Operating Systems

The structure of a Computer system can be visualized as having four basic components:

1. Hardware – Provides basic computing resources - CPU, memory, I/O devices

2. Operating system – Controls and coordinates use of hardware among various applications and users

3. Application programs – Define the ways in which the system resources are used to solve the computing problems of the users

4. Users – Anybody who desires work to be done by a computer system. People, machines, other computers

Types of OS

OS are classified into following different types depending on their capability of processing.

Single User and Single Tasking OS:

These are simple operating system designed to manage one task at a time ,for use by a single user for a standalone single computer for performing a single task .

Single User and Multitasking OS:

These OS allow execution of more than one task or process concurrently by dividing the processor time amongst different tasks.

Multi-programming OS:

These OS allow more than one programs to run at the same time .

Real Time OS:

These are designed to respond to an event within a predetermined time. These operating systems are used to control processes

Embedded OS:

Embedded in a device in the ROM. They are specific to a device and are less resource intensive.

Function Of OS

The OS performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating networking, and managing files

Introduction to UNIX OS

- Unix is an OS for Programmers as shell(the command interpreter)provides the programming facility.
- It provides an in-built security mechanism through the user name and password, combined with the access rights associated with files
- Developed by Ken Thompson and Ritchie originally in assembly, and later in C, thus making it portable to other machines

Supports C, Fortran, Basic, Pascal, COBOL, Lisp, Prolog, Java, Ada compilers

Features of Unix Operating System

• **Multi-user and Multitasking**

• Everything is a file

• Configuration data stored in text

• Small, single purpose programs

• Ability to chain programs together to perform complex task

• Facility of background processing

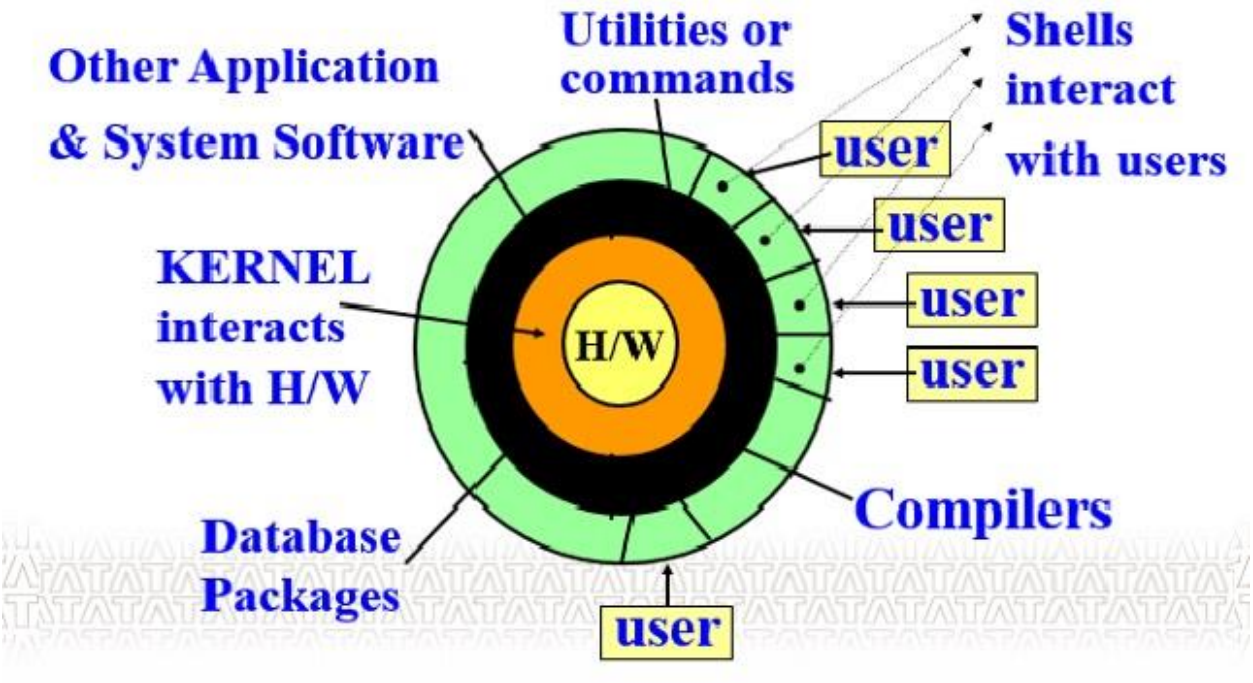
Architecture of the UNIX System

The UNIX operating system has a layered architecture having three main components

• Kernel

• Shell (command interpreter)

• Utilities (performs single tasks)



The Unix Kernel

Kernel is a collection of programs mostly written in C which allocate the system resources and coordinate all the details of the computer's internals.

Functions of Kernel:

- It allocates time and memory to programs and handles the file store and communications
- Interacts directly with the hardware through device drivers
- Provides sets of services to programs
- Manages memory, controls access, maintains file system, handles interrupts, allocates resources of the computer

System calls

The system calls are functions used in the kernel itself. UNIX system calls are used to manage the file system, control processes, and to provide interprocess communication.

System calls can be categorized as:

- File structure related calls -For example create, open,read, write,lseek,dup etc.
- Process Related calls -For example fork,exec,wait,exit etc
- Inter process related calls - For example pipe,msgget,msgsnd etc

The Unix File System:

Unix File System is a hierarchical collection of 3 types of files:

- Ordinary Files
- Directory Files
- Special Files (device, pipe, fifo, socket).

Characteristics of Unix Files:

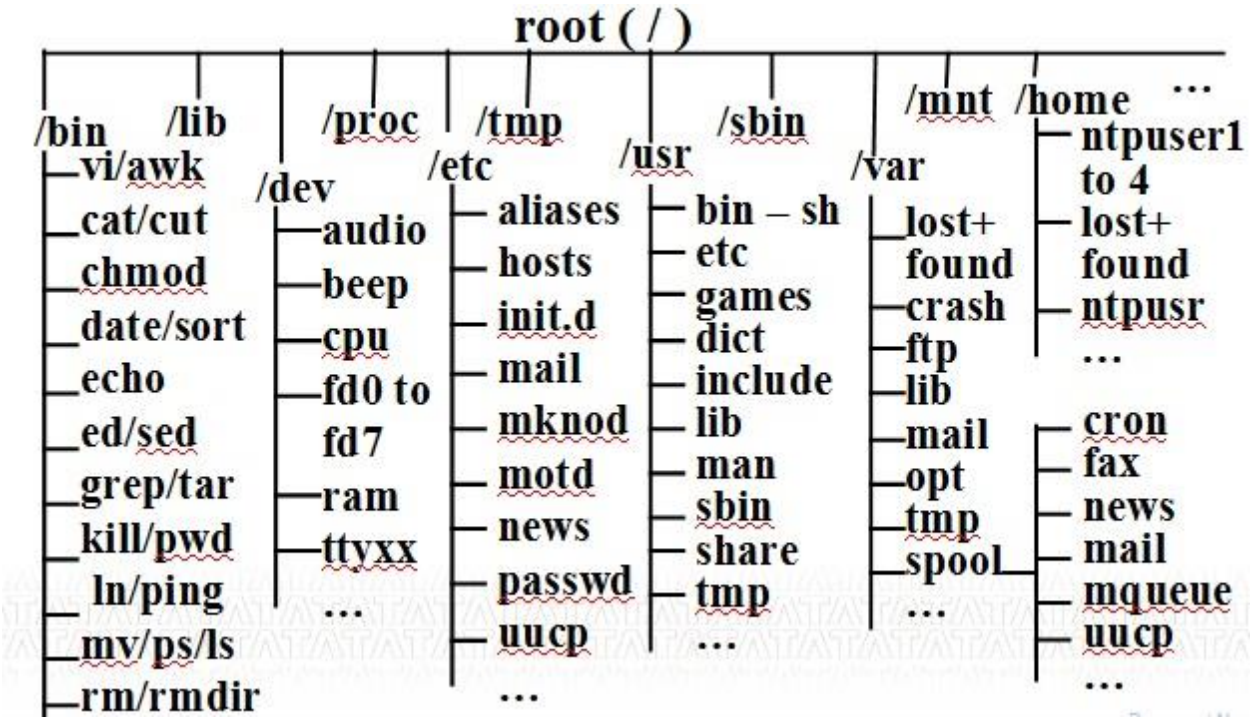
- UNIX file is featureless because it is simply an array of bytes.
- Dominant file type in UNIX is the text file.
- System related files are also stored in text form.
- Separate device can be added by creating a file for it.
- Root is the supremo and is represented by the '/'. Every sub-directory must have a parent.
- File names can be up to 14 characters long, can contain both upper and lower case alphabets, digits, a dot, hyphen (-), underscore (_) anywhere but should not have a blank or tab.
- Files are accessed using path names. Path names are a sequence of directory names separated by '/'.

There are two types of path names in unix.

- **Absolute path name -** file location is determined with respect to the root.
- **Relative path name -** file location is determined with respect to the current directory.

Structure of Unix File System:

The Unix File System (UFS) looks hierarchical but it is actually a directed a-cyclic graph because files can be shared.



Following are some directories available under root in UFS.

/home – It holds user's home directories. In other UNIX systems, this can be /usr directory.

/bin – It holds many of the basic Linux programs; bin stands for binaries, files that are executable.

/usr – It holds many user-oriented directories:

/sbin – It holds system files that are usually run automatically.

/etc – It and its subdirectories hold many of Linux config files.

/dev – It holds device files. All info sent to /dev/null is thrown into trash. Your terminal is one of the /dev/tty files.

Unix file System Organization:

- Unix divides physical disks into logical disks called ***partition*** which is comprising of a set of consecutive cylinders.
- The UFS resides on a single partition.
- Each file system contains four blocks :

a) **boot block** : This block is located in the first few sectors of a file system. It contains the initial bootstrap program used to load the operating system.Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.

b) **super block** : This block describes the state of the file system such as the total size of the partition, the block size, pointers to a list of free blocks, the inode number of the root directory, magic number or file signature (the first few bytes of a file which are unique to a particular file type), etc.

c) **inode list** : A linear array of *inodes* (index nodes). This is a data structure which describes the attributes of a file. There is a one to one mapping of files to inodes and vice versa. An inode is identified by it's "inode number". Users uses file names to refer to a file but Unix represent files in terms of inodes.

d) **data blocks** : blocks containing the actual contents of files

Internal File Maintenance:

For each file created in the system, an inode is also created. Inode is a disk file record of 64 bytes that maintains the permanent attributes of a file.

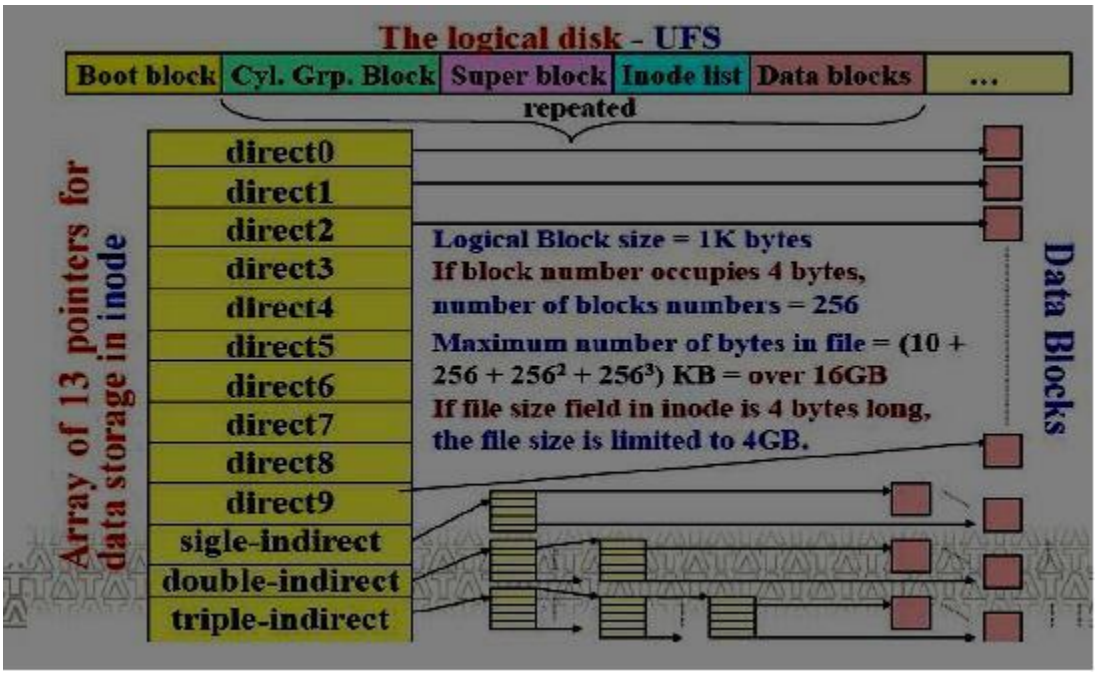
An inode is permanent and it exists until the corresponding file is removed from the system.

Sample details of an inode –

- Owner and group identifiers
- File type and file size

- Number of links for this file
- Times of file creation, last file access and modification, and last inode modification
- List of access rights – read/write/execute permissions
- Reference count showing number of times file is opened
- Physical address of file on the disk: array of 13 pointers for data storage

Whenever a file is opened, its inode is brought into main memory. The active inode is kept there until the file is closed and is used to locate the beginning of an open file on disk and to verify that every I/O request comes from a valid user as per specified access permissions.



Types of user accounts:
Root account, system account,user account

/etc file contains all user accounts and passwords

Owner level, group lvl, other lvl

File Permissions Types

r

- ✓ View the contents of the file
- ✓ View the contents of the directory using "ls" command

w

- ✓ Edit and save changes in the file
- ✓ Run or execute the file as a program/script

x

- ✓ Run or execute the file as a program/script
- ✓ Access the files in the directory

File Ownerships and Permissions

Following is an output from ls –l command.

```
-rw-rw-r-- 1 e308701 e308701   37 Mar 10 14:29 file2.txt
drwxrwxr-x 2 e308701 e308701 4096 Mar 10 14:37 iClass_23032016
-rw-rw-r-- 1 e308701 e308701   21 Mar 10 19:28 apple.txt
```

Nine columns in the Output

First column contains a string of ten characters

First character of the string in first column represents the file type and remaining characters represent the file permission details

File Ownerships and Permissions

```
drwxrwxr-x 2 e308701 e308701 4096 Mar 10 14:37 iClass_23032016
```

The following table specifies the significance of the values:

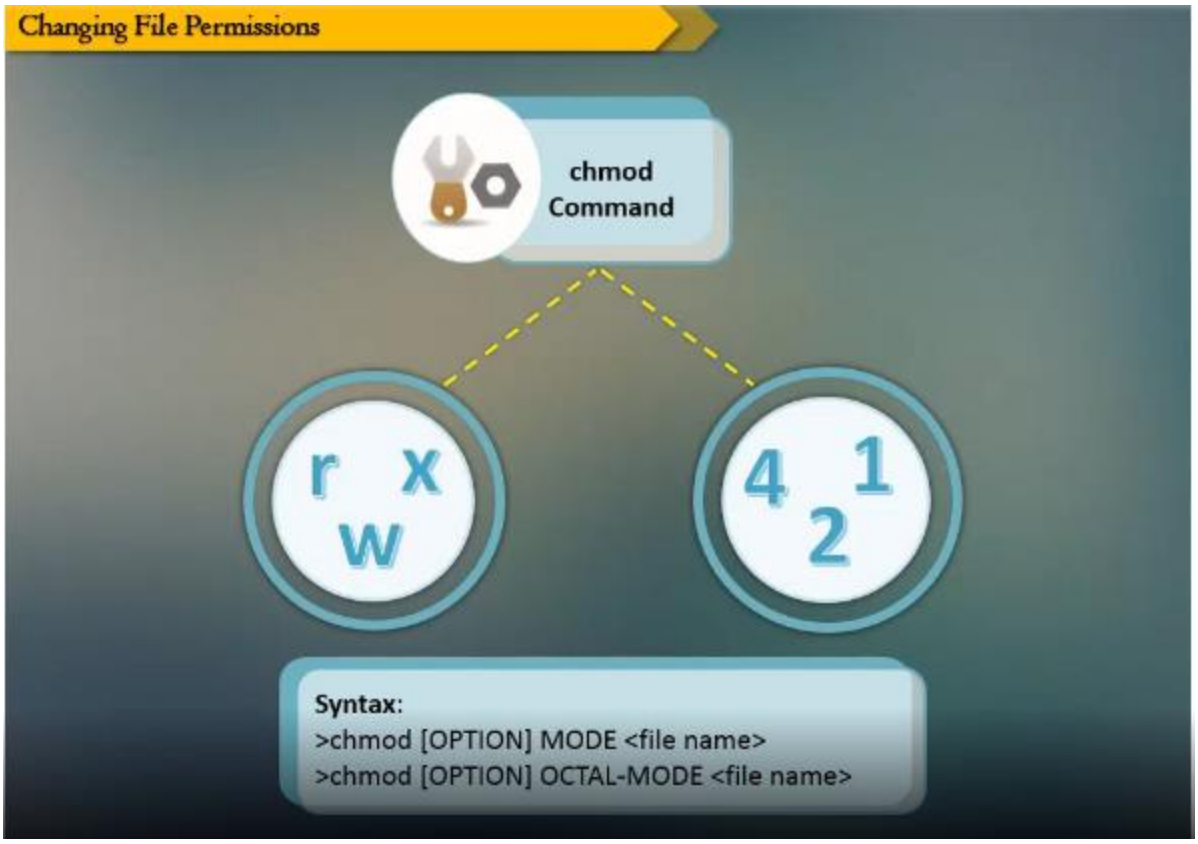
Character position	Value	Ownership Level	Significance
1	d	NA	Indicates whether file or directory. 'd' meaning directory; '-' meaning file
2	r	Owner	r- Read permission for file owner.
3	w	Owner	w - Write permission for file owner.
4	x	Owner	x- Execute permission for file owner.
5	r	Group	r- Read permission for the file owner's group.
6	w	Group	w- Write permission for the file owner's group.
7	x	Group	x- Execute permission for the file owner's group.
8	r	Others	r- Read permission for the other users excluding the file's owner and group.
9	w	Others	w- Write permission for the other users excluding the file's owner and group.
10	x	Others	x- Execute permission for the other users excluding the file's owner and group.

