How To Make The Best Use Of Live Sessions

- Please login on time
- Please do a check on your network connection and audio before the class to have a smooth session
- All participants will be on mute, by default. You will be unmuted when requested or as needed
- Please use the “Questions” panel on your webinar tool to interact with the instructor at any point during the class
- Ask and answer questions to make your learning interactive
- Please have the support phone number (US : 1855 818 0063 (toll free), India : +91 90191 17772) and raise tickets from LMS in case of any issues with the tool
- Most often logging off or rejoining will help solve the tool related issues

COURSE OUTLINE

MODULE 6

INTRODUCTION TO LINUX

INSTALLATION AND INITIALISATION

USER ADMINISTRATION

BOOT AND PACKAGE MANAGEMENT

NETWORKING

LINUX OVERVIEW AND SCRIPTING

LINUX FOR SOFTWARE DEVELOPMENT

SECURITY ADMINISTRATION

Objectives

After completing this module, you should be able to:

- Perform Process Management
- Learn Basic Linux Commands
- Understand system related commands
- Learn basic shell scripting
- Understand basic python and its usage

edureka!

Linux Overview And Scripting

Process Management

edureka!

What Is A Process?

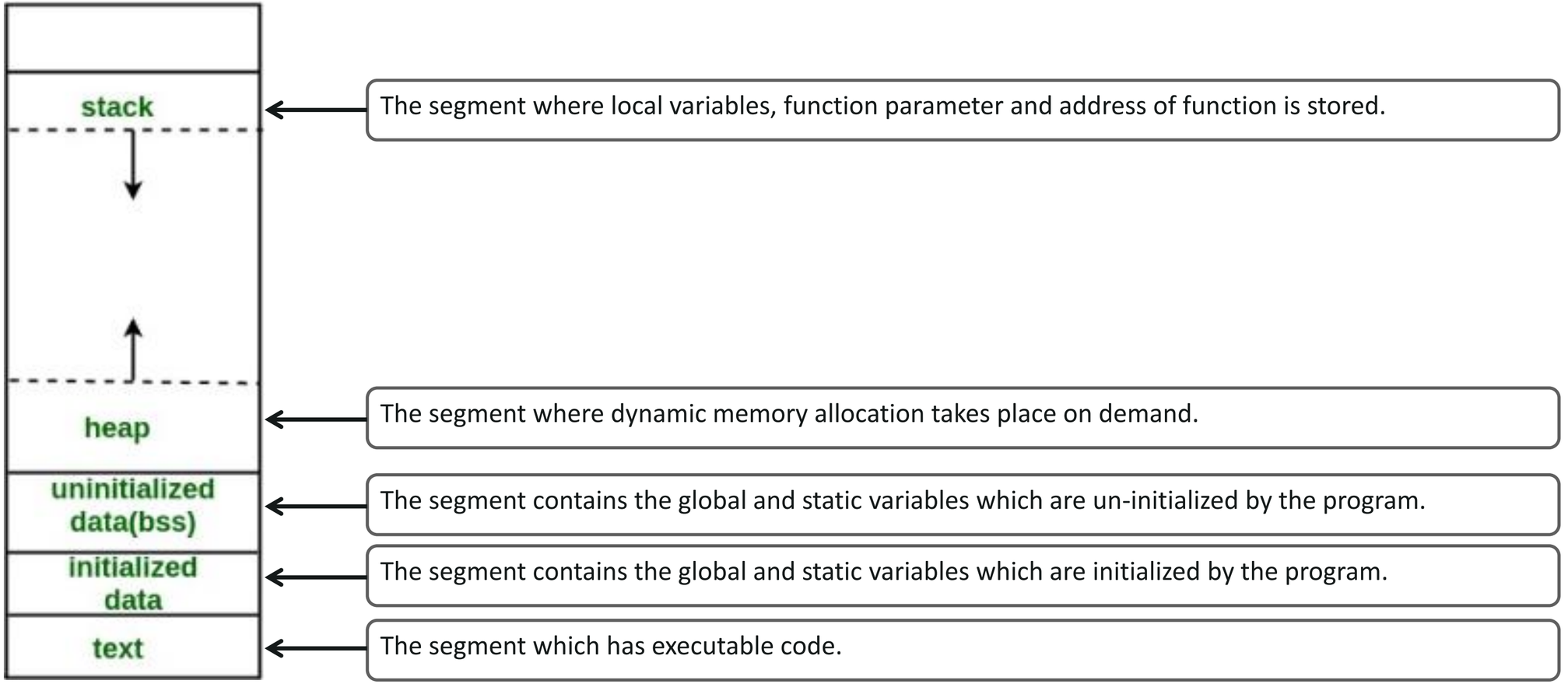
- A process is a series of commands to instruct the processor, what to do.
- It may use any available resources that Linux kernel has and allows to go ahead with the execution.
- The process management involves scheduling, interrupt handling, signaling, process prioritization, process switching, process memory, etc.
- Every process has its own life-cycle such as :
  - Creation
  - Execution
  - Termination
  - Removal

