Zombie and Orphan Processes

Normally, when a child process is killed, the parent process is updated via a **SIGCHLD** signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," the **init** process, becomes the new PPID (parent process ID). In some cases, these processes are called orphan processes.

When a process is killed, a **ps** listing may still show the process with a **Z** state. This is a zombie or defunct process. The process is dead and not being used. These processes are different from the orphan processes. They have completed execution but still find an entry in the process table.

## Daemon Processes

Daemons are system-related background processes that often run with the permissions of root and services requests from other processes.

A daemon has no controlling terminal. It cannot open **/dev/tty**. If you do a **"ps -ef"** and look at the **tty** field, all daemons will have a **?** for the **tty**.

To be precise, a daemon is a process that runs in the background, usually waiting for something to happen that it is capable of working with. For example, a printer daemon waiting for print commands.

If you have a program that calls for lengthy processing, then it’s worth to make it a daemon and run it in the background.

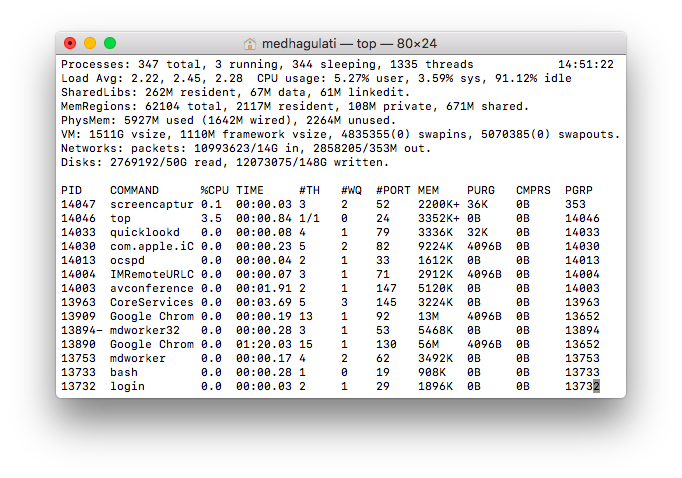
## The top Command

The **top** command is a very useful tool for quickly showing processes sorted by various criteria.

It is an interactive diagnostic tool that updates frequently and shows information about physical and virtual memory, CPU usage, load averages, and your busy processes.

The simple syntax to run top command and to see the statistics of CPU utilization by different processes −

$top



## Job ID Versus Process ID

Background and suspended processes are usually manipulated via **job number (job ID)**. This number is different from the process ID and is used because it is shorter.

In addition, a job can consist of multiple processes running in a series or at the same time, in parallel. Using the job ID is easier than tracking individual processes.



**8. Discuss the Unix/Linux Network Communication Utilities and the associated commands.**

When we work in a distributed environment, we need to communicate with remote users and we also need to access remote Unix machines.

There are several Unix utilities that help users compute in a networked, distributed environment.

## The ping Utility

The **ping** command sends an echo request to a host available on the network. Using this command, we can check if your remote host is responding well or not.

The ping command is useful for the following −

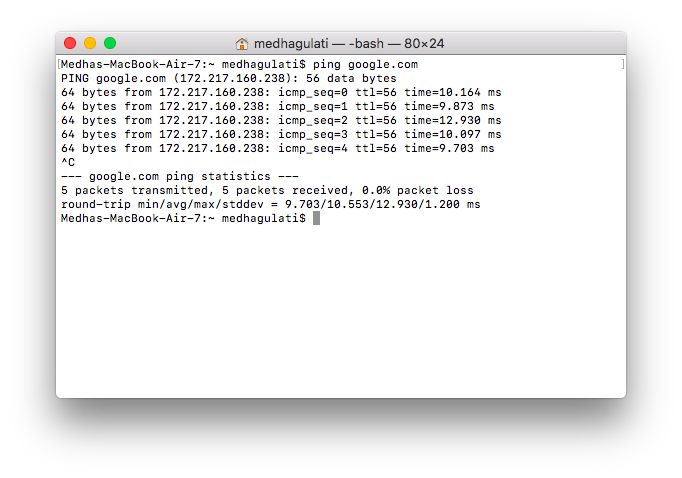
* Tracking and isolating hardware and software problems.
* Determining the status of the network and various foreign hosts.
* Testing, measuring, and managing networks.