File Ownerships and Permissions

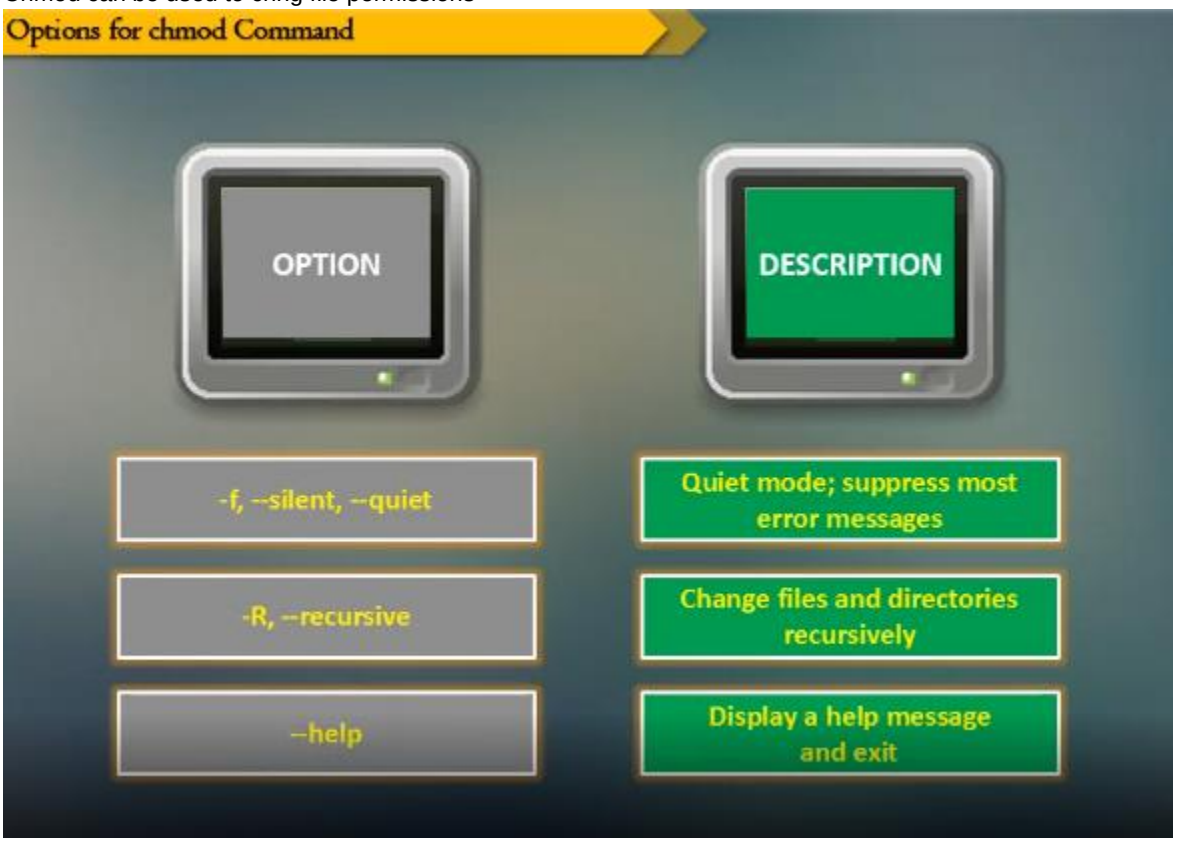
```
drwxrwxr-x 2 e308701 e308701 4096 Mar 10 14:37 iClass_23032016
```

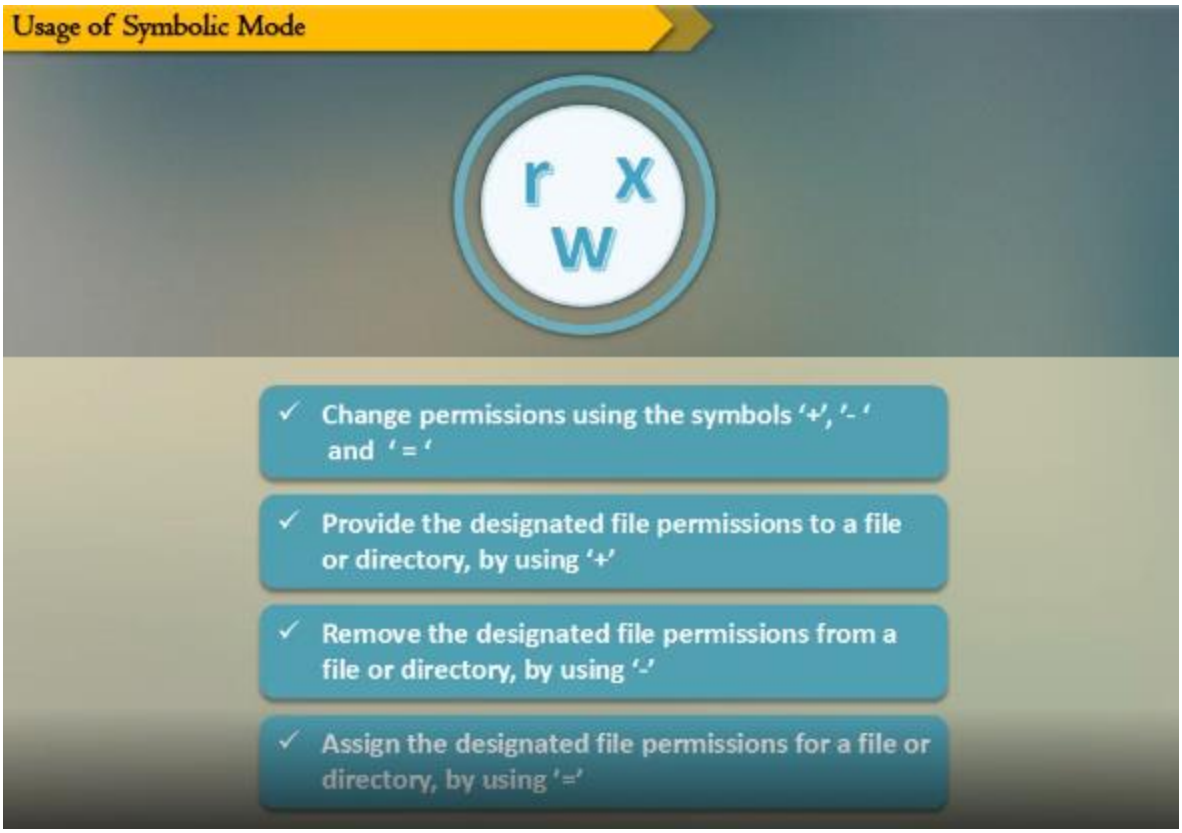

Character position	Value	Ownership Level	Significance
1	d	NA	Indicates whether file or directory. 'd' meaning directory; '-' meaning file
2	r	Owner	r- Read permission for file owner.
3	w	Owner	w - Write permission for file owner.
4	x	Owner	
5	r	Group	
6	w	Group	
7	x	Group	
8	r	Others	r- Read permission for the other users excluding the file's owner and group.
9	w	Others	w- Write permission for the other users excluding the file's owner and group.
10	x	Others	x- Execute permission for the other users excluding the file's owner and group.

If any of the values in character positions 2-10 is '-', it indicates that particular permission is denied.

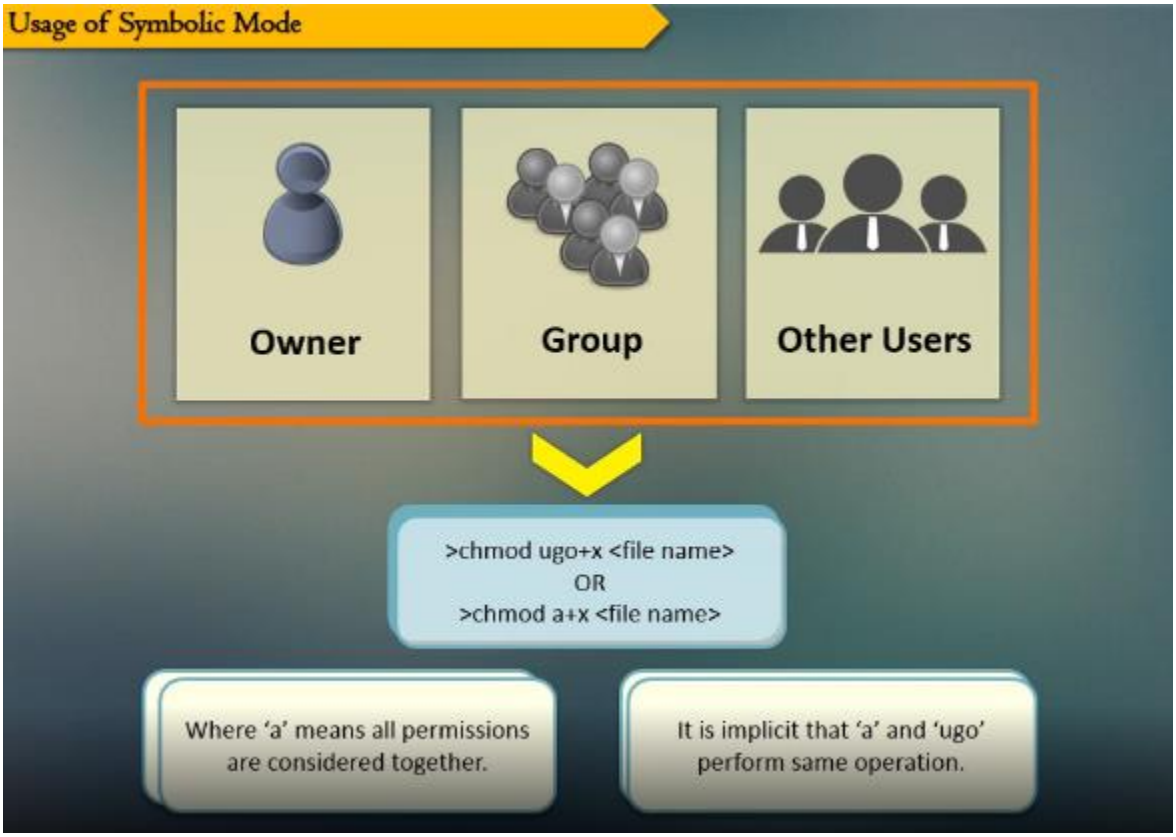


Chmod can be used to chng file permissions





If we want to provide execute permission to all levels of ownership



Usage of Symbolic Mode

chmod Command

>chmod 745<file name>

7 — Represents the read, write and execute permission of the owner.

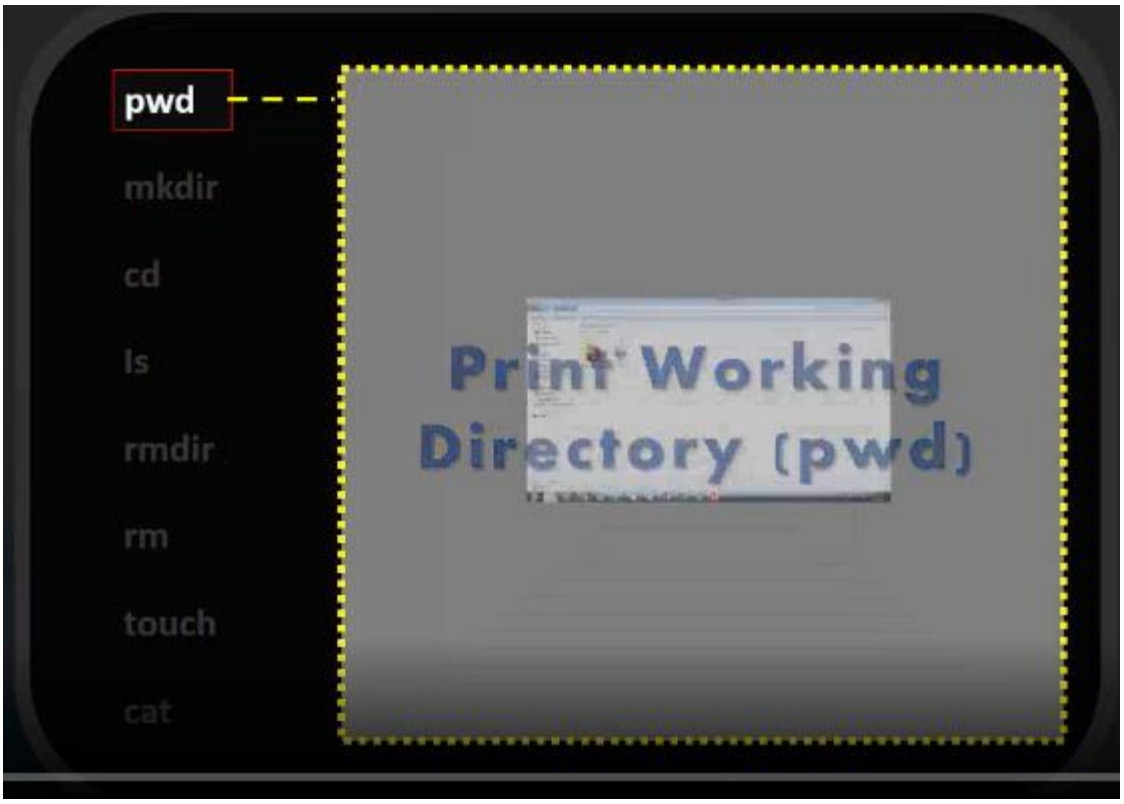
4 — Represents the read permission of the group.

5 — Represents the read and execute permission for all other accounts.

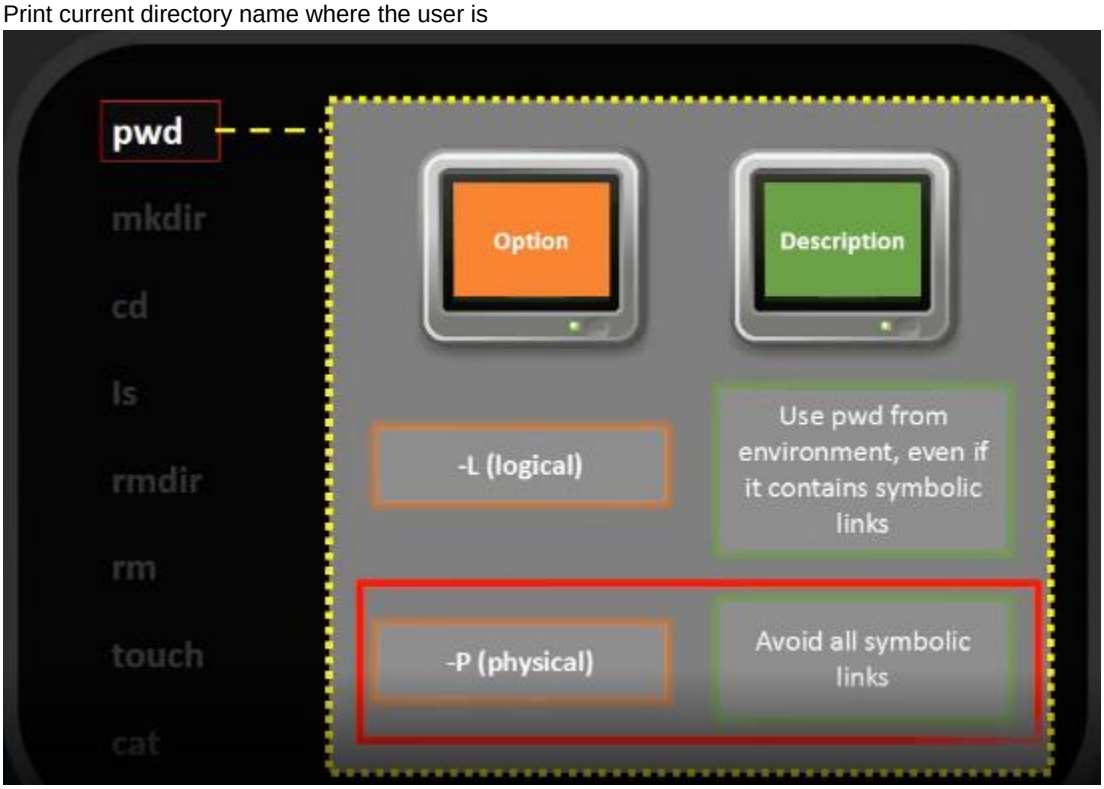
https://chmodcommand.com/chmod-745/

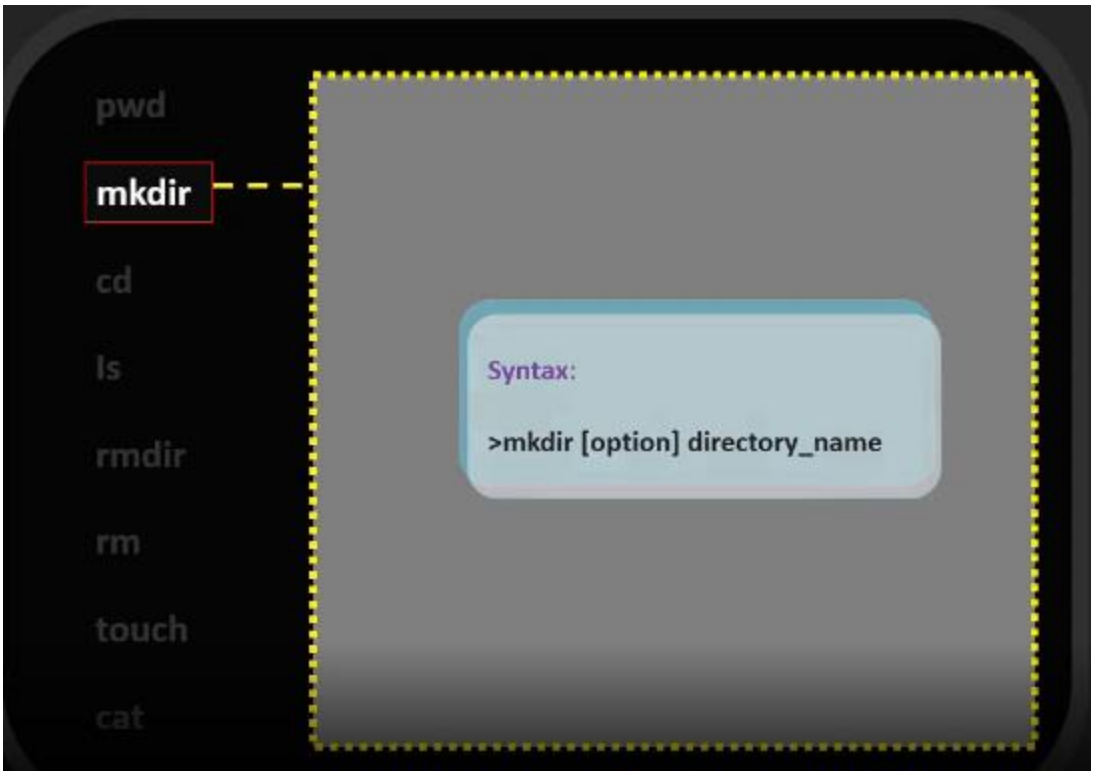
Usage of Absolute Mode (Octal Mode)

Octal Mode Representation	Binary Representation of the Octal Code			Symbolic Mode Representation	Description
	r	W	x		
0	0	0	0	---	None
1	0	0	1	--x	Execute only
2	0	1	0	-w-	Write only
3	0	1	1	-wx	Execute and write
4	1	0	0	r--	Read only
5	1	0	1	r-x	Read and execute
6	1	1	0	rw-	Read and write
7	1	1	1	rwx	Read, write, and execute