Process States

TASK_RUNNING	The process is executing or about to be executed.
TASK_INTERUPTIBLE	The process is waiting for a signal or a resource.
TASK_UNITERRUPTIBLE	The process is forced to halt for certain condition.
TASK_KILLABLE	Allows waiting task to respond to a signal to be killed.
TASK_STOPPED	The process is stopped by receiving certain signals.
TASK_TRACED	The process is being debugged and temporarily stopped.
EXIT_ZOMBIE	The process is terminated but information available in process table.
EXIT_DEAD	The process is released completely.

Process Memory Area

The process memory area is broadly divided in these segments :



Process Synchronization

Multiple process running on system share resources which may lead to issues like:

Resource blocking

Data inconsistency

Race conditions

Process Synchronization

Process synchronization can be achieved by:

Semaphores

- A variable or data type used to control access to common resource in a multi-processing operating system.
- It is a signaling mechanism.
- Semaphores are mainly of two types : binary semaphore and counting semaphore.

Mutex locks

- Entire section of code is locked and no-one can access it until the lock is released by the process who applied the lock.
- Multiple locks can be applied in a part of the code. It is a locking mechanism.

Ps Command

- Process status is used to check information related to various process running on system.
- It is primarily used for process monitoring.
- The information generally displayed are :
  - PID : unique process ID.
  - TTY : terminal type used is logged into.
  - TIME : amount of CPU in minutes and seconds that the process has been running.
  - CMD : name of the command that launched the process.

Syntax

ps <options>

Ex: # ps –e

ps Command Options

-e	: display every active process.
-f	: display full format list for the process.
-x	: display all process owned by the owner.
-r	: display all running process.
-T	: display all process associated with this terminal.
-g	: display all process owned by the particular group.
-p	: display process by particular PID.
-- forest	: display how processes are linked to each other.

Demo - ps

```
ubuntu@ubuntu#ps
  PID TTY          TIME CMD
 5783 pts/0    00:00:00 bash
28654 pts/0    00:00:00 ps
ubuntu@ubuntu#
ubuntu@ubuntu#
ubuntu@ubuntu#ps -aef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 May06 ?        00:00:02 /sbin/init
root           2     0  0 May06 ?        00:00:00 [kthreadd]
root           3     2  0 May06 ?        00:02:34 [ksoftirqd/0]
root           5     2  0 May06 ?        00:00:00 [kworker/0:0H]
root           7     2  0 May06 ?        00:00:12 [rcu_preempt]
root           8     2  0 May06 ?        00:00:00 [rcu_sched]
root           9     2  0 May06 ?        00:00:00 [rcu_bh]
root          10     2  0 May06 ?        00:00:08 [migration/0]
root          11     2  0 May06 ?        00:00:00 [watchdog/0]
root          12     2  0 May06 ?        00:00:00 [watchdog/1]
```

edureka!

Top

- The command shows the processor and memory being used by the system.

Syntax

```
top
```

- Options Used :
  - -u : append option to display specific user process details.
  - c : press ‘c’ button to view absolute path of the process.
  - d : press ‘d’ button to change the screen refresh interval.
  - Shift+p : press ‘shift’ and ‘p’ to sort based on CPU utilization.
  - -n : append option to limit the number of iteration
  - q : press ‘q’ to exit the window

Ex: # top

Demo - top

```
top - 00:33:17 up 1 day, 13:30,  2 users,  load average: 0.32, 0.91, 0.99
Tasks: 184 total,   1 running, 183 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
GiB Mem :    7.796 total,    2.915 free,    0.097 used,    4.785 buff/cache
GiB Swap:    0.000 total,    0.000 free,    0.000 used.    7.413 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   37784   5856   3996 S   0.0  0.1   0:02.16 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   2:34.47 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   0:12.75 rcu_preempt
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_sched
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:08.64 migration/0
```

Kill

- The command is used to terminate a particular process

Syntax

kill <signal> <PID>

- Common signals sent to a process are:
  - SIGHUP : numeric value 1, hangs up the process.
  - SIGKILL : numeric value 9, kills the process without saving.
  - SIGTERM : numeric value 15, terminates the process.