### Syntax

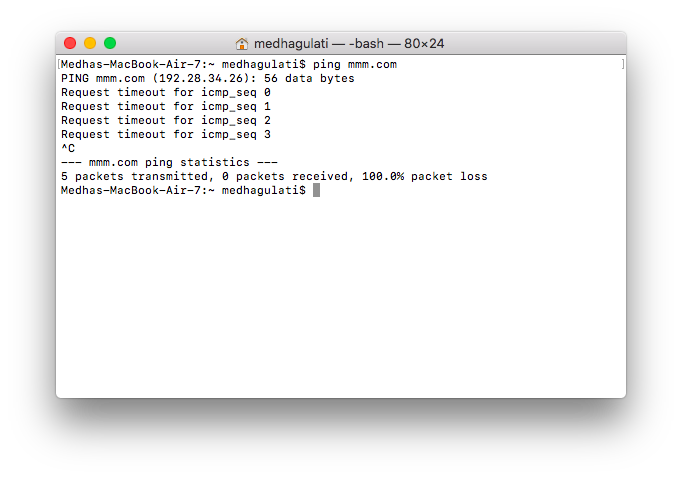
Following is the simple syntax to use the ping command −

$ping hostname or ip-address

The above command starts printing a response after every second. To come out of the command, you can terminate it by pressing **CNTRL + C** keys.



If a host does not exist, we will receive the following output −



## The ftp Utility

Here, **ftp** stands for **F**ile **T**ransfer **P**rotocol. This utility helps you upload and download your file from one computer to another computer.

The ftp utility has its own set of Unix-like commands. These commands help you perform tasks such as −

* Connect and login to a remote host.
* Navigate directories.
* List directory contents.
* Put and get files.
* Transfer files as **ascii**, **ebcdic** or **binary**.

$ftp hostname or ip-address

This command would prompt for the login ID and the password. Once we are authenticated, we can access the home directory of the login account and we would be able to perform various commands.

The following tables lists out a few important commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **put filename**  Uploads filename from the local machine to the remote machine. |
| 2 | **get filename**  Downloads filename from the remote machine to the local machine. |
| 3 | **mput file list**  Uploads more than one file from the local machine to the remote machine. |
| 4 | **mget file list**  Downloads more than one file from the remote machine to the local machine. |
| 5 | **prompt off**  Turns the prompt off. By default, you will receive a prompt to upload or download files using **mput** or **mget** commands. |
| 6 | **prompt on**  Turns the prompt on. |
| 7 | **dir**  Lists all the files available in the current directory of the remote machine. |
| 8 | **cd dirname**  Changes directory to dirname on the remote machine. |
| 9 | **lcd dirname**  Changes directory to dirname on the local machine. |
| 10 | **quit**  Helps logout from the current login. |

It should be noted that all the files would be downloaded or uploaded to or from the current directories. If we want to upload our files in a particular directory, we need to first change to that directory and then upload the required files.

## The telnet Utility

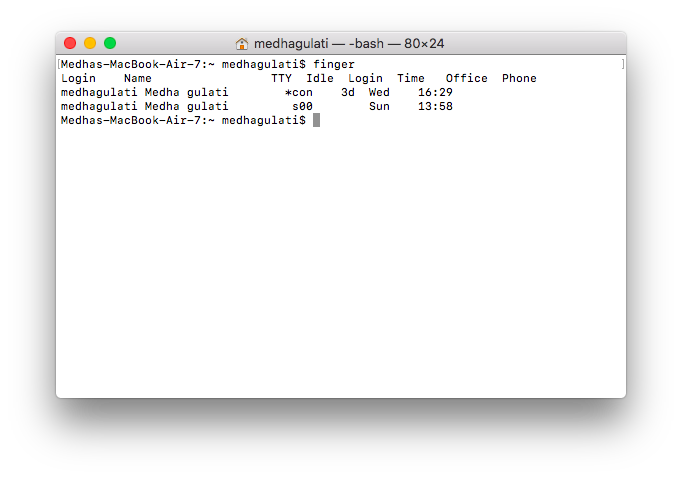
There are times when we are required to connect to a remote Unix machine and work on that machine remotely. **Telnet** is a utility that allows a computer user at one site to make a connection, login and then conduct work on a computer at another site.

Once you login using Telnet, you can perform all the activities on your remotely connected machine.

## The finger Utility

The **finger** command displays information about users on a given host. The host can be either local or remote.