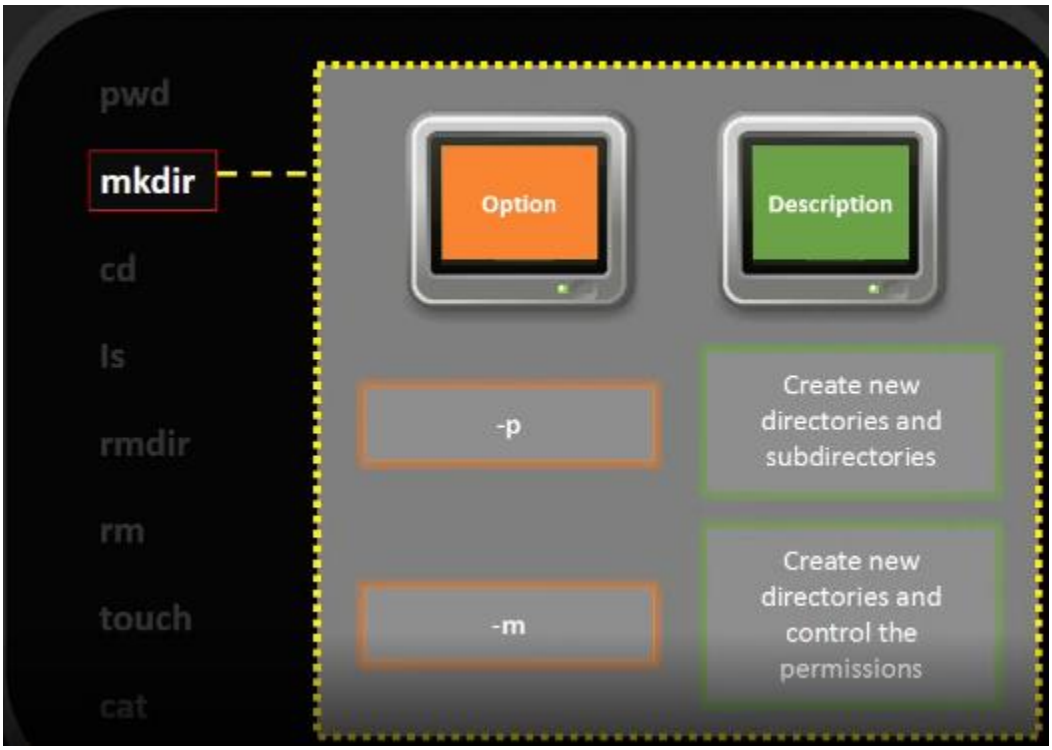


Print current directory name where the user is





Used to create directories, subdirectories



cd : change directory used to get to a desired directory
Cd ~ : go to home directory

```
e254840@inchnilp02:~/RoyalMailHotel
pts/0][12:59:34:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -a
```

Type the command **ls (space) (minus) (a)** to see the hidden files along with the files and directories in the directory RoyalMailHotel.

```
e254840@inchnilp02:~/RoyalMailHotel
[pts/0][12:59:34:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -a
.   BillDetails.txt  CustomerOrderDetails  FoodMenu
..  CustomerDetails  Employee              ReservedTableInfo
[pts/0][13:00:53:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -l
```

The command **ls (space)(minus)(l)** can be used to check the file properties like file creation date and time, created by, modified by, file size and file permission, etc.

```
e254840@inchnilp02:~/RoyalMailHotel
[pts/0][12:59:34:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -a
.   BillDetails.txt  CustomerOrderDetails  FoodMenu
..  CustomerDetails  Employee              ReservedTableInfo
[pts/0][13:00:53:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -l
total 12
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 BillDetails.txt
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 CustomerDetails
drwxrwxr-x 2 e254840 e254840 4096 Apr 26 18:55 CustomerOrderDetails
drwxrwxr-x 2 e254840 e254840 4096 May  5 12:59 Employee
drwxrwxr-x 2 e254840 e254840 4096 May  5 12:58 FoodMenu
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 ReservedTableInfo
[pts/0][13:01:06:e254840@inchnilp02 ] ~/RoyalMailHotel>
```

Objective: To understand the usage of the rest of the basic UNIX commands in Unix Operating System

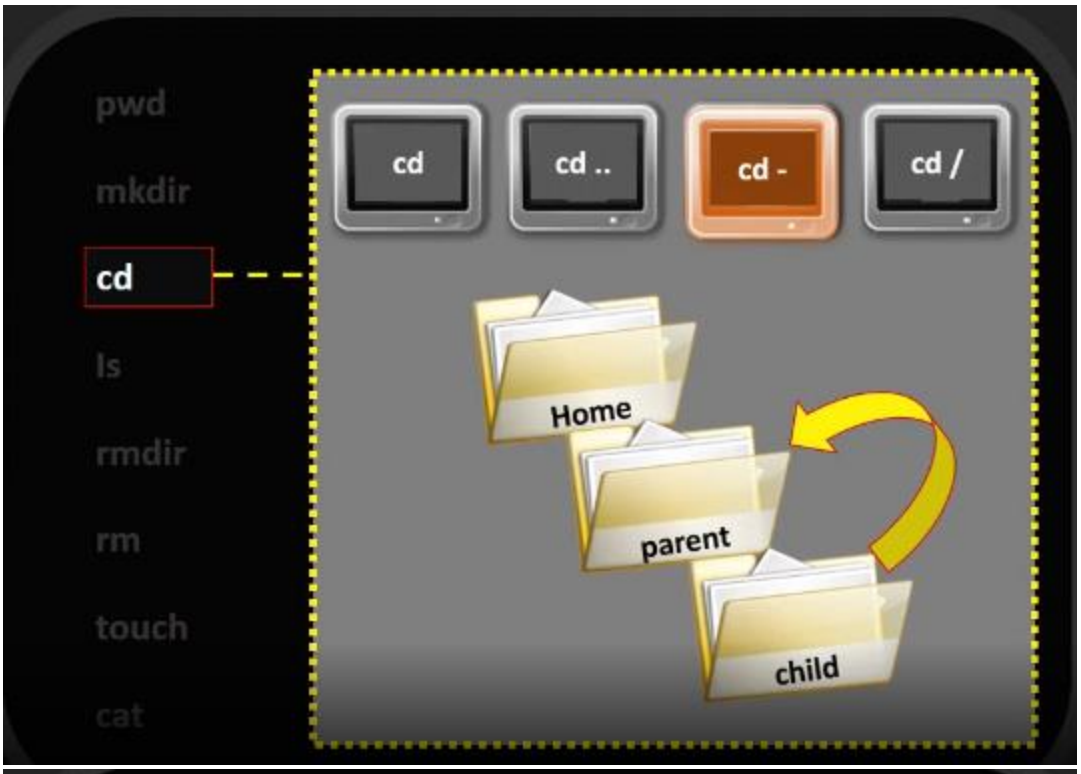
Royal Mail Hotel

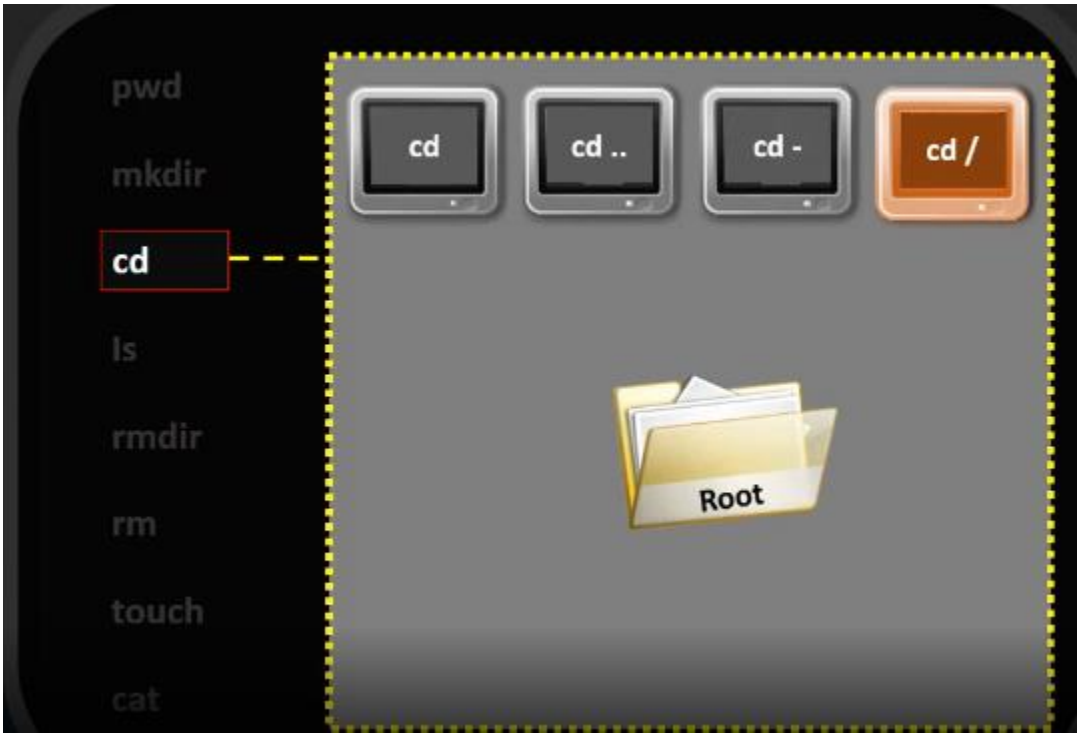
Second Criteria:

- Manager would like to know the files which are starting with the word "Customer" and the files having names with the word "Table".

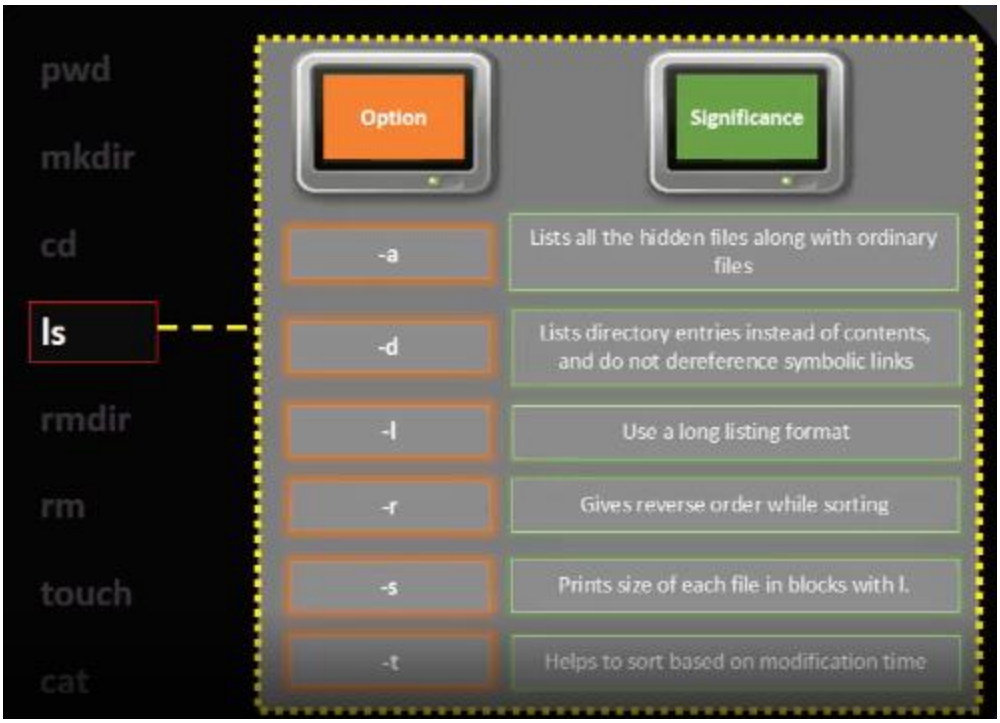
```
[pts/0][13:01:06:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -l Customer*
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 CustomerDetails

CustomerOrderDetails:
total 0
```

```
[pts/0][13:01:43:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -l *Table*
-rw-rw-r-- 1 e254840 e254840 0 Apr 26 18:59 ReservedTableInfo
```

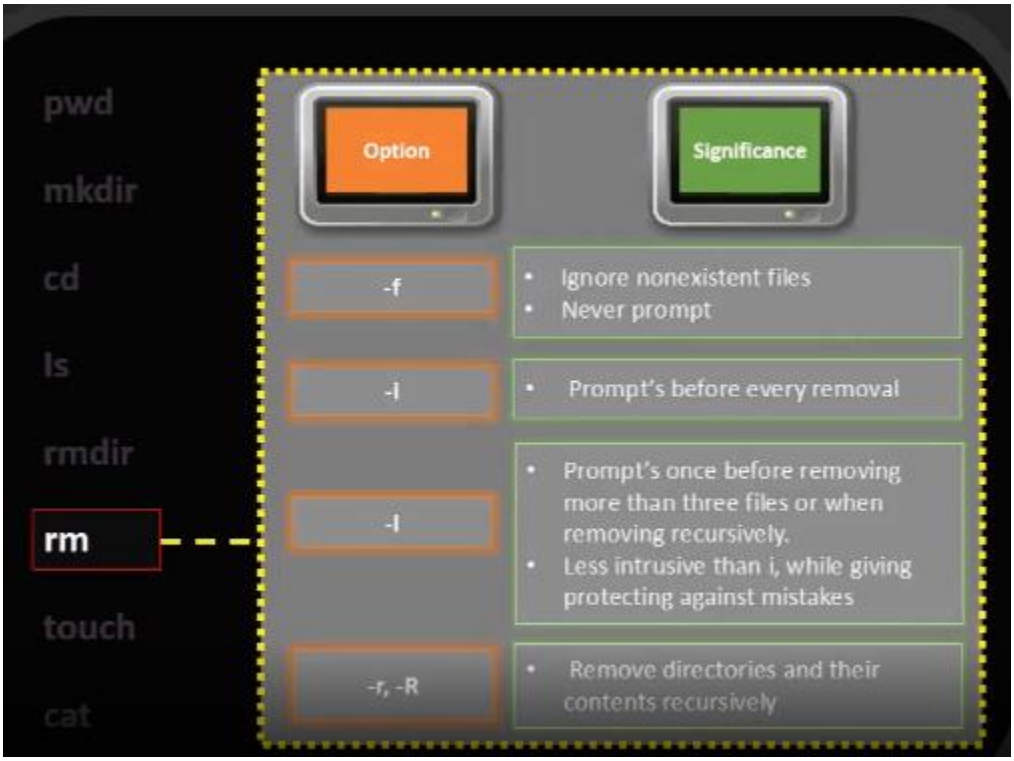




Ls: lists the unhidden files in the current directory

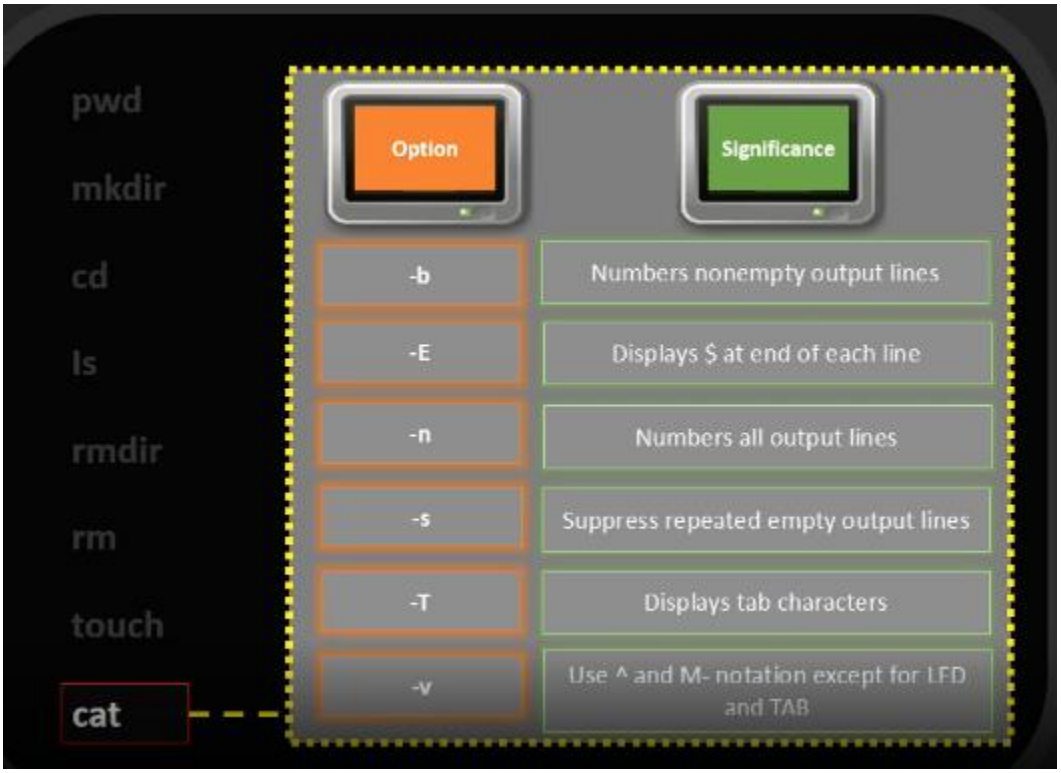


Rmdir:remove directory
Rm:remove



Touch :create an empty file or changes modification time to current time

Cat: used to put content in the file or used to concat and print as output: cat>filename ,,then text content
Cat <space> filename prints the contents of the file
cat>>filename: used to append data to the existing content instead of over writing



Man command **********

```
e254840@inchnilp02:~/RoyalMailHotel
[pts/0][13:02:19:e254840@inchnilp02 ] ~/RoyalMailHotel> man touch
```

- Use the command **man (space) touch.**

man command is used to display the manual page for any UNIX command. The manual page contains the description of the command and options used for that command.

```
e254840@inchnilp02:~/RoyalMailHotel/FoodMenu
[pts/0][13:03:26:e254840@inchnilp02 ] ~/RoyalMailHotel> ls -l
total 12
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 BillDetails.txt
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 CustomerDetails
drwxrwxr-x 2 e254840 e254840 4096 Apr 26 18:55 CustomerOrderDetails
drwxrwxr-x 2 e254840 e254840 4096 May  5 12:59 Employee
drwxrwxr-x 2 e254840 e254840 4096 May  5 12:58 FoodMenu
-rw-rw-r-- 1 e254840 e254840    0 Apr 26 18:59 ReservedTableInfo
[pts/0][13:03:40:e254840@inchnilp02 ] ~/RoyalMailHotel> cd FoodMenu
[pts/0][13:03:56:e254840@inchnilp02 ] ~/RoyalMailHotel/FoodMenu> ls -l
total 4
-rwxrwxrwx 1 e254840 e254840 120 Apr 18 12:51 FoodItemDetails
[pts/0][13:04:04:e254840@inchnilp02 ] ~/RoyalMailHotel/FoodMenu> cp FoodItemDetails FoodIt
emDetails_bkup
```

To copy the content from an existing file FoodItemDetails to the new file named FoodItemDetails_bkup use command **cp (space) FoodItemDetails(space) FoodItemDetails_bkup**

```
[pts/0][13:04:04:e254840@inchnilp02 ] ~/RoyalMailHotel/FoodMenu> cp FoodItemDetails FoodIt
emDetails_bkup
[pts/0][13:04:28:e254840@inchnilp02 ] ~/RoyalMailHotel/FoodMenu> ls -l
total 8
-rwxrwxrwx 1 e254840 e254840 120 Apr 18 12:51 FoodItemDetails
-rwxrwxr-x 1 e254840 e254840 120 May  5 13:04 FoodItemDetails_bkup
[pts/0][13:04:58:e254840@inchnilp02 ] ~/RoyalMailHotel/FoodMenu>
```

Use the command **mv (space) RoyalMailHotel/Employee/EmployeeDetails_bkup (space) RoyalMailHotel_bkup**
is used to move the file from either one directory to another directory or within the same directory.