Ex: # kill -9 2345

- Use ‘ps’ to find the process ID or use the command ‘pidof’

  # pidof mysql

Demo - Kill

```
ubuntu@ubuntu#pidof top
32200
ubuntu@ubuntu#
ubuntu@ubuntu#kill -9 32200
ubuntu@ubuntu#
ubuntu@ubuntu#pidof top
ubuntu@ubuntu#
ubuntu@ubuntu#
```

edureka!

Some Common Process Commands

bg

- Starts running a suspended process in the background.
- # bg

fg

- Brings the suspended or the background running process to the foreground.

fg

jobs

- Displays the status of jobs in the current shell.
- # jobs

edureka!

System Calls

It can be grouped in five major categories

Process Control

- create process,
- load,
- execute,
- terminate, etc.

File Managment

- create file,
- open,
- read,
- write,
- delete, etc.

Device Management

- request device,
- release,
- read,
- write, etc.

Information Maintenance

- get time/date,
- system data,
- process attributes, etc.

Communication

- create connection,
- send,
- receive,
- delete,
- transfer, etc.

System calls runs in different memory-space and with different privileges.

The instructions are generally available as assemble language instructions.

edureka!

Some Linux Basic Commands

edureka!

Output Redirection

- In Linux there are 3 default files for Input/Output(I/O) :

Stdin	: generally attached to keyboard, read input from this file.
Stdout	: generally attached to monitor, send results to this file.
Stderr	: generally attached to monitor, send status/error to this file.

```
//To re-direct standard output to a file use '>' .
# ps -e > file.txt

//To append to a file use '>>'.
//To redirect errors to a file use '2>'.
//To redirect both output and error to a file use '&>'.
```

edureka!

Demo - Output Redirection

```
ubuntu@ubuntu#ls -l /root  2> error.txt
ubuntu@ubuntu#
ubuntu@ubuntu#cat error.txt
ls: cannot open directory '/root': Permission denied
ubuntu@ubuntu#
ubuntu@ubuntu#cat file.txt > redirect.txt
ubuntu@ubuntu#
ubuntu@ubuntu#cat redirect.txt
hello, this is a file.
2 second line.
Third line with upper case.
fourth line with lower case.
ubuntu@ubuntu#
ubuntu@ubuntu#
```

Pipe

Pipe is used to redirect the output of one command as the input for other.

Multiple commands can be combined together to generate a single output.

Pipes are unidirectional and flow from left to right direction.

Some support commands for it are:

less : it shows one scroll length of content at a time. Can move up or down

pg : it shows one page at a time.

more : it shows one scroll length of content at a time. Can only go down.

Sort

Sort command sorts the content of a file line by line

The rules of sorting are :

- Line starting with number will appear before one with the letter.
- Letter coming first in alphabet table will appear first.
- Lowercase has higher preceding the uppercase.

Syntax

sort <filename>

Ex: # sort file.txt

It doesn't modify the file and output can be written in another file.

Rules for sorting can be changed.

edureka!

Demo - Sort

```
ubuntu@ubuntu#cat file.txt
hello, this is a file.
2 second line.
Third line with upper case.
fourth line with lower case.
ubuntu@ubuntu#
ubuntu@ubuntu#sort file.txt
2 second line.
fourth line with lower case.
hello, this is a file.
Third line with upper case.
ubuntu@ubuntu#
ubuntu@ubuntu#
```

edureka!

Tee

It is used to store and view the output of another command.

We can also write output to multiple files

ls | tee file.txt file2.txt file3.txt

It can also be used to pass it to multiple commands :

cat file.txt | tee file2.txt | sed 's/hello/hi/' | sed 's/file/text/'

Syntax

<command> | tee <filename>

Ex: # ls | tee file.txt

edureka!

Demo - tee

```
ubuntu@ubuntu#vi file.txt
ubuntu@ubuntu#cat file.txt
hello, this is a file.
ubuntu@ubuntu#
ubuntu@ubuntu#cat file.txt | tee file2.txt | sed 's/hello/hi/'
hi, this is a file.
ubuntu@ubuntu#
ubuntu@ubuntu#cat file2.txt
hello, this is a file.
ubuntu@ubuntu#cat file.txt | tee file2.txt | sed 's/hello/hi/' | sed 's/file/text/'
hi, this is a text.
ubuntu@ubuntu#
```

Crond

- Cron is a daemon which checks every fixed interval for scheduled tasks in cron table.
- Cron consists of six fields separated by tab or space:
  - Minute (hold values between 0-59).
  - Hour (hold values between 0-23).
  - Day of Month (hold values between 1-31).
  - Month of the year (hold values between 1-12 or Jan-Dec, you can use first three letters of each).
  - Day of week (hold values between 0-6 or Sun-Sat.)
  - Command
- Use “crontab –e” to add a new entry.