Finger may be disabled on other systems for security reasons.





**9. Perform basic operations in Unix/Linux vi Editor.**

There are many ways to edit files in Unix. Editing files using the screen-oriented text editor **vi** is one of the best ways. This editor enables you to edit lines in context with other lines in the file.

An improved version of the vi editor which is called the **VIM** has also been made available now. Here, VIM stands for **Vi IM**proved.

vi is generally considered the de facto standard in Unix editors because −

* It's usually available on all the flavors of Unix system.
* Its implementations are very similar across the board.
* It requires very few resources.
* It is more user-friendly than other editors such as the **ed** or the **ex**.

We can use the **vi** editor to edit an existing file or to create a new file from scratch. We can also use this editor to just read a text file.

## Starting the vi Editor

The following table lists out the basic commands to use the vi editor −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **vi filename**  Creates a new file if it already does not exist, otherwise opens an existing file. |
| 2 | **vi -R filename**  Opens an existing file in the read-only mode. |
| 3 | **view filename**  Opens an existing file in the read-only mode. |

Following is an example to create a new file **testfile** if it already does not exist in the current working directory −

$vi testfile

The above command will generate the following output −



A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

## Operation Modes

While working with the vi editor, we usually come across the following two modes −

* **Command mode** − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
* **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.

vi always starts in the **command mode**. 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.

## Getting Out of vi

The command to quit out of vi is **:q**. Once in the command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or use **:wq** and return.

The easiest way to **save your changes and exit vi** is with the ZZ command. When you are in the command mode, type **ZZ**. The **ZZ** command works the same way as the **:wq** command.

If you want to specify/state any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type **:w filename2** and return.

## Moving within a File

To move around within a file without affecting your text, you must be in the command mode (press Esc twice). The following table lists out a few commands you can use to move around one character at a time −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **k**  Moves the cursor up one line |
| 2 | **j**  Moves the cursor down one line |
| 3 | **h**  Moves the cursor to the left one character position |
| 4 | **l**  Moves the cursor to the right one character position |

The following points need to be considered to move within a file −

* vi is case-sensitive. You need to pay attention to capitalization when using the commands.
* Most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2j** moves the cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Remember that you must be in the command mode (**press Esc twice**). The following table lists out a few commands to move around the file −

Given below is the list of commands to move around the file.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **0 or |**  Positions the cursor at the beginning of a line |
| 2 | **$**  Positions the cursor at the end of a line |
| 3 | **w**  Positions the cursor to the next word |
| 4 | **b**  Positions the cursor to the previous word |
| 5 | **(**  Positions the cursor to the beginning of the current sentence |
| 6 | **)**  Positions the cursor to the beginning of the next sentence |
| 7 | **E**  Moves to the end of the blank delimited word |
| 8 | **{**  Moves a paragraph back |
| 9 | **}**  Moves a paragraph forward |
| 10 | **[[**  Moves a section back |
| 11 | **]]**  Moves a section forward |
| 12 | **n|**  Moves to the column **n** in the current line |
| 13 | **1G**  Moves to the first line of the file |
| 14 | **G**  Moves to the last line of the file |
| 15 | **nG**  Moves to the **nth** line of the file |
| 16 | **:n**  Moves to the **nth** line of the file |
| 17 | **fc**  Moves forward to **c** |
| 18 | **Fc**  Moves back to **c** |
| 19 | **H**  Moves to the top of the screen |
| 20 | **nH**  Moves to the **nth** line from the top of the screen |
| 21 | **M**  Moves to the middle of the screen |
| 22 | **L**  Move to the bottom of the screen |
| 23 | **nL**  Moves to the **nth** line from the bottom of the screen |
| 24 | **:x**  Colon followed by a number would position the cursor on the line number represented by **x** |

## Control Commands

The following commands can be used with the Control Key to performs functions as given in the table below −

Given below is the list of control commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **CTRL+d**  Moves forward 1/2 screen |
| 2 | **CTRL+f**  Moves forward one full screen |
| 3 | **CTRL+u**  Moves backward 1/2 screen |
| 4 | **CTRL+b**  Moves backward one full screen |
| 5 | **CTRL+e**  Moves the screen up one line |
| 6 | **CTRL+y**  Moves the screen down one line |
| 7 | **CTRL+u**  Moves the screen up 1/2 page |
| 8 | **CTRL+d**  Moves the screen down 1/2 page |
| 9 | **CTRL+b**  Moves the screen up one page |
| 10 | **CTRL+f**  Moves the screen down one page |
| 11 | **CTRL+I**  Redraws the screen |