To change the name of a file, use the following command format (where **thirdfile** and **file3** are sample file names):

mv thirdfile file3

What will be the result of the command "echo welcome > /dev / tty"?

A. Echoes welcome only in the terminal in which it is runned

B. Echoes welcome in all the termilas that are logged on.

C. Echoes welcomes in all the termilnals that are switched on.

D. None

The correct answer to this question is A and C.

There is **a space on both the sides of the operator.**

The command should be enclosed **within inverted commas.**

There should not be any space while assigning value to a variable.

```
[pts/0][18:33:46:e244121@inchnilp02 ] ~/trdf> cat arithm.sh
#!/bin/sh
x=15
y=30
#Example using + Operator
adt=`expr $x + $y`
echo "x + y : $adt"

#Example using - Operator
sub=`expr $x - $y`
echo "x - y : $sub"

#Example using * Operator
mul=`expr $x \* $y`
echo "x * y : $mul"

#Example using / Operator
div=`expr $y / $x`
echo "y / x : $div"

#Example using % Operator
mod=`expr $y % $x`
echo "y % x : $mod"

#Example using == Operator
y=15
if [ $x == $y ]
then
echo "x is equal to y"
fi

#Example using != Operator
y=35
if [ $x != $y ]
then
echo "x is not equal to y"
```

Shell Program

```
[pts/0][18:35:49:e244121@inchnilp02 ] ~/trdf> sh arithm.sh
x + y : 45
x - y : -15
x * y : 450
y / x : 2
y % x : 0
x is equal to y
x is not equal to y
```

Output

Relational operators are used to perform comparisons among the given operands.

Operators	Description
-eq	This operator is used to check whether given two operands are equal and return TRUE if both are equal.
-ne	This operator is used to check whether given two operands are not equal and it returns TRUE if they are not equal.
-gt	This operator is used to check whether the left hand side operand is greater than right hand side operand, and it returns TRUE if the left hand side operand is greater than the right hand side operand.
-lt	This operator is used to check whether the left hand side operand is lesser than right hand side operand, and it returns TRUE if the left hand side operand is lesser than the right hand side operand.
-ge	This operator is used to check whether the left hand side operand is greater than or equal to right hand side operand, and it returns TRUE if the left hand side operand is greater than or equal to the right hand side operand.
-le	This operator is used to check whether the left hand side operand is lesser than or equal to right hand side operand, and it returns TRUE if the left hand side operand is lesser than or equal to the right hand side operand.

Shell Program

```
#!/bin/sh
x=10
y=20
#Example using -eq Operator
if [ $x -eq $y ]
then
echo "$x -eq $y : x is equal to y"
else
echo "$x -eq $y: x is not equal to y"
fi

#Example using -ne Operator
if [ $x -ne $y ]
then
echo "$x -ne $y: x is not equal to y"
else
echo "$x -ne $y : x is equal to y"
fi

#Example using -gt Operator
if [ $x -gt $y ]
then
echo "$x -gt $y: x is greater than y"
else
echo "$x -gt $y: x is not greater than y"
fi

#Example using -lt Operator
if [ $x -lt $y ]
then
echo "$x -lt $y: x is less than y"
else
echo "$x -lt $y: x is not less than y"
fi

#Example using -ge Operator
if [ $x -ge $y ]
then
echo "$x -ge $y: x is greater or equal to y"
else
echo "$x -ge $y: x is not greater or equal to y"
fi

#Example using -le Operator
if [ $x -le $y ]
then
echo "$x -le $y: x is less or equal to y"
else
```

Output

```
[pts/0][18:35:58:e2441210@inchnilp02 ] ~/trdf> sh relation.sh
10 -eq 20: x is not equal to y
10 -ne 20: x is not equal to y
10 -gt 20: x is not greater than y
10 -lt 20: x is less than y
10 -ge 20: x is not greater or equal to y
10 -le 20: x is less or equal to y
```

Boolean Operators

Operators	Description
!	This operator is equivalent to the not logic gate used in electronics, as it converts a true condition to false and vice-versa.
-o	This operator returns TRUE if either of the given conditions is true.
-a	This operator returns TRUE if both the conditions are true.

Shell Program

```
[pts/0][18:51:48:e244121@inchnilp02 ] ~/trdf> cat bool.sh
#!/bin/sh

x=20
y=30

#Example using != Operator
if [ $x != $y ]
then
echo "$x != $y : x is not equal to y"
else
echo "$x != $y: x is equal to y"
fi

#Example using -a Operator
if [ $x -lt 100 -a $y -gt 15 ]
then
echo "$x -lt 100 -a $y -gt 15 : returns true"
else
echo "$x -lt 100 -a $y -gt 15 : returns false"
fi

#Example using -o Operator
if [ $x -lt 100 -o $y -gt 100 ]
then
echo "$x -lt 100 -o $y -gt 100 : returns true"
else
echo "$x -lt 100 -o $y -gt 100 : returns false"
fi

#Example using -o Operator
if [ $x -lt 5 -o $y -gt 100 ]
then
echo "$x -lt 5 -o $y -gt 100 : returns true"
else
echo "$x -lt 5 -o $y -gt 100 : returns false"
fi
```

Output

```
[pts/0][18:52:39:e244121@inchnilp02 ] ~/trdf> sh bool.sh
20 != 30 : x is not equal to y
20 -lt 100 -a 30 -gt 15 : returns true
20 -lt 100 -o 30 -gt 100 : returns true
20 -lt 5 -o 30 -gt 100 : returns false
```

String operators are used to compare two strings.

Operators	Description
=	Return the value as TRUE, if both the string operands are equal
!=	Return the value as TRUE, if the given string operands are not equal
z	Return the value as TRUE, if the size of the given string operand is zero
n	Return the value as TRUE, if the size of the given string operand is non-zero
str	Return the value as TRUE, if the given string operand is not empty

String Operators

```
[pts/0][18:57:51:e244121@inchnilp02 ] ~/trdf> cat stringfile.sh
#!/bin/sh
x="abc"
y="xyz"
#Example using = Operator
if [ $x = $y ]
then
echo "$x = $y : x is equal to y"
else
echo "$x = $y: x is not equal to y"
fi

#Example using != Operator
if [ $x != $y ]
then
echo "$x != $y: x is not equal to y"
else
echo "$x != $y : x is equal to y"
fi

#Example using -z Operator
a=""
if [ -z $a ]
then
echo "-z $a : string length is zero "
else
echo "-z $a : string length is not zero "
fi

#Example using -n Operator
if [ -n $y ]
then
echo "-n $y: string length is not zero "
else
echo "-n $y: string length is zero "
fi

#Example using str Operator
if [ $x ]
then
echo "$x : string is not empty"
else
echo "$x : string is empty"
fi
```

Shell Program

```
[pts/0][18:52:44:e244121@inchnilp02 ] ~/trdf> sh stringfile.sh
abc = xyz: x is not equal to y
abc != xyz: x is not equal to y
-z  : string length is zero
-n xyz: string length is not zero
abc : string is not empty
```

Output

File Test Operators

File Test operators are used to test the various properties of UNIX files.

Operators	Description
b	Returns TRUE, if the given file is block special file
c	Returns TRUE, if the given file is character special file
d	Returns TRUE, if the given file is a directory
f	Returns TRUE, if the given file is an ordinary file and not special file or directory file
g	Returns TRUE, if the given file is set with its group ID
k	Returns TRUE, if the given file has its sticky bit
p	Returns TRUE, if the given file is a named pipe

File Test Operators

File Test operators are used to test the various properties of UNIX files.

Operators	Description
t	Returns TRUE, if the given file descriptor is open
u	Returns TRUE, if the given file is set with its user ID
r	Returns TRUE, if the given file is a readable file
w	Returns TRUE, if the given file is a writeable file
x	Returns TRUE, if the given file is an executable file
s	Returns TRUE, if the given file size is greater then zero
e	Returns TRUE, if the file exists

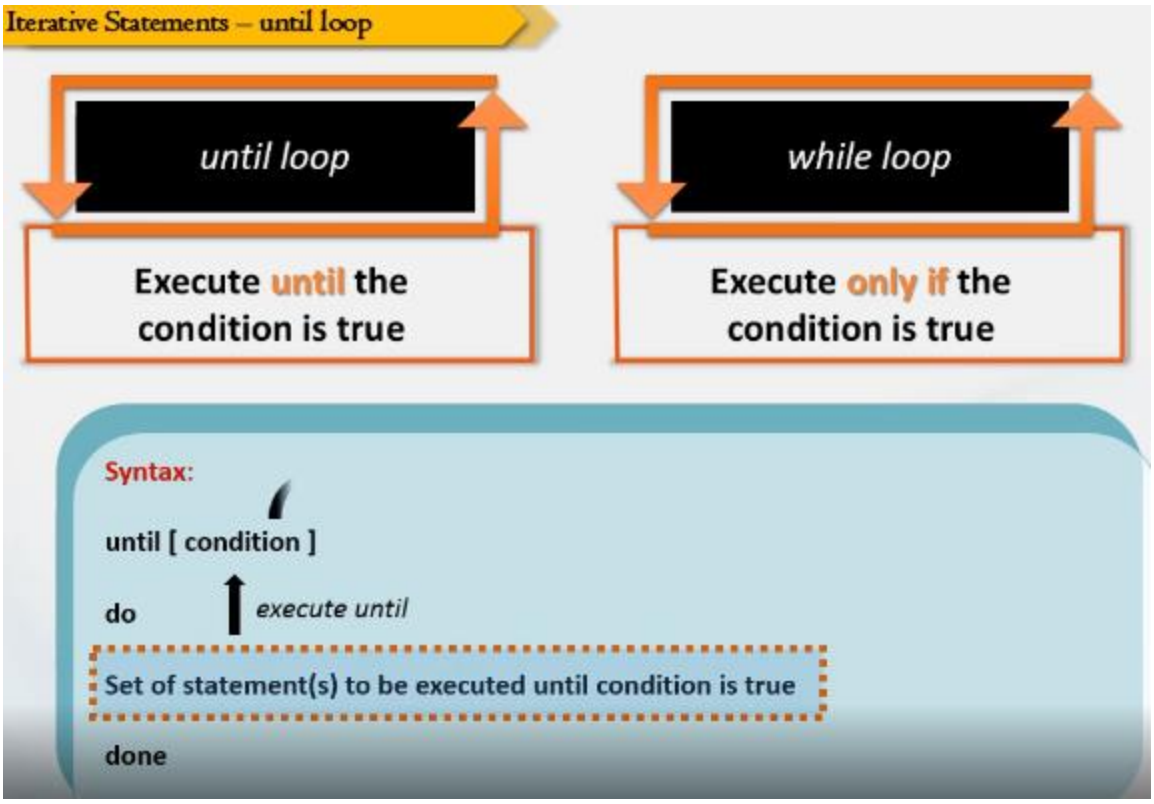
```
[pts/0][15:28:00:e244121@inchnilp02 ] ~/trdf> cat for_list.sh
#!/bin/sh

echo "For Loop example using lists of items"
for i in 0 1 2 3 4
do
   echo $i
done
```

```
[pts/0][15:31:19:e244121@inchnilp02 ] ~/trdf> sh for_list.sh
For Loop example using lists of items
0
1
2
3
4
```

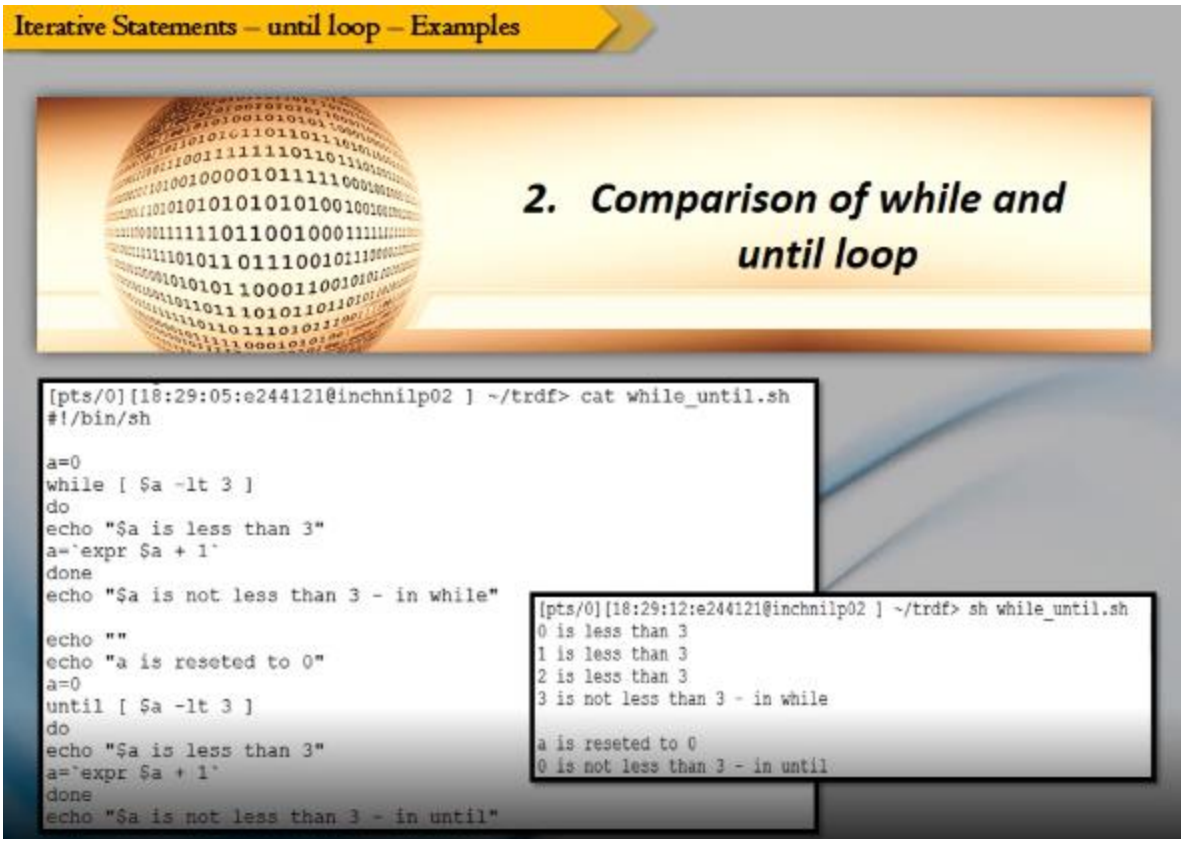
```
[pts/0][16:42:57:e244121@inchnilp02 ] ~/trdf> cat while_condi.sh
#!/bin/sh

a=10
b=5
while [ $a -gt $b ]
do
echo "$a is greater than $b"
a=`expr $a - 1`
done
echo "$a is not greater than $b"
```

```
[pts/0][16:42:59:e244121@inchnilp02 ] ~/trdf> sh while_condi.sh
10 is greater than 5
9 is greater than 5
8 is greater than 5
7 is greater than 5
6 is greater than 5
5 is not greater than 5
```

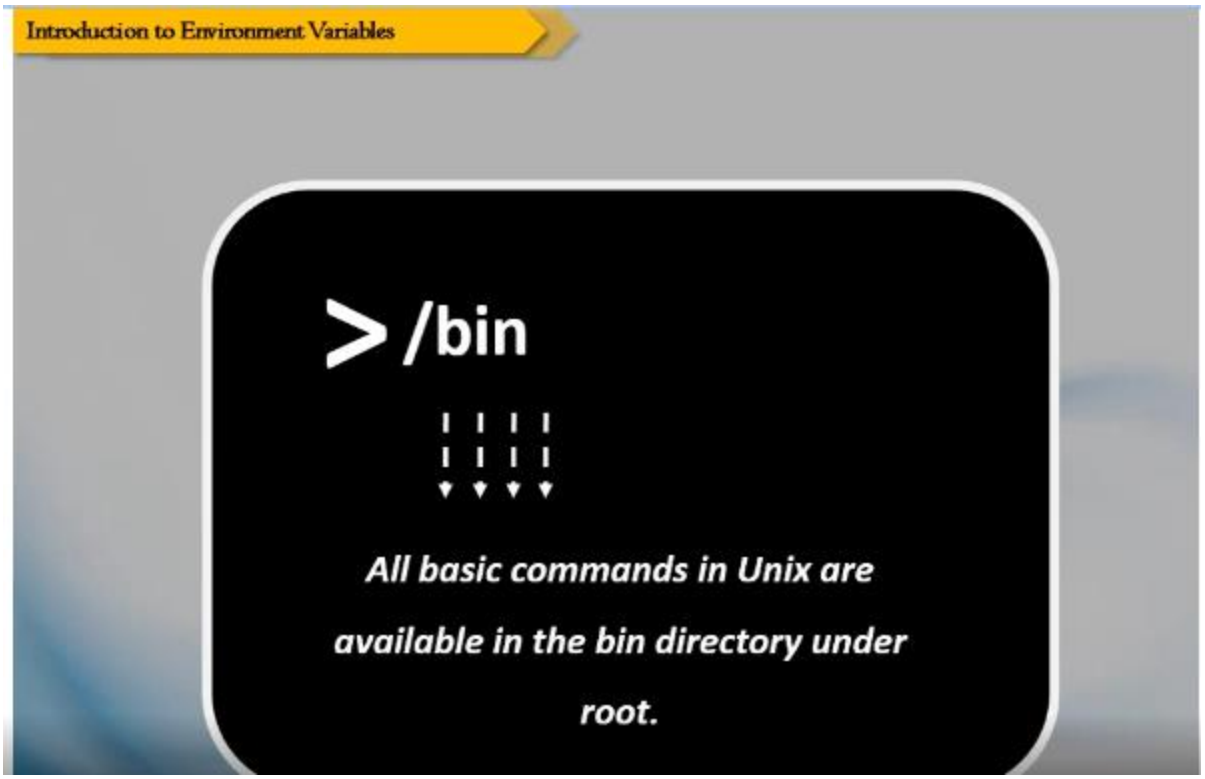
```
[pts/0][16:36:34:e244121@inchnilp02 ] ~/trdf> cat fruits.txt
Apple
Gova
Kiwi
Orange
Mango
[pts/0][16:36:44:e244121@inchnilp02 ] ~/trdf> cat while_file.sh
#!/bin/sh

while read line
do
echo $line
done < fruits.txt
```