Scripting

edureka!

What Is A Script?

- A script is a series of commands within a file which can be executed without being compiled.
- Scripts are general text document in human readable format that need to be executed by a certain program to achieve desired results.
- They can be written with any normal text editor.
- Each has their own specific extension or tags embedded inside the file.
- For ex - .py extension is for python scripts and .sh is for shell scripts.

What Is Scripting?

- Scripting is done to automate the task of a user.
- A sequence of commands which is given as an input on regular intervals can be bundled together as a script and executed as a single file when required.
- This reduces the chances of typing error while pacing-up the process of command execution as the next command is immediately executed after finsing the current execution.
- Scripts can provide with help/suggestions in case of the user doing some error with a particular command.

Scripting In Linux

- Each distribution of Linux has shell which executes commands.
- We provide a command to shell as input and it executes them.
- The shell supports scripting and we can bundle a set of commands to be executed together by writing in a file with shell details.
- Each script should have execute permission set to it.
- The script can also be added to be executed during boot or specific process execution.
- We can also install various tools like python and write their own scripts.

BASH Scripting

edureka!

Bash Script

The first line is generally used to define, it is a bash script.
#!/bin/bash

Anything after # is not executed and used as a comment.

The script must have execution permission assigned to it.

Anything that works on command line can be a part of bash script and vice-versa.

Bash scripts instructs bash to run a set of commands sequentially.

Bash scipts have an extension of .sh

Variables In Bash

A variable is a temporary memory block used to store information which is present for that block of code.

We perform two actions on variables :

- Set the value of it.
- Read the value of it.

Bash works line by line. For every line it identifies the variables and replaces it with its value and then re-iterates the process in next line

When reading a variable, place a '$' symbol before it.

```
# echo $name
```

Special Variables In Bash

$0 - The name of the Bash script.

$1 - $9 - The first 9 arguments to the Bash script. (As mentioned above.)

$# - How many arguments were passed to the Bash script.

$@ - All the arguments supplied to the Bash script.

$? - The exit status of the most recently run process.

$$ - The process ID of the current script.

$USER - The username of the user running the script.

$HOSTNAME - The hostname of the machine the script is running on.

$SECONDS - The number of seconds since the script was started.

$RANDOM - Returns a different random number each time is it referred to.

$LINENO - Returns the current line number in the Bash script.

Input In Bash

Use command ‘read’ to ask user to provide an entry for the variable.

We can provide multiple variable to read in a go by separating them with whitespaces

Some options read provides are :

Syntax

read <options> <variable_name>

Ex: # read student_id

-p : allows you to specify a prompt.

-s : makes input silent and doesn’t display on screen.

-r : backslash doesn’t act as an escape character.

edureka!

Arithmetic In Bash

- We use to double parenthesis to do arithmetic calculations.
- **'let'** is a built-in function to do simple arithmetic.

  # let a--

  # let a = $2 – 21
- **'expr'** is similar but it prints the result than assigning to a variable.

  # expr 30 % 2

Syntax

$((expression))

Ex: # a = $((4*5))

If Statement In Bash

If statement helps to decide to run a piece of code based upon conditions that we may set.

Case is also supported and works similar to a C program.

Syntax

```
If [ <test_condition> ]
then
        <command>
elif [ <test_condition_2> ]
then
        <command_2>
else
        <command_3>
fi
```

Loop in bash

While loop is easiest to support and works till the expression is true.

'While' Syntax

```
while [ <condition> ]
do
        <commands>
done
```

'Until' and 'for' loop is also supported.

'for' Syntax

```
for var in <list>
do
        <commands>
done
```

Function in bash

Function is a block of code which can be re-used multiple times.

No argument can be passed in functions in bash.

The function definition must appear in the script before any calls to the function.

Syntax

```
function function_name {
<commands>
}
```

Ex:

```
print () {
echo this is a function
}
```

Demo - bash

```
ubuntu@ubuntu#vi script.sh
ubuntu@ubuntu#chmod 755 script.sh
ubuntu@ubuntu#
ubuntu@ubuntu#cat script.sh
#!/bin/bash
# declare STRING variable
STRING="Hello World"
#print variable on a screen
echo $STRING
ubuntu@ubuntu#
ubuntu@ubuntu#
ubuntu@ubuntu#./script.sh
Hello World
ubuntu@ubuntu#
```

edureka!