## Editing Files

To edit the file, you need to be in the insert mode. There are many ways to enter the insert mode from the command mode −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **i**  Inserts text before the current cursor location |
| 2 | **I**  Inserts text at the beginning of the current line |
| 3 | **a**  Inserts text after the current cursor location |
| 4 | **A**  Inserts text at the end of the current line |
| 5 | **o**  Creates a new line for text entry below the cursor location |
| 6 | **O**  Creates a new line for text entry above the cursor location |

## Deleting Characters

Here is a list of important commands, which can be used to delete characters and lines in an open file −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **x**  Deletes the character under the cursor location |
| 2 | **X**  Deletes the character before the cursor location |
| 3 | **dw**  Deletes from the current cursor location to the next word |
| 4 | **d^**  Deletes from the current cursor position to the beginning of the line |
| 5 | **d$**  Deletes from the current cursor position to the end of the line |
| 6 | **D**  Deletes from the cursor position to the end of the current line |
| 7 | **dd**  Deletes the line the cursor is on |

As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2x** deletes two characters under the cursor location and **2dd** deletes two lines the cursor is on.

It is recommended that the commands are practiced before we proceed further.

## Change Commands

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **cc**  Removes the contents of the line, leaving you in insert mode. |
| 2 | **cw**  Changes the word the cursor is on from the cursor to the lowercase **w** end of the word. |
| 3 | **r**  Replaces the character under the cursor. vi returns to the command mode after the replacement is entered. |
| 4 | **R**  Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting. |
| 5 | **s**  Replaces the current character with the character you type. Afterward, you are left in the insert mode. |
| 6 | **S**  Deletes the line the cursor is on and replaces it with the new text. After the new text is entered, vi remains in the insert mode. |

## Copy and Paste Commands

You can copy lines or words from one place and then you can paste them at another place using the following commands −

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **yy**  Copies the current line. |
| 2 | **yw**  Copies the current word from the character the lowercase w cursor is on, until the end of the word. |
| 3 | **p**  Puts the copied text after the cursor. |
| 4 | **P**  Puts the yanked text before the cursor. |

## Advanced Commands

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

Given below is the list advanced commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **J**  Joins the current line with the next one. A count of j commands join many lines. |
| 2 | **<<**  Shifts the current line to the left by one shift width. |
| 3 | **>>**  Shifts the current line to the right by one shift width. |
| 4 | **~**  Switches the case of the character under the cursor. |
| 5 | **^G**  Press Ctrl and G keys at the same time to show the current filename and the status. |
| 6 | **U**  Restores the current line to the state it was in before the cursor entered the line. |
| 7 | **u**  This helps undo the last change that was done in the file. Typing 'u' again will re-do the change. |
| 8 | **J**  Joins the current line with the next one. A count joins that many lines. |
| 9 | **:f**  Displays the current position in the file in % and the file name, the total number of file. |
| 10 | **:f filename**  Renames the current file to filename. |
| 11 | **:w filename**  Writes to file filename. |
| 12 | **:e filename**  Opens another file with filename. |
| 13 | **:cd dirname**  Changes the current working directory to dirname. |
| 14 | **:e #**  Toggles between two open files. |
| 15 | **:n**  In case you open multiple files using vi, use **:n** to go to the next file in the series. |
| 16 | **:p**  In case you open multiple files using vi, use **:p** to go to the previous file in the series. |
| 17 | **:N**  In case you open multiple files using vi, use **:N** to go to the previous file in the series. |
| 18 | **:r file**  Reads file and inserts it after the current line. |
| 19 | **:nr file**  Reads file and inserts it after the line **n**. |

## Word and Character Searching

The vi editor has two kinds of searches: **string** and **character**. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the last line of the screen, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place −

* The **/** command searches forwards (downwards) in the file.
* The **?** command searches backwards (upwards) in the file.