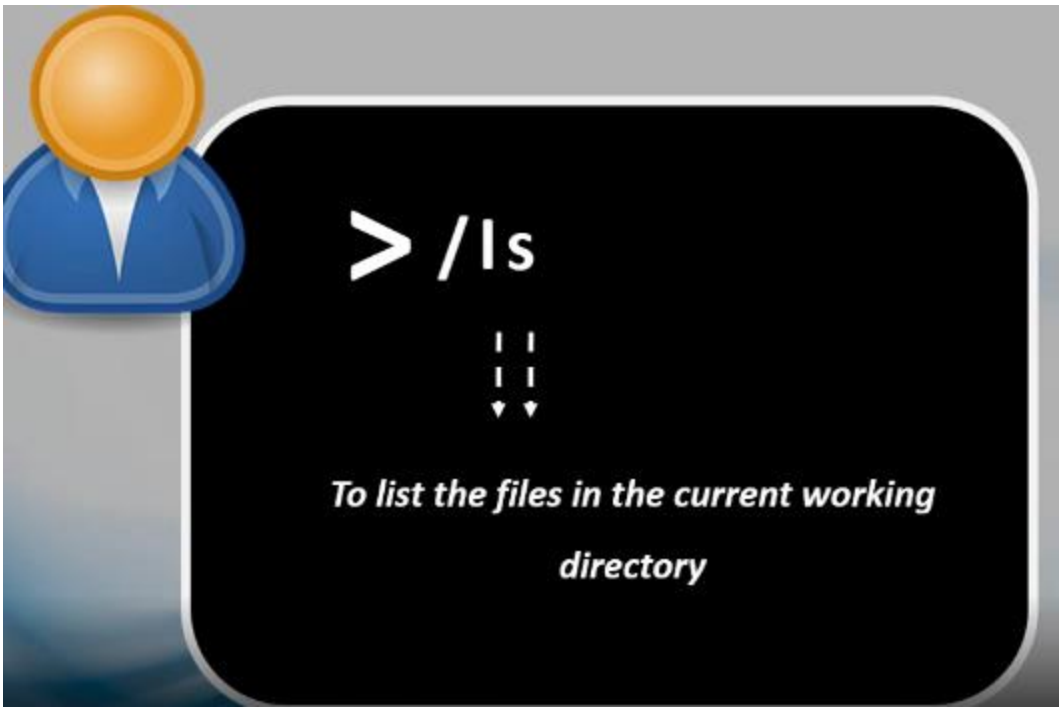

```
[pts/0][18:20:05:e244121@inchnilp02 ] ~/trdf> cat until_condi.sh
#!/bin/sh

a=10
b=5
until [ !$a -gt $b ]
do
echo "$a is greater than $b"
a=`expr $a - 1`
done
echo "$a is not greater than $b"
```



1. Set execute permission on your script:
   `chmod +x script-name-here.sh`
2. To run your script, enter:
   `./script-name-here.sh`
   OR
   `sh script-name-here.sh`


Introduction to Environment Variables
> /bin
All basic commands in Unix are available in the bin directory under root.

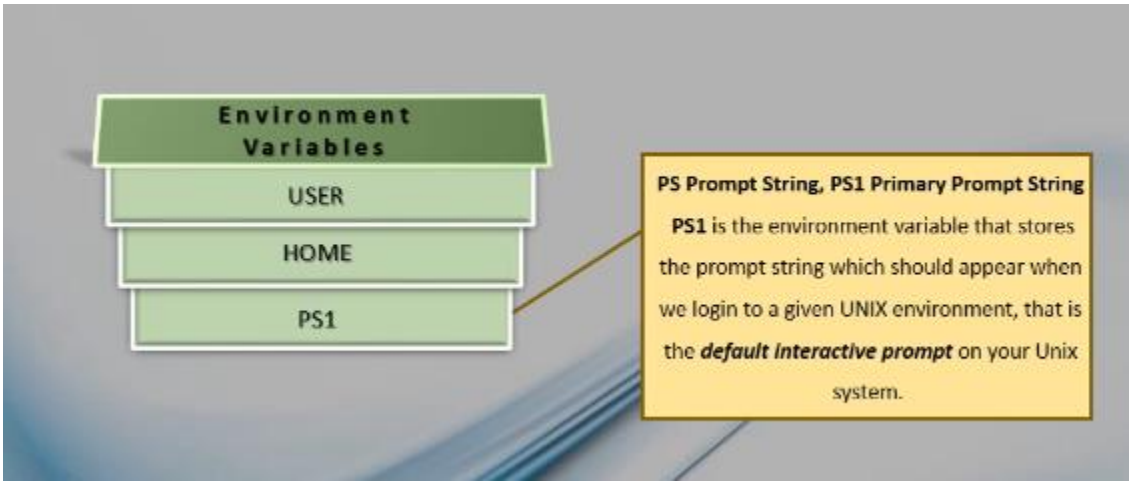

> /bin/ls
To list the current working directory

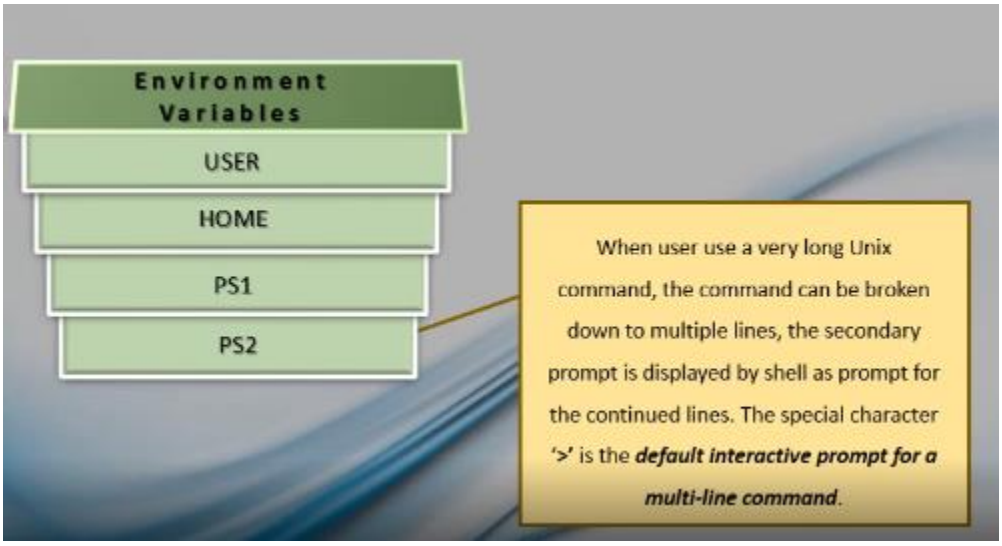


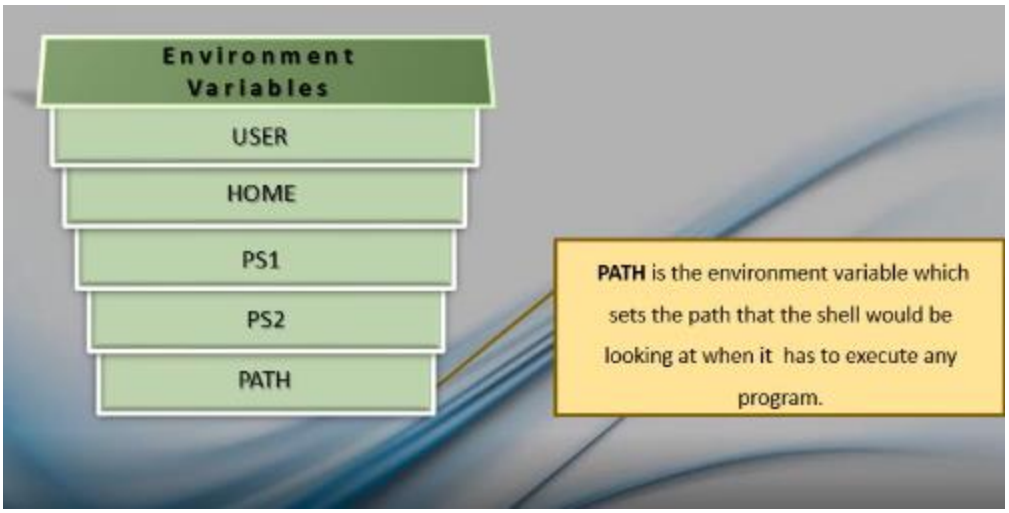
However users can type in simply ls due to the environment variable called path

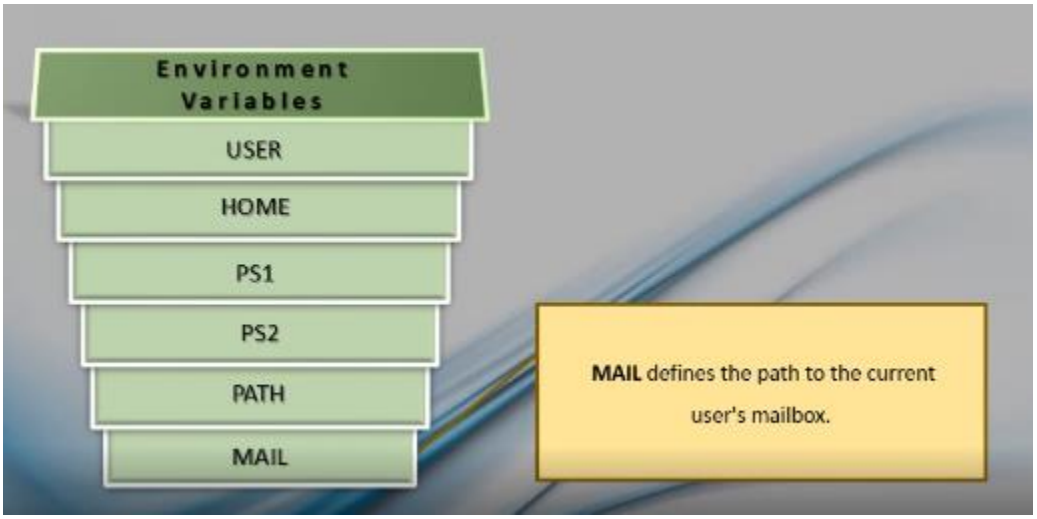
Few examples of environment variables:

Environment Variables	Description
OS TYPE	Operating system name
USER	User login name for that particular console
HOME	Path of the home directory, when user login to the Unix environment
PATH	The directories that the shell search during the execution of *find* command
PS1	Primary prompt string
PS2	Secondary prompt string
HISTSIZE	Number of commands to be displayed, when history command is executed without options
PWD	Present working directory






Environment Variables
USER
HOME
PS1
PS2
PATH
PATH is the environment variable which sets the path that the shell would be looking at when it has to execute any program.


Environment Variables
USER
HOME
PS1
PS2
PATH
MAIL
MAIL defines the path to the current user's mailbox.

Setting Environment Variables

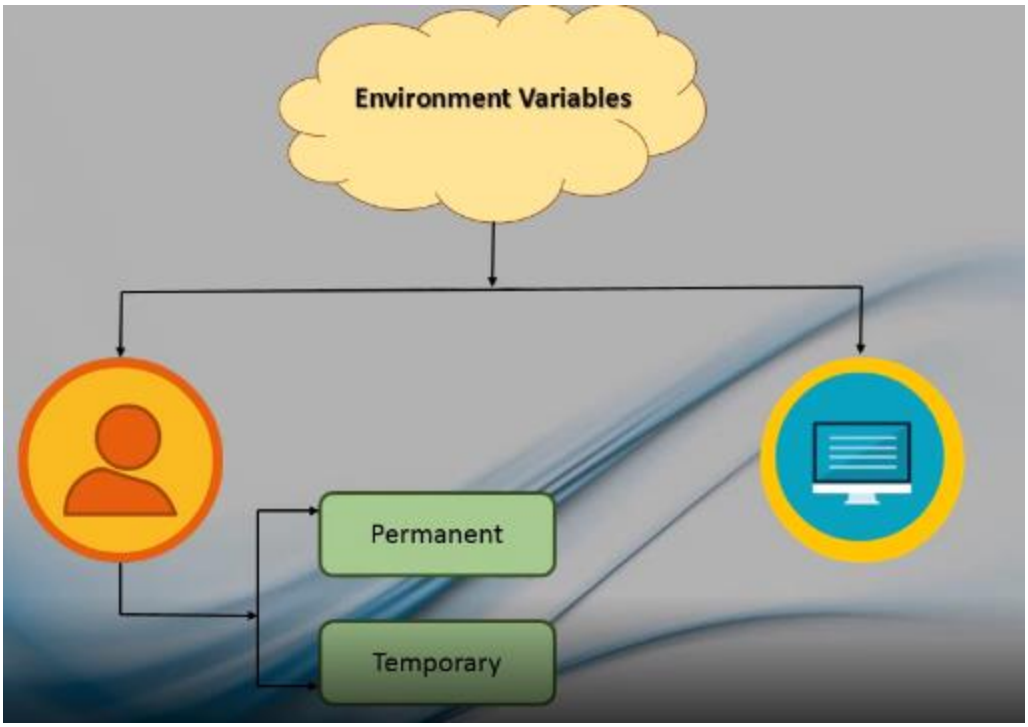
Variable Name = Value (Any string as a value)

```
> HOME = /home/CPP
```

It will set the home directory to the CPP directory.

```
> PATH=/usr:/bin/:usr/local/bin:
```

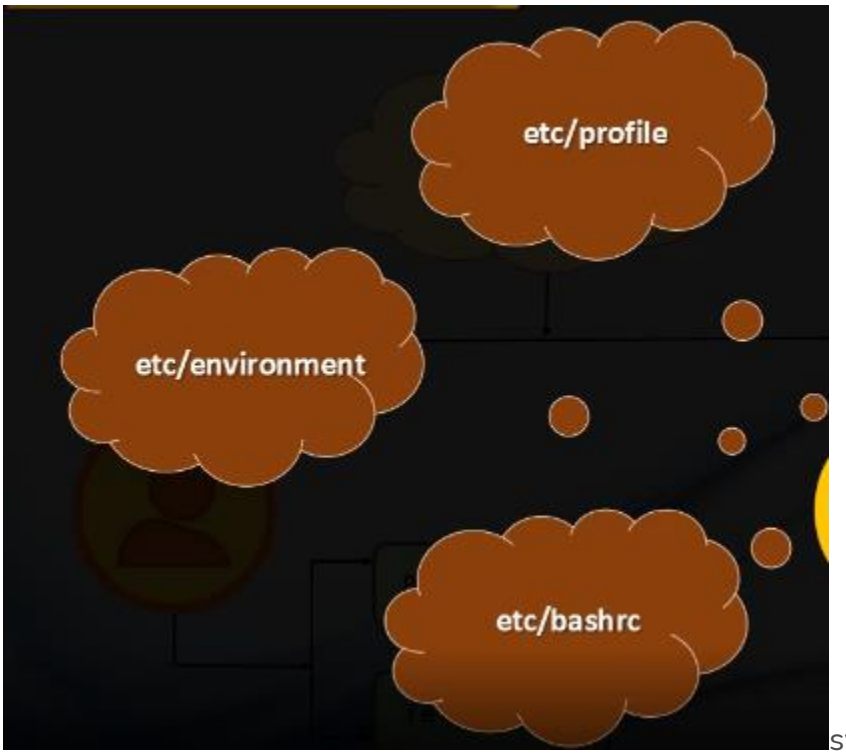
Multiple values are separated by a ':'.



1.user wide change and 2.system wide change

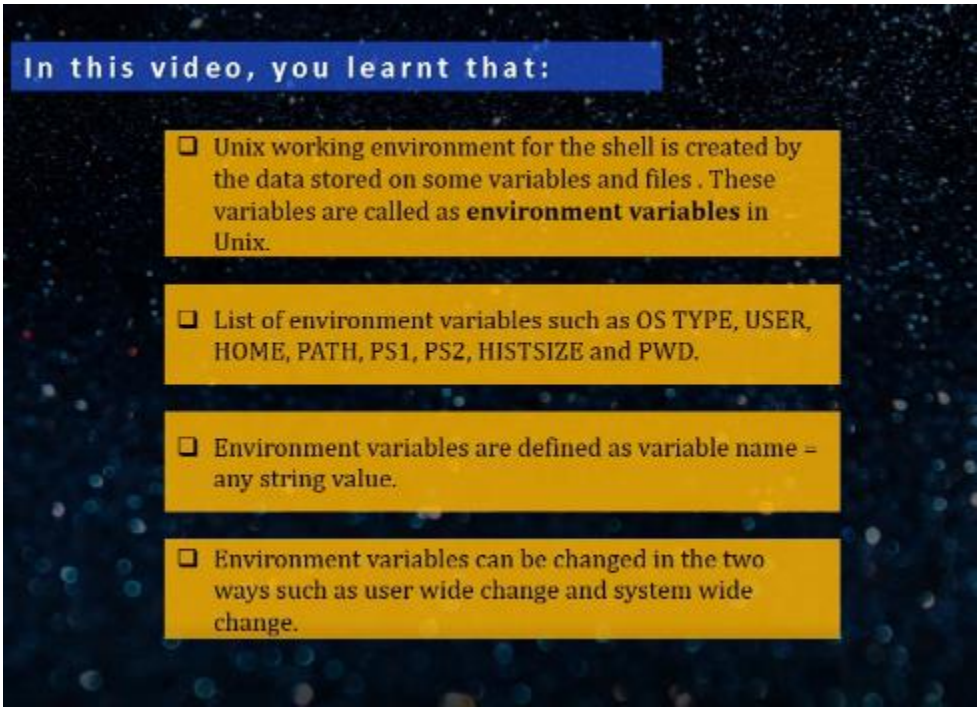
Permanent: to make the changes in the environment variable permanent

In the temporary user wide change, the changes made will reflect only in that particular session, where the value of variable was changed. The changes are made in the command prompt of the UNIX console, by assigning the new values to the environment variables.

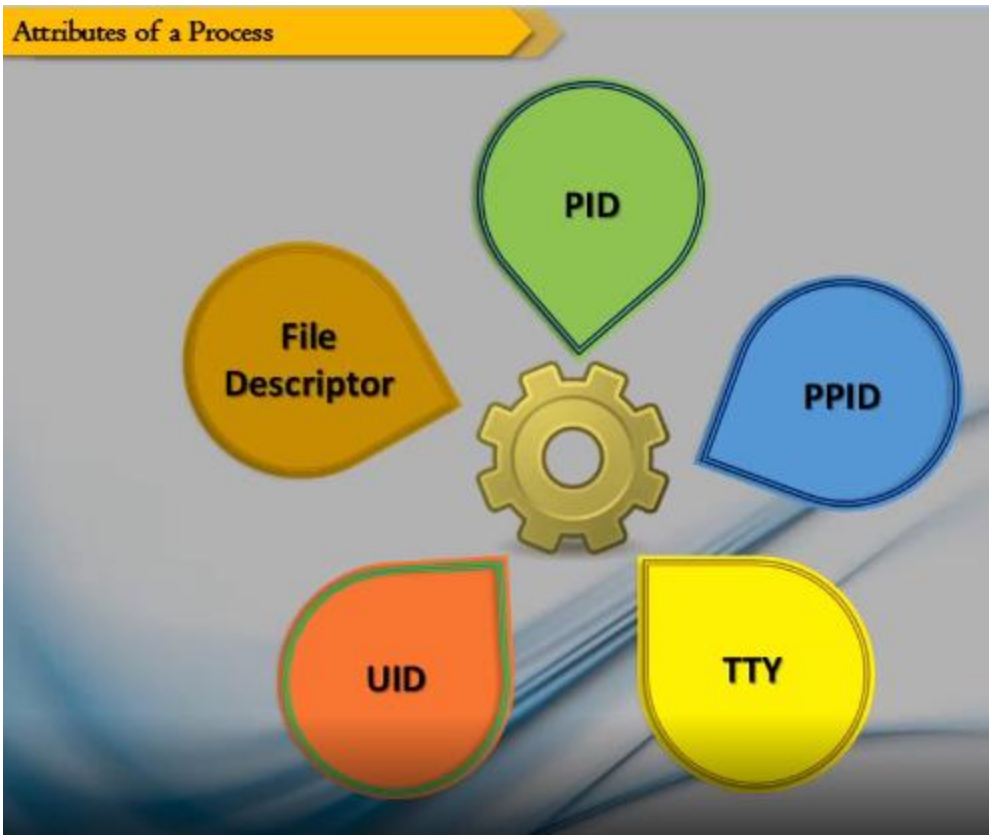


system wide environment

variables are present in these files.



Process is an instance of a program which runs in the background.
When a program is called a process is created and assigned a process id.
getpid() is used t get the id assigned to a process



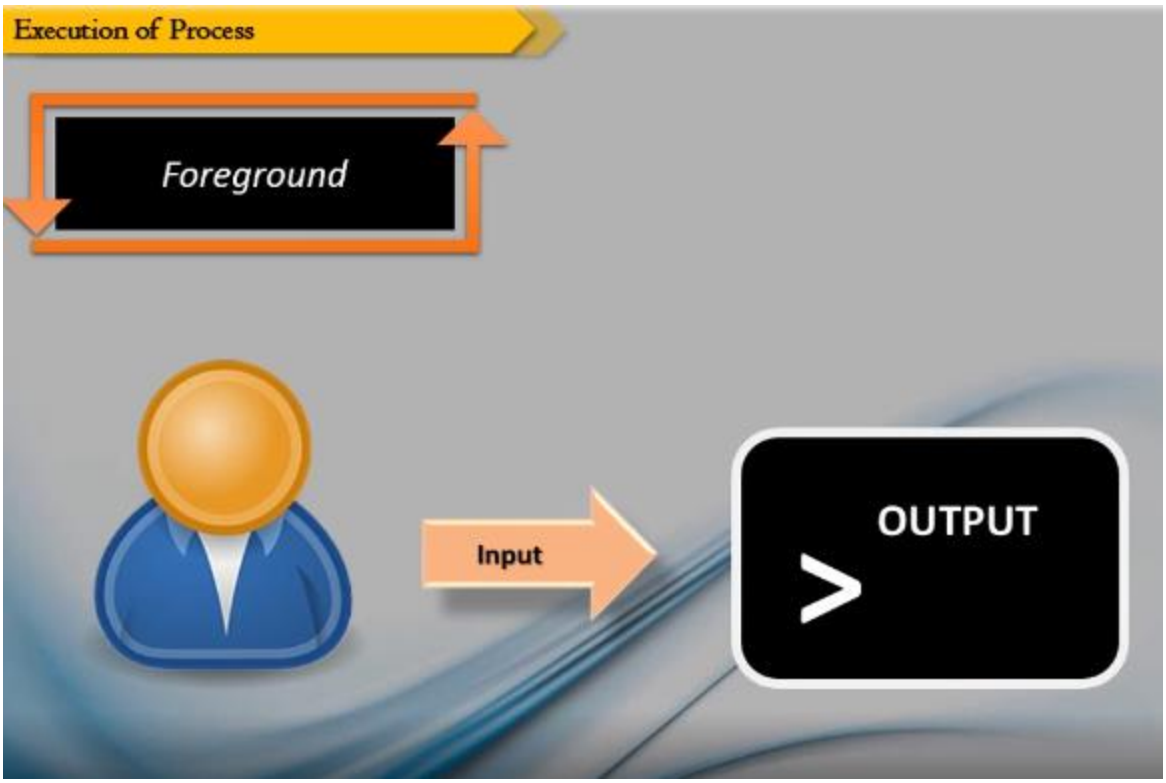
1.pid is the unique identification number that a process can have and is used by the kernel to identify the processes. It gets recycled

2.ppid: processes are created by other processes called as parent process.parent process id.

3.tty:terminal type. Every command is run from a terminal with which it is associated with.It is not mandatory for a process to bee associated with a terminal.these processes are called daemon

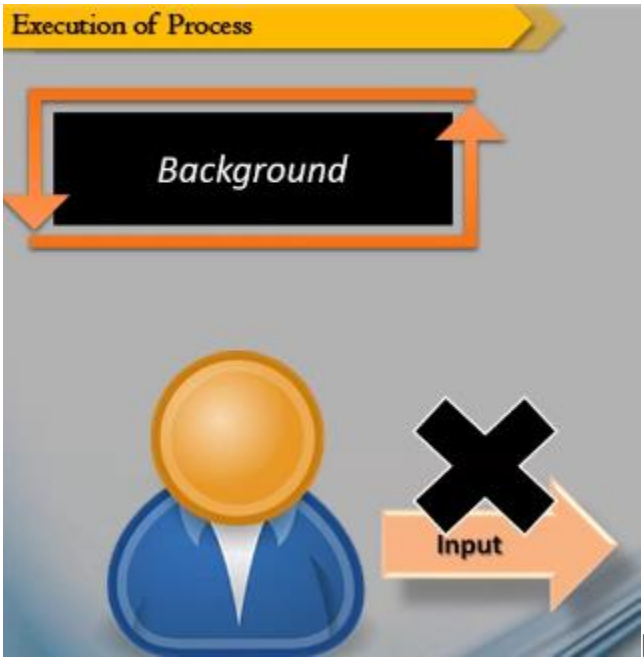
4.Uid: id of the user to whom the process belongs to. Owner of the process can kill the process.Owner defines permissions of accessibility.

5.file descriptor : input output error

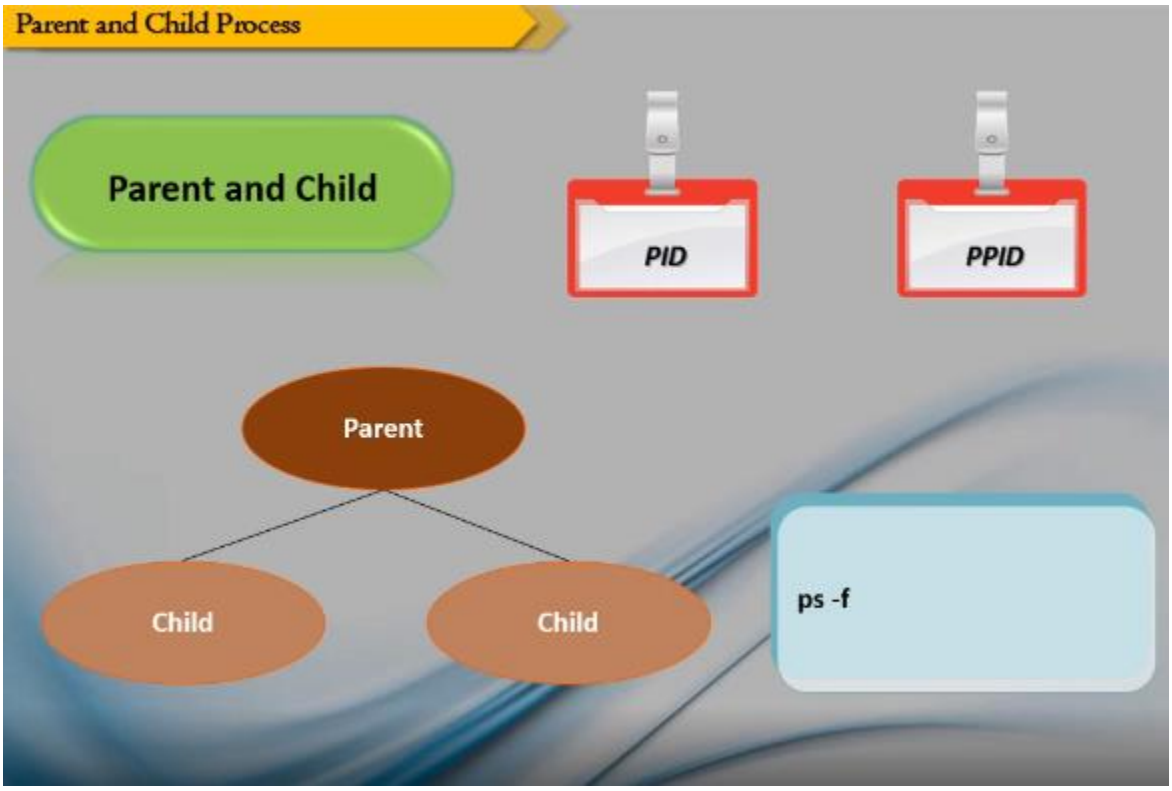


Forground processes takes the input from the user.

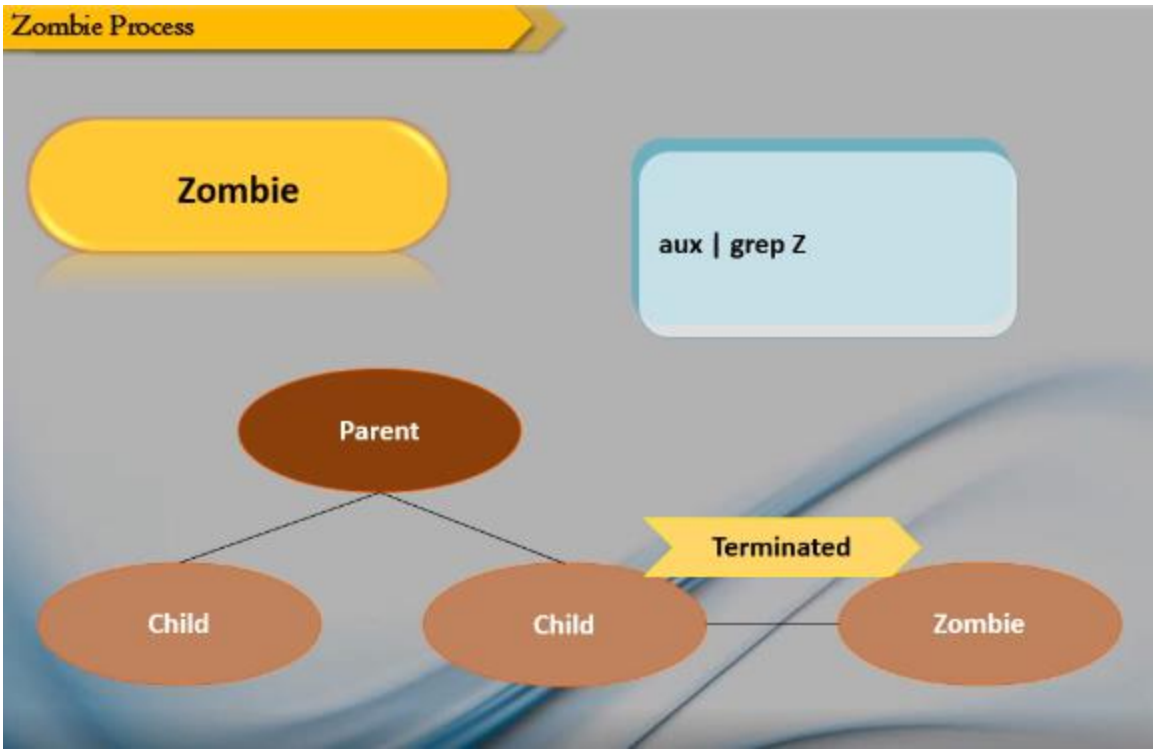
Eg :ls



background does not take input

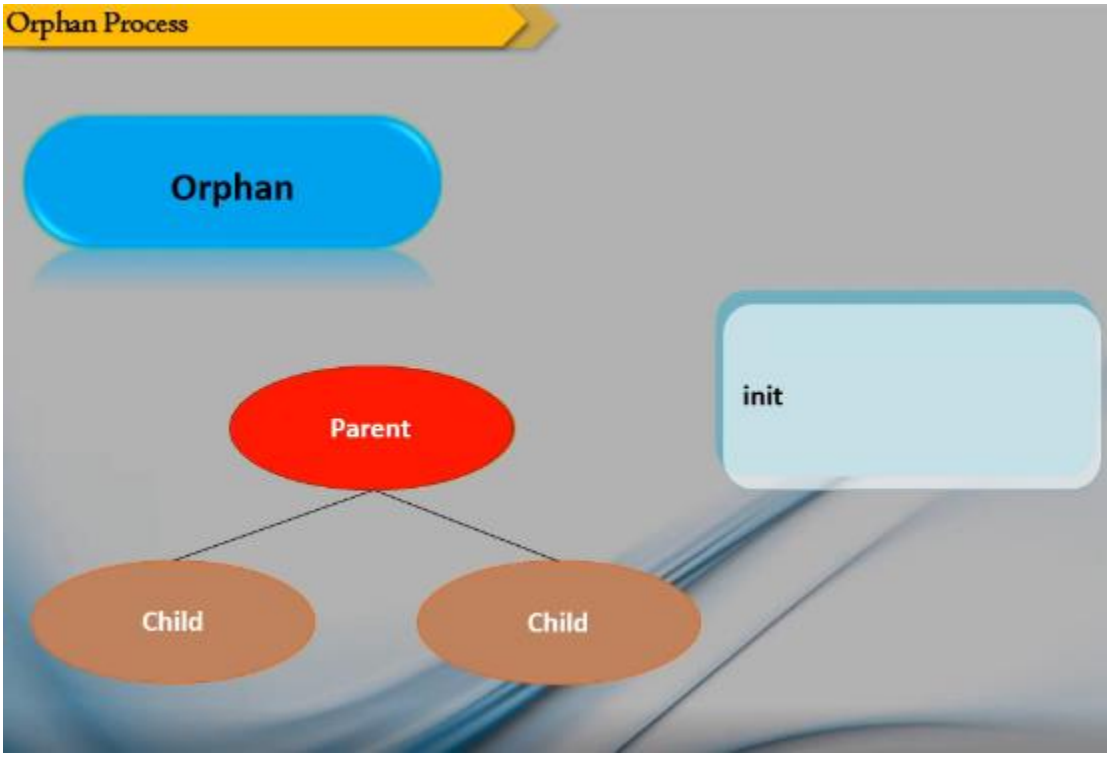


Ps -f can be used to print the pid and ppid

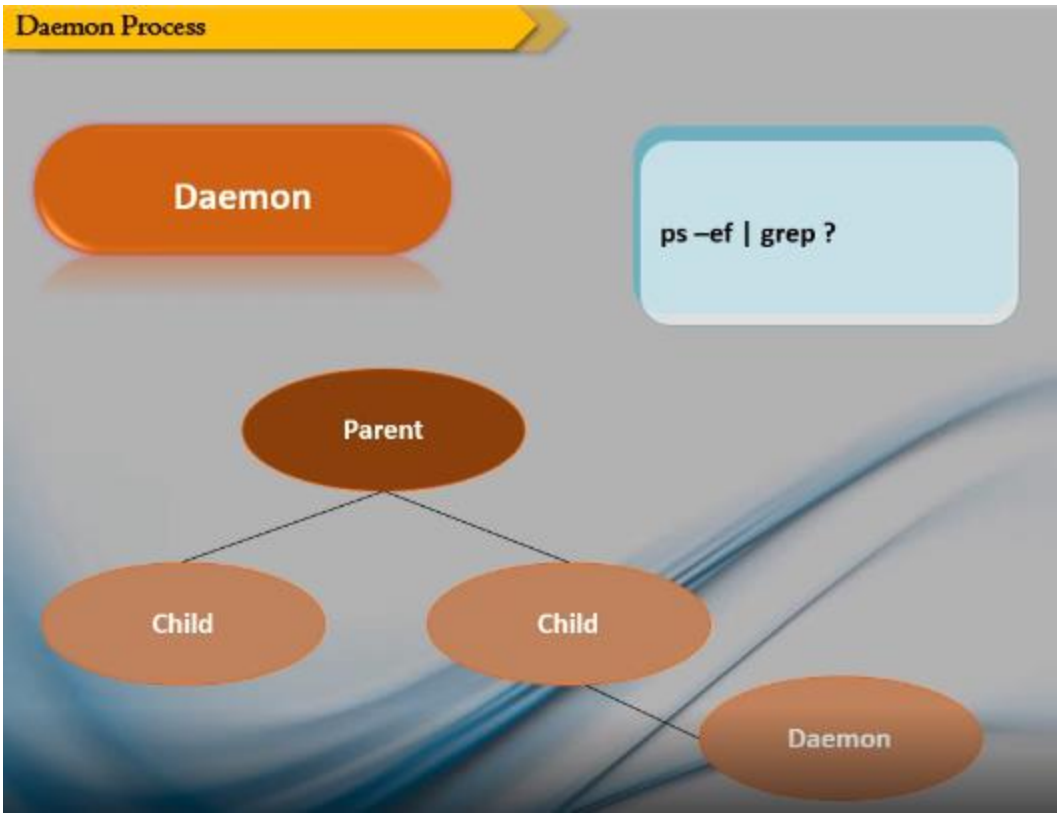


Terminated process is a process which cant be killed. Zombie process.

Above cmd is used to print all the zombie processes.