The **n** and **N** commands repeat the previous search command in the same or the opposite direction, respectively. Some characters have special meanings. These characters must be preceded by a backslash (**\**) to be included as part of the search expression.

|  |  |
| --- | --- |
| **Sr.No.** | **Character &Description** |
| 1 | **^**  Searches at the beginning of the line (Use at the beginning of a search expression). |
| 2 | **.**  Matches a single character. |
| 3 | **\***  Matches zero or more of the previous character. |
| 4 | **$**  End of the line (Use at the end of the search expression). |
| 5 | **[**  Starts a set of matching or non-matching expressions. |
| 6 | **<**  This is put in an expression escaped with the backslash to find the ending or the beginning of a word. |
| 7 | **>**  This helps see the '**<**' character description above. |

The character search searches within one line to find a character entered after the command. The **f** and **F** commands search for a character on the current line only. **f** searches forwards and **F** searches backwards and the cursor moves to the position of the found character.

The **t** and **T** commands search for a character on the current line only, but for **t**, the cursor moves to the position before the character, and **T** searches the line backwards to the position after the character.

## Set Commands

You can change the look and feel of your vi screen using the following **:set**commands. Once you are in the command mode, type **:set** followed by any of the following commands.

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **:set ic**  Ignores the case when searching |
| 2 | **:set ai**  Sets autoindent |
| 3 | **:set noai**  Unsets autoindent |
| 4 | **:set nu**  Displays lines with line numbers on the left side |
| 5 | **:set sw**  Sets the width of a software tabstop. For example, you would set a shift width of 4 with this command — **:set sw = 4** |
| 6 | **:set ws**  If *wrapscan* is set, and the word is not found at the bottom of the file, it will try searching for it at the beginning |
| 7 | **:set wm**  If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: **:set wm = 2** |
| 8 | **:set ro**  Changes file type to "read only" |
| 9 | **:set term**  Prints terminal type |
| 10 | **:set bf**  Discards control characters from input |

## Running Commands

The vi has the capability to run commands from within the editor. To run a command, you only need to go to the command mode and type **:!** command.

For example, if you want to check whether a file exists before you try to save your file with that filename, you can type **:! ls** and you will see the output of **ls** on the screen.

You can press any key (or the command's escape sequence) to return to your vi session.

## Replacing Text

The substitution command (**:s/**) enables you to quickly replace words or groups of words within your files. Following is the syntax to replace text −

:s/search/replace/g

The **g** stands for globally. The result of this command is that all occurrences on the cursor's line are changed.



**10. Introduction to Shell Scripting in Unix/Linux environment.**

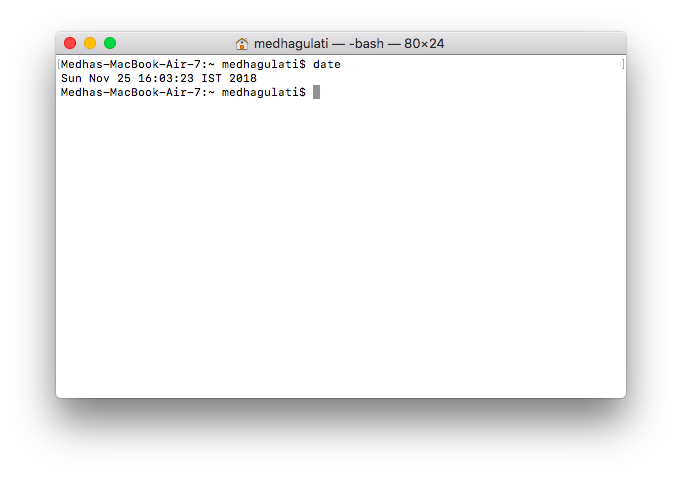
A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

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

## Shell Prompt

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press **Enter**. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.



## Shell Types

In Unix, there are two major types of shells −

* **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
* **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

* Bourne shell (sh)
* Korn shell (ksh)
* Bourne Again shell (bash)
* POSIX shell (sh)

The different C-type shells follow −

* C shell (csh)
* TENEX/TOPS C shell (tcsh)

The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as **/bin/sh** on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

## Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by **#**sign, describing the steps.

Shell scripts and functions are both interpreted. This means they are not compiled.

## Example Script

Assume we create a **test.sh** script. Note all the scripts would have the **.sh** extension. Before we add anything else to our script, we need to alert the system that a shell script is being started. This is done using the **shebang** construct. For example −

#!/bin/sh

This tells the system that the commands that follow are to be executed by the Bourne shell. *It's called a shebang because the****#****symbol is called a hash, and the ! symbol is called a bang*.