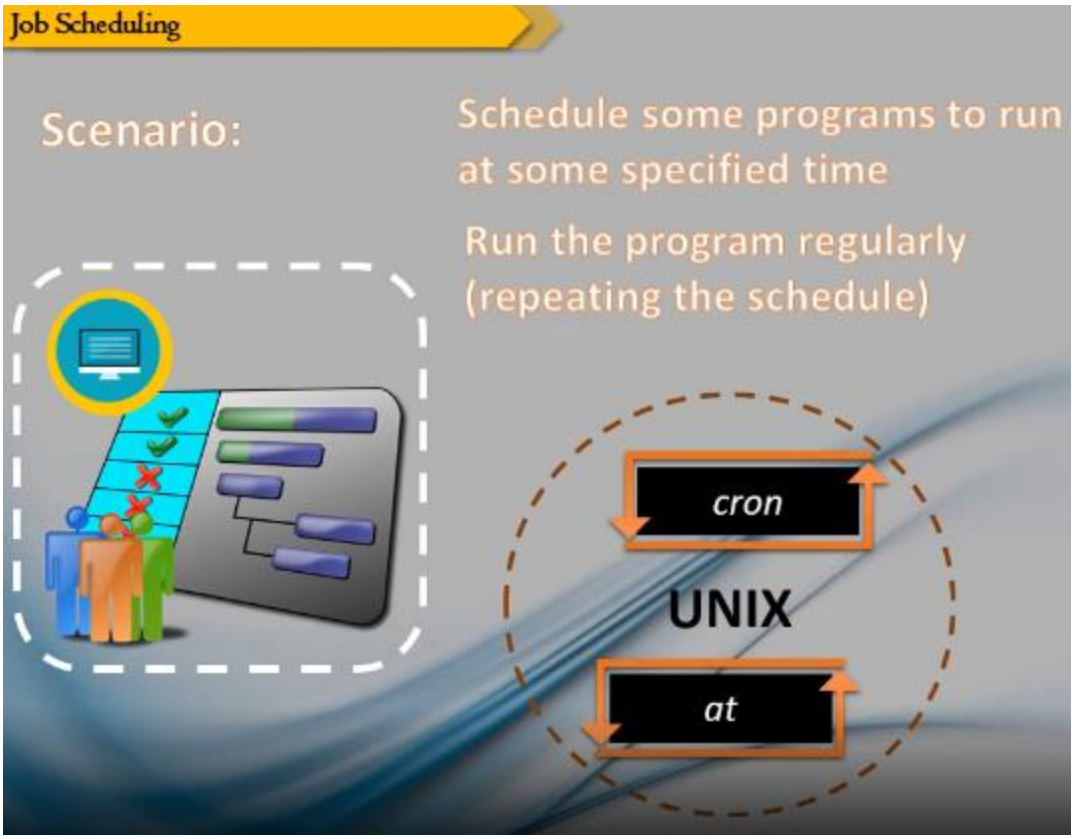
Parent has stopped, child is still running. Init process adopts the orphan processes.



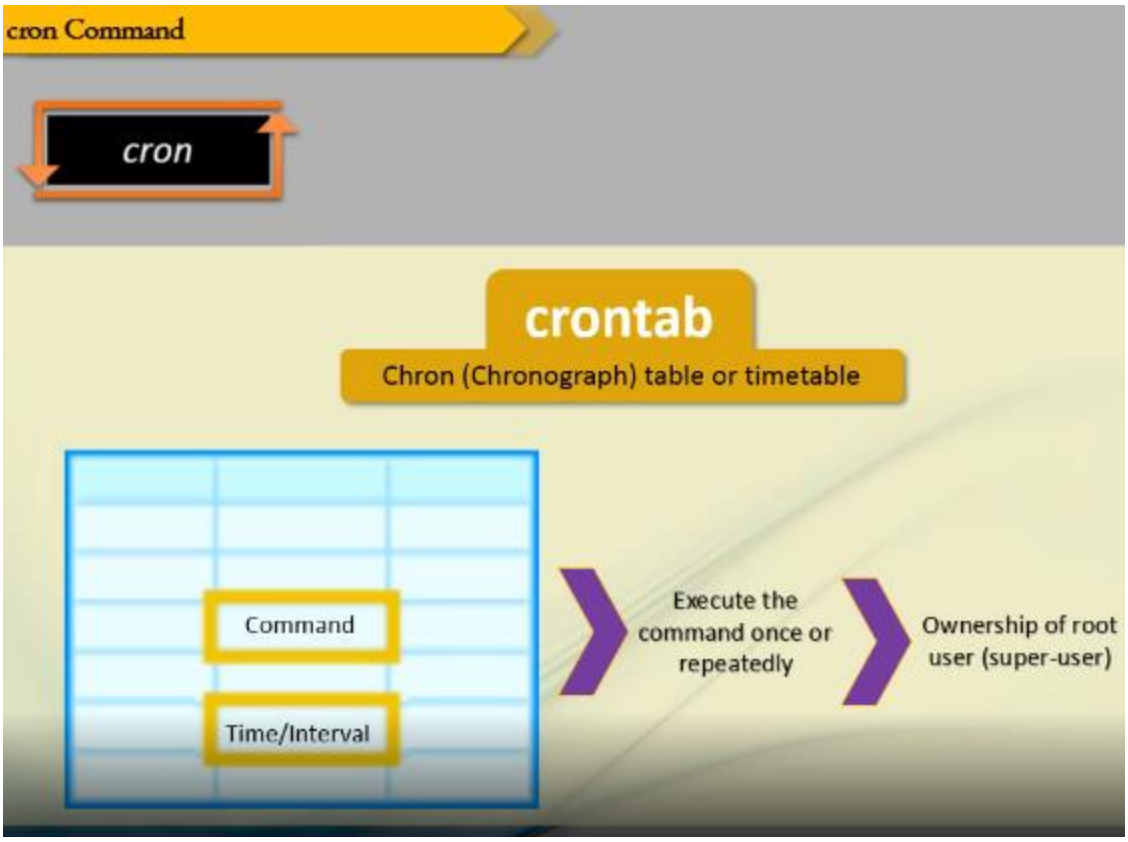
Background or zombie process running for a long time is a daemon process.

Ps -f | grep ? is used to find all the zombie processes

Ps -ef is used to check is a process is daemon or not



Crontab and the at are two utilities used for job scheduling



It is a special table where it is possible to specify time of execution.

cron

crontab [-u user] -option [file]

cron

Option	Usage
-u	Specifies the name of the user. Updates the current user's crontab if the option is not given.
-l	Lists the current crontab contents.
-e	Edits the current crontab content.
-I	Interactive update of crontab content.

cron

components

```
[pts/0][18:31:55:root@inchnilp02 ] /etc> cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

cron

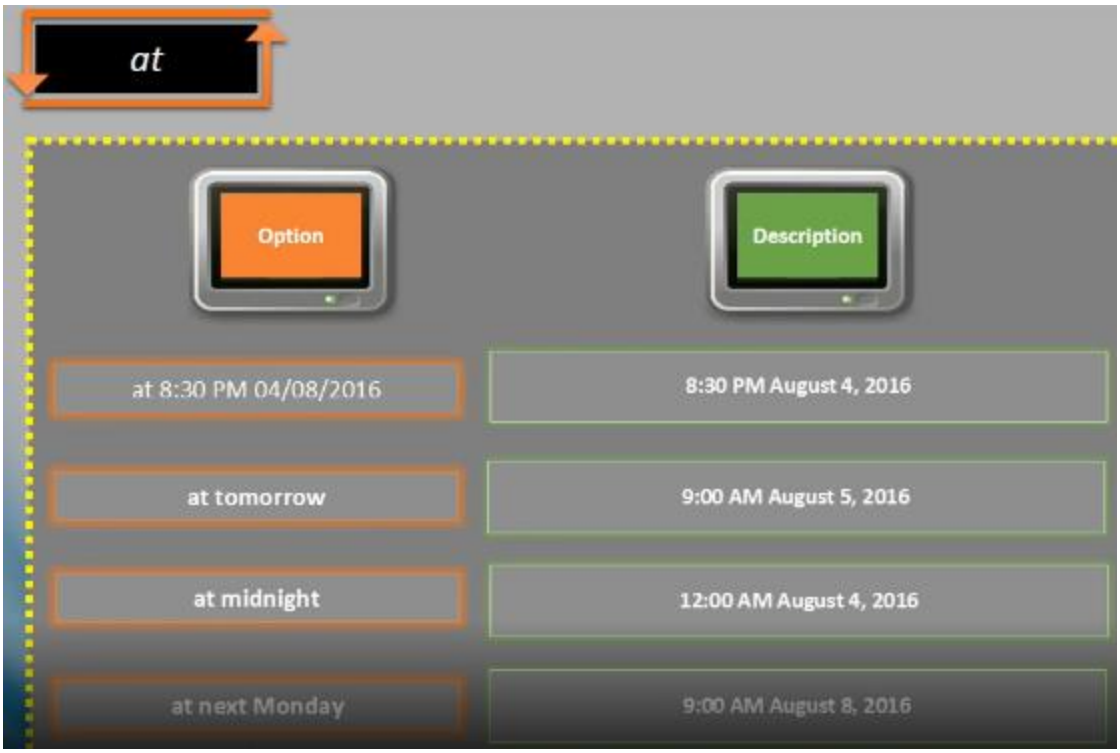
Field	Value	Description
Minute	0-59	The exact minute the command executes
Hour	0-23	The hour of the day, the command to execute
Day	1-31	The day of the month, the command to execute
Month	1-12	The month of the year, the command to execute
Weekday	0-6	The day of the week, the command to execute (Sunday = 0, Monday = 1, and so forth)

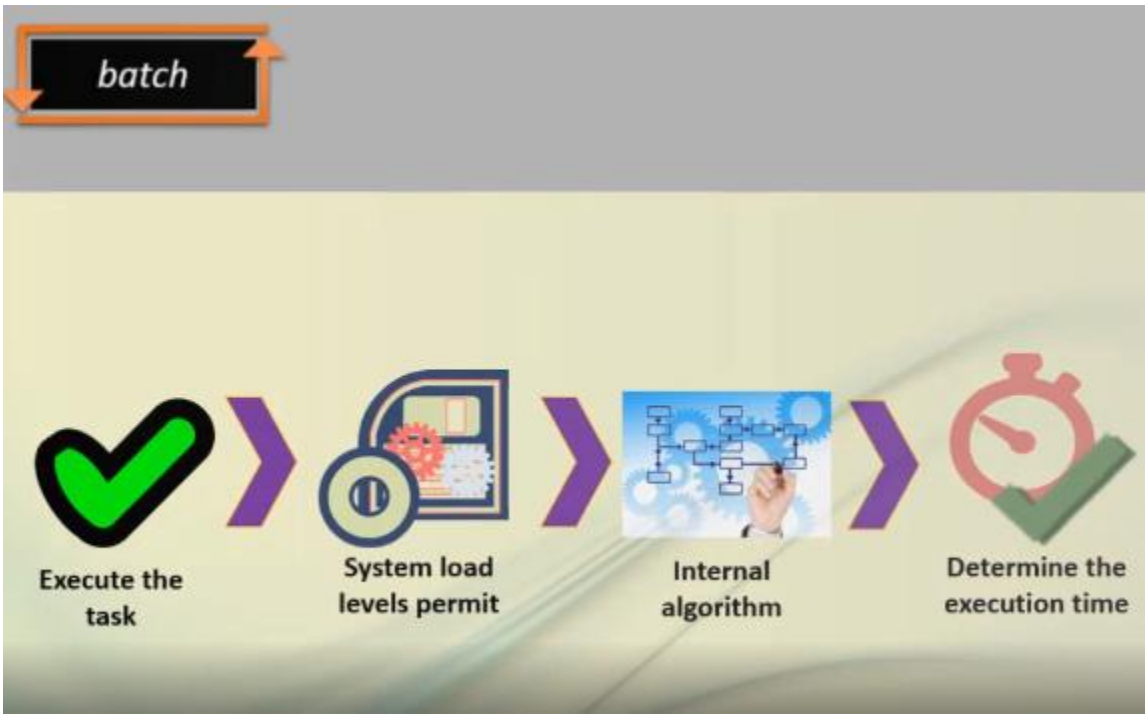
cron

Field	Value	Description
@yearly @annually	Execute once a year	0 0 1 1 *
@monthly	Execute once a month	0 0 1 * *
@weekly	Execute once a week	0 0 * * 0
@daily @midnight	Execute once a day	0 0 * * *
@hourly	Executes every hour	0 * * * *
@reboot	Execute at startup	NA

at

At command takes the time as argument when the job needs to be executed and the job executes at specified time





The batch cmd executes a task when system load permits. It does not take any argument and uses an algo to determine the time required for execution.

batch

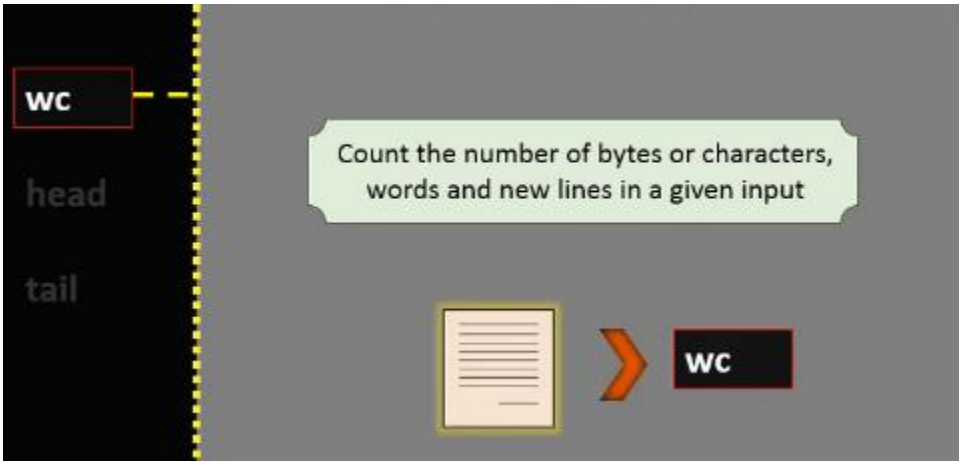
```
[pts/0][18:18:09:root@inchnilp02 ] /etc> crontab -l
00 00 * * * /usr/bin/cfg2html-linux -o /tmp 2> /tmp/cfg2html.err
```

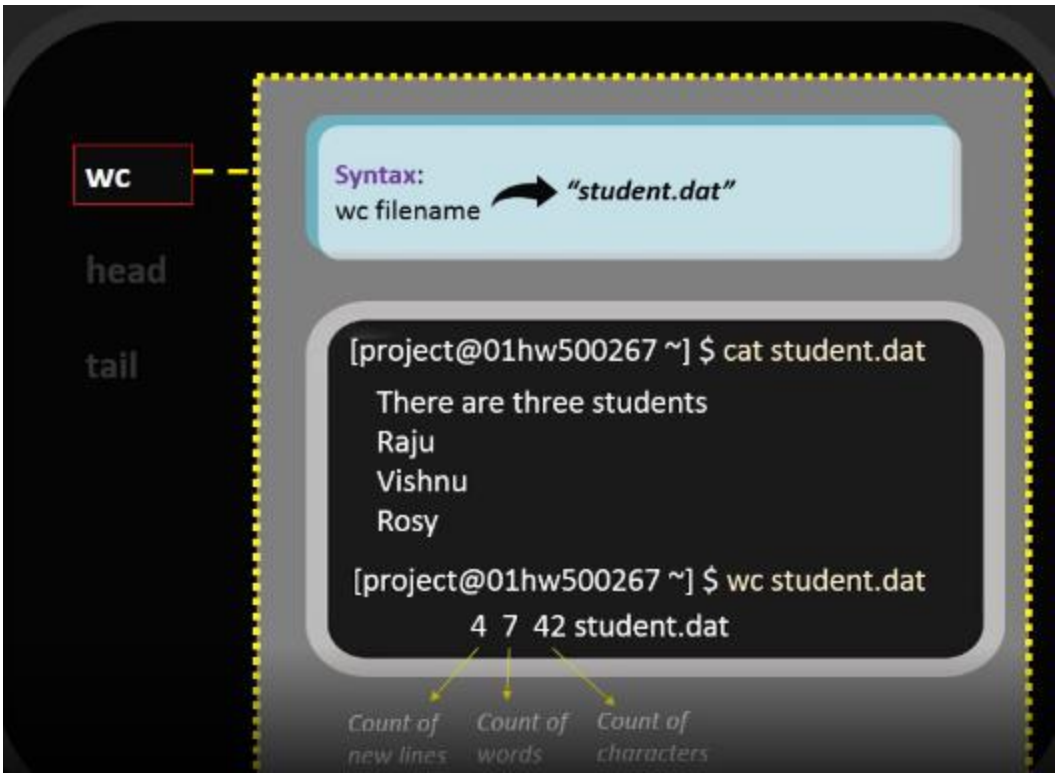
One scheduled job that runs at the midnight everyday

In this video, you learnt that:

- crontab and at are the utilities used for job scheduling in UNIX.
- crontab is a special table used to specify commands and time/interval to execute the command once or repeatedly.
- at command takes time as an argument and executes the job in the specified time.
- batch command does not take any arguments and uses the internal algorithm to execute the task when system load level permits.


Pipes and Filters
wc
head
tail
tail


wc
head
tail
Count the number of bytes or characters, words and new lines in a given input
wc


wc
head
tail
Syntax:
wc filename
"student.dat"
[project@01hw500267 ~] $ cat student.dat
There are three students
Raju
Vishnu
Rosy
[project@01hw500267 ~] $ wc student.dat
4 7 42 student.dat
Count of new lines
Count of words
Count of characters

wc

head

tail

Option	Significance
l	lines (prints the newline count)
c	bytes (prints the byte count)
w	words (prints the word count)
L	max line length (prints the length of the longest line)

wc

head

tail

Syntax:
wc –c filename

```
[project@01hw500267 ~] $ wc –c student.dat
   42 student.dat
[project@01hw500267 ~] $ who|wc –l
```

wc

head

tail

Print lines from the beginning of given input file or files

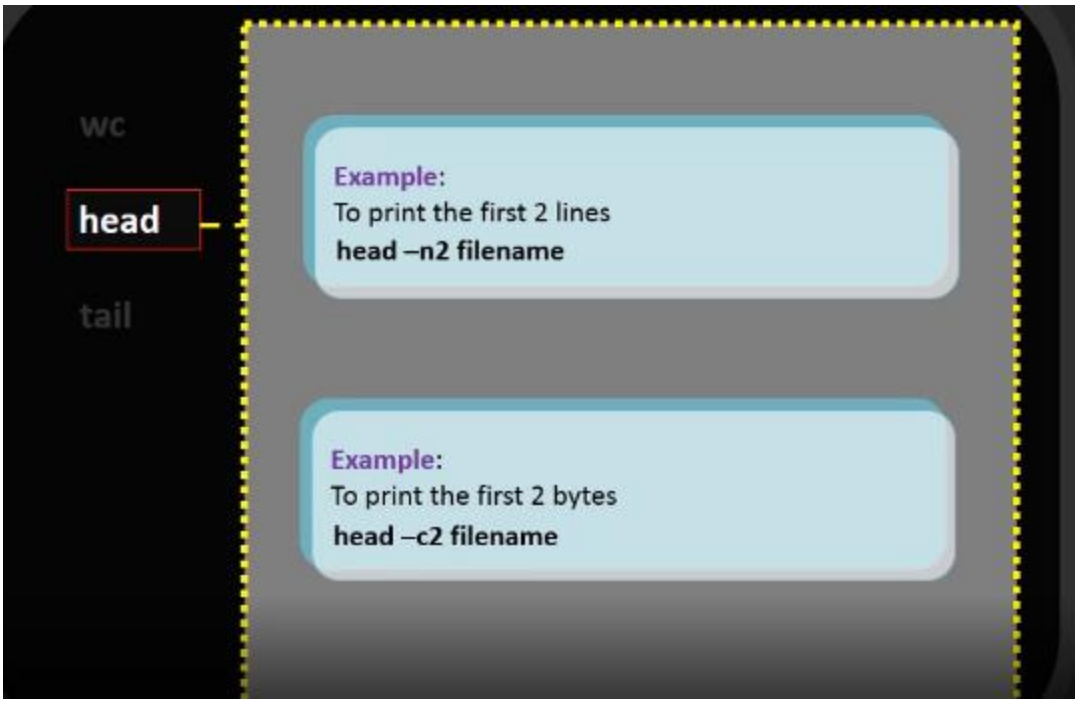
Syntax:
head [options] [filenames]

wc

head

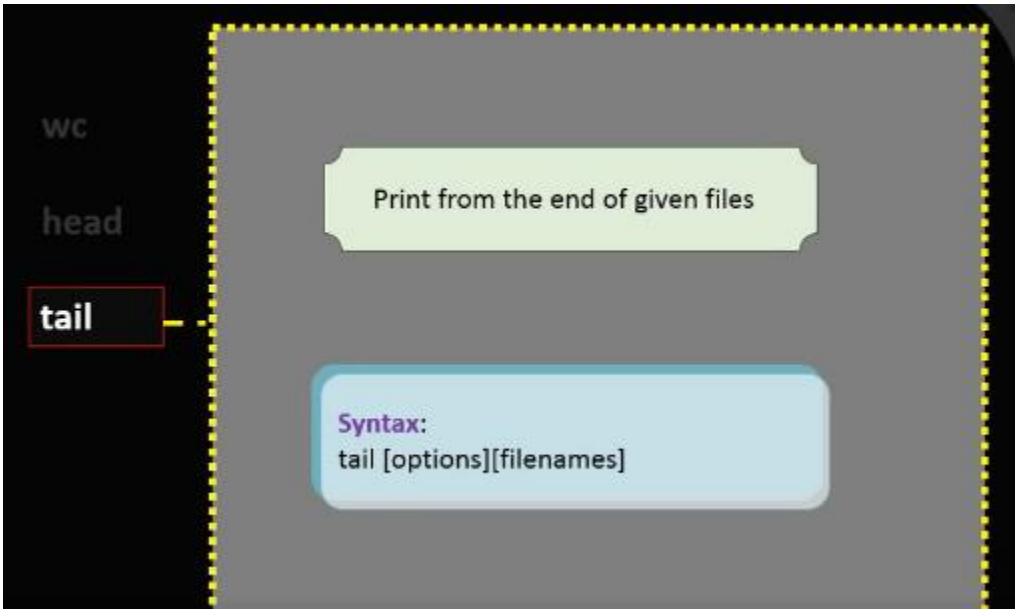
tail

Option	Significance
n	prints the first n lines of each file
c	prints the first n bytes of each file



Without option it prints the first 10 lines of the input file

Without filename it awaits for the input from the user



wc

head

tail

Option	Significance
n	prints the last n lines of each file
c	prints the last n bytes of each file

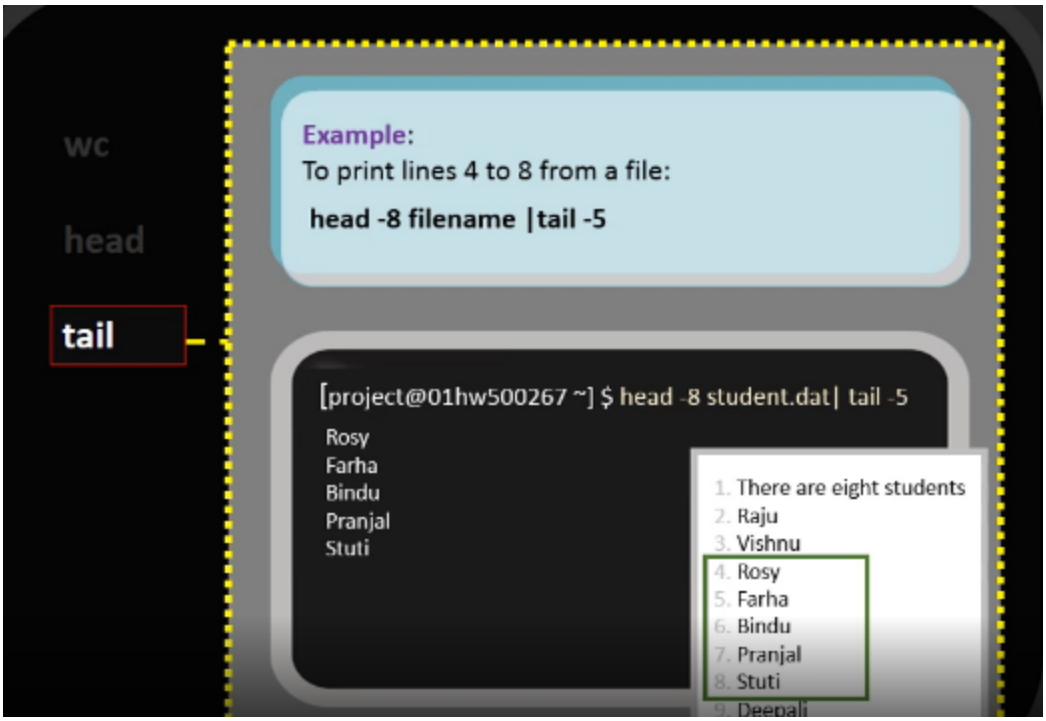
wc

head

tail

Example:
To print the last 2 lines
tail –n2 filename

Example:
To print the last 2 bytes
tail –c2 filename

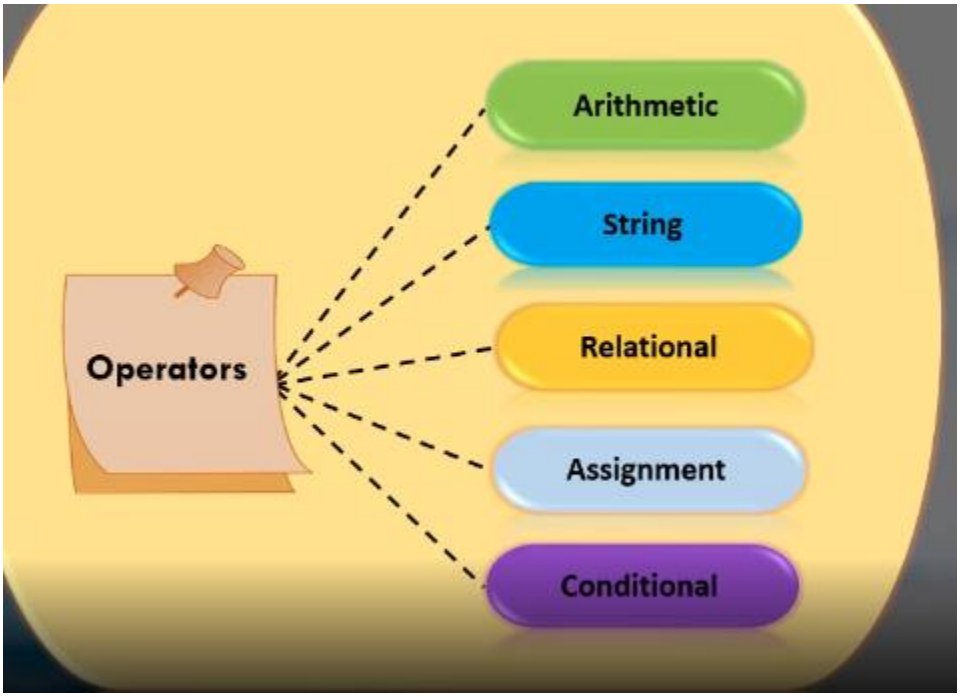


Awk is a powerful tool to filter and manipulate data.

It can perform following actions:

1. Processing input text
2. Formatting report generation
3. Arithmetic and string operations

It is like a programming language



List of built-in variables used along with awk filter:

Built-in Variables	Description
FS	Input field separator
OFS	Output field separator
NF	Specify the number of fields in a line
NR	Specify the number of records in a file
RS	Record separator
ORS	Output record separator
FILENAME	Specify the current file name for reading

Few examples, where awk filter is used:

```
[pts/0][16:39:58:e244121@inchnilp02 ] ~> awk '
> BEGIN {i = 1;
> while (i <= 10) {
> print i " * 5 = " i*5;
> ++i } }'
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

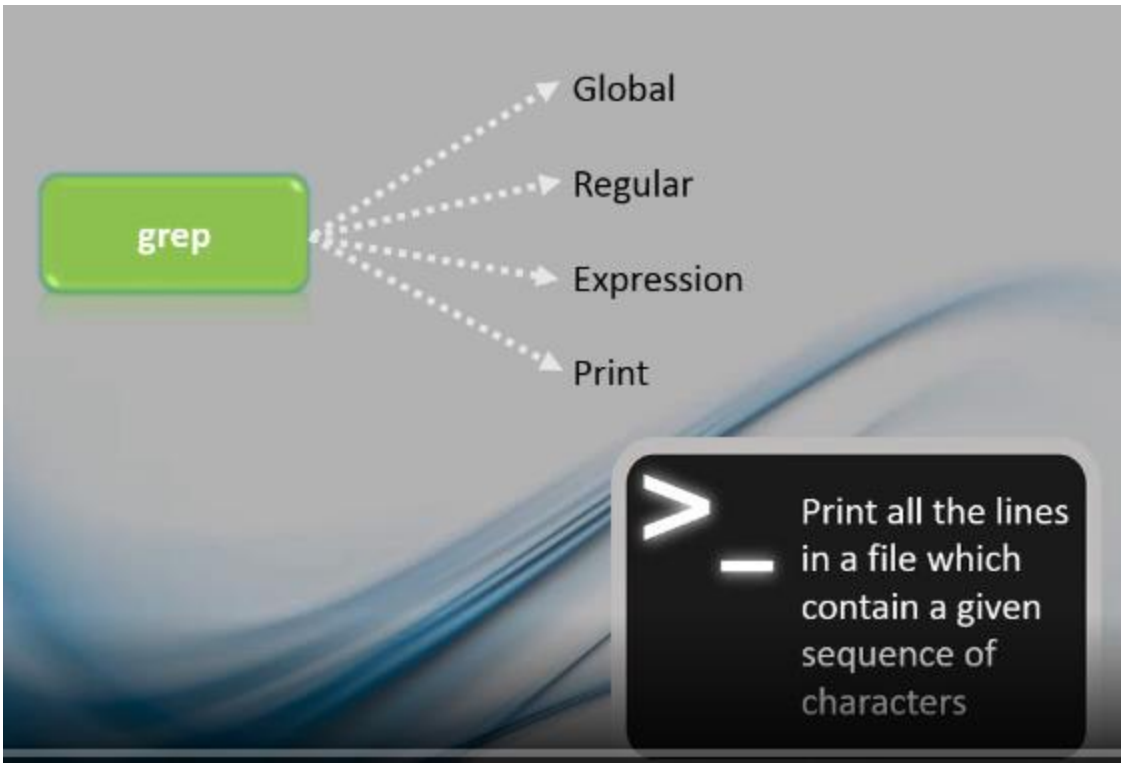
Few examples, where awk filter is used:

```
[pts/0][11:48:11:e244121@inchnilp02 ] ~/trdf> cat fruits.txt
mango
green apple
black grapes
apple
pine apple
[pts/0][11:48:18:e244121@inchnilp02 ] ~/trdf> awk '
> BEGIN {FS=" "}
> {print "Number of fields in record ", NR, " is ", NF}' fruits.txt
Number of fields in record  1  is  1
Number of fields in record  2  is  2
Number of fields in record  3  is  2
Number of fields in record  4  is  1
Number of fields in record  5  is  2
```

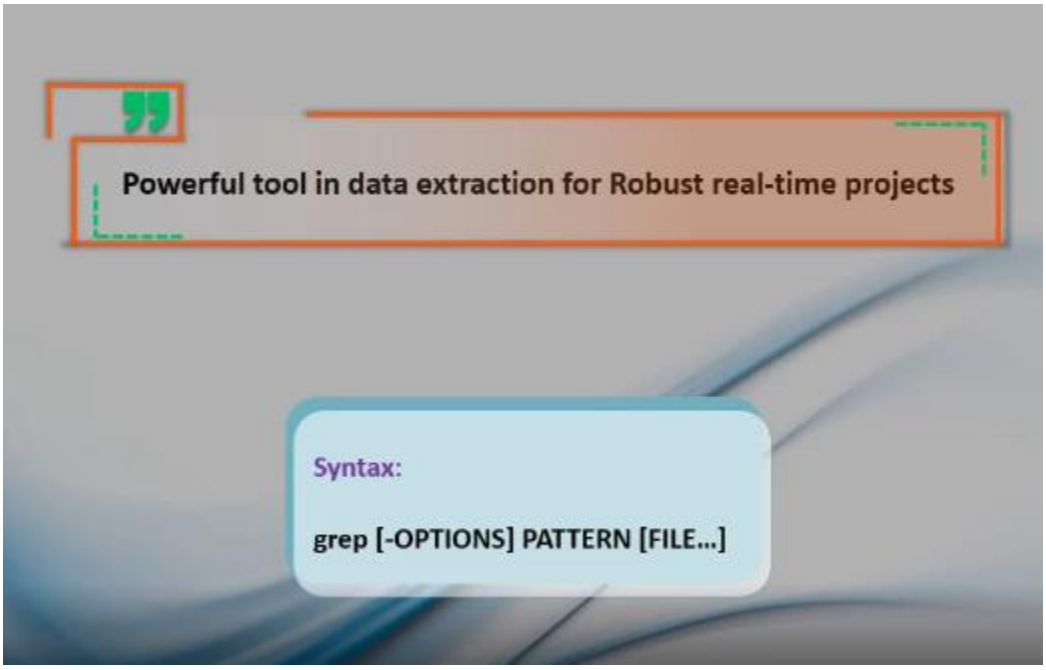
Few examples, where awk filter is used:

```
[pts/0][11:48:39:e244121@inchnilp02 ] ~/trdf> cat emp.txt
Role;Basic;Salary;HRA;DA;Medical
Stweard;5000;40;12;200;12
Chef;7500;40;12;200;12
Manager;12500;40;12;200;12
[pts/0][11:48:46:e244121@inchnilp02 ] ~/trdf> awk '
> BEGIN {FS=";"}
> END {print "No of records in emp.txt file is "NR}' emp.txt
No of records in emp.txt file is 4
```

GREP= global regular expression print


grep
Global
Regular
Expression
Print
Print all the lines in a file which contain a given sequence of characters


grep without options
Names of the file
Respective lines


Powerful tool in data extraction for Robust real-time projects
Syntax:
grep [-OPTIONS] PATTERN [FILE...]

Option	Significance
e	Search for multiple patterns, print the respective lines along with file names and find patterns starting with hyphen (-).
i	Search and print the lines containing pattern irrespective of the case.
c	Count the number of occurrences of a given pattern in the file, where it is searched.
v	Print the lines along with file names, where the given pattern does not occur in the given file(s).

Options and Significances:

Option	Significance
w	Print the lines containing the given pattern, along with file names, wherein the pattern appears as a whole word and not as a part of any other word.
l	Print only the names of the files in which the given pattern is found.
n	Print the line-numbers and file names of those lines, in which the searched pattern is found.
o	Print only the file name and the pattern that is being searched for.

List of Wildcards that are pre-dominantly used along with grep command:

Wildcard	Description
^	Matches the beginning of lines with the searched pattern
$	Matches the end of lines with the searched pattern
.	Matches any single character, where '.' is specified in the searched pattern
*	Matches zero or more occurrences character present before '*' in the searched pattern
[chars]	Matches any one of the characters given in chars, where chars is a sequence of characters

Character-Sets the general forms of the [chars] Wildcard:

Wildcard	Description
[chars]	Matches any one of the characters given in chars, where chars is a sequence of characters

Character-Set	Description
[a-z]	Matches any single lowercase letter, where [a-z] is specified in the given pattern
[A-Z]	Matches any single uppercase letter, where [A-Z] is specified in the given pattern
[a-zA-Z]	Matches any single irrespective of case, where [a-zA-Z] is specified in the given pattern
[0-9]	Matches any single number, where [0-9] is specified in the given pattern
[a-zA-Z0-9]	Matches any single letter or number, where [a-zA-Z0-9] is specified in the given pattern

1

Search for a **letter 'a'** in all the files having file names starting with the word **'Employee'** in the present working directory.

```
[pts/0][11:55:05:e308701@inchnilp02 ] ~> grep a Employee*
EmployeeAgeDetails.txt:Jai     -24
EmployeeAgeDetails.txt:Rajini -60
EmployeeAgeDetails.txt:Ram     -45
EmployeeAgeDetails.txt:Satish -22
EmployeeAgeDetails.txt:Vijay  -26
EmployeeCityDetails.txt:Anish  -Chennai
EmployeeCityDetails.txt:Jai    -Chennai
EmployeeCityDetails.txt:John   -Mumbai
EmployeeCityDetails.txt:Neetu  -Nagpur
EmployeeCityDetails.txt:Nikhil -Chennai
EmployeeCityDetails.txt:Rajini -Trichy
EmployeeCityDetails.txt:Ram    -Kolkata
EmployeeCityDetails.txt:Satish -Nagpur
EmployeeCityDetails.txt:Vijay  -Chennai
EmployeeExperienceDetails.txt:Jai     -2
EmployeeExperienceDetails.txt:Rajini -11
EmployeeExperienceDetails.txt:Ram     -9
EmployeeExperienceDetails.txt:Satish -2
EmployeeExperienceDetails.txt:Vijay  -2
```

2

Search for a **letter 's' or 'S'** irrespective of its case, in the file **'EmployeeCityDetails.txt'** in the present working directory.

```
[pts/0][12:01:52:e308701@inchnilp02 ] ~> grep -i 's' EmployeeCityDetails.txt
Anish  -Chennai
Satish -Nagpur
```

In this video, you learnt that:

- grep is an acronym that stands for Global Regular Expression Print.
- grep command is used to print all the lines in a file which contain a given sequence of characters.
- grep command is used as a powerful tool in data extraction for Robust real-time projects.
- A few Wildcards (^, $, ., *, [chars]) are predominantly used along with grep command.