Demo - bash (continued)

```
ubuntu@ubuntu#vi count.sh
ubuntu@ubuntu#chmod 755 count.sh
ubuntu@ubuntu#
ubuntu@ubuntu#./count.sh file.txt
./count.sh counts the lines of code
file.txt: 4
1 files in total, with 4 lines in total
ubuntu@ubuntu#
```

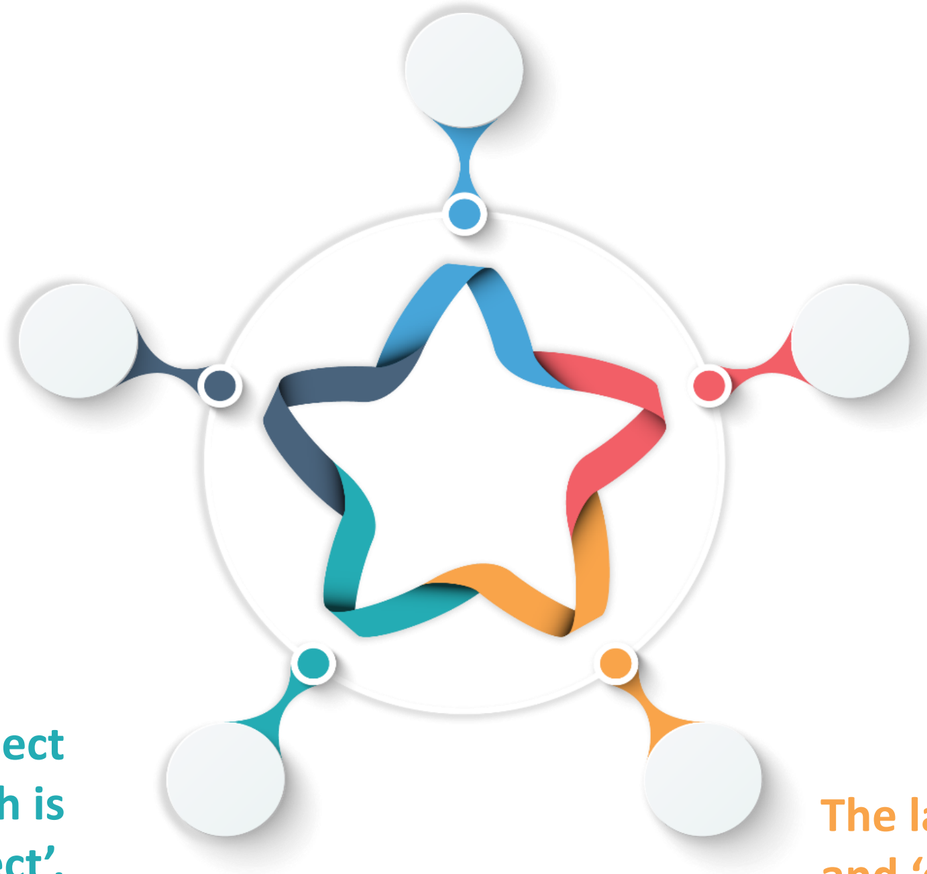
Expect Scripting

edureka!

Expect

Expect is useful to automate the process which requires interaction between the user and the program.

The Expect script expects an input and sends an automated response without interacting with user.



The script should be given executable permissions.

The first line defines the expect command path which is '#!/usr/bin/expect'.

The last line should be the end of the file and 'eof' is used to signify it.

Commands Used By Expect Script

Spawn	: The spawn command is used to start a script or program.
Expect	: The Expect command waits for input.
Send	: The send command is used to send a reply to a script or a program.
Interact	: The interact command allows you to define a predefined user interaction.

Basic Usage Of Expect Script : Example

script.sh

```
#!/bin/bash
echo “What is you name?”
read $reply
echo “Enter your password?”
read $reply
```

To execute :

chmod 755 expect_script

./expect_script

expect_script

```
#!/usr/bin/expect
set timeout -1
 spawn ./script.sh
expect “What is you name?”
send  “Tom\r”
expect “Enter your password?”
send  “tom123\r”
expect eof
```

edureka!

Conditional Tests

It supports conditional test if there is a “if” statement or multiple option to be sent from the script.

It also supports if-else statements. Loops are also supported.

Syntax

```
expect {
        “option_one" { send "send fist reply\r" }
        “option_two" { send "send second reply\r" }
}
```

Interact Command

Some sensitive information might not be put in the script. In such cases, script should ask for the data from user and then proceed as usual.

The interact command reverts the control back to the keyboard.

When interact is executed, Expect will start reading from the keyboard.

Syntax

```
interact
```

Ex:

```
# ./interact
```

autoexpect

Autoexpect works like expect, but it builds the automation script for you.

The script to be automated is passed as an argument and based on the replies to the input asked, the script is generated.

The generated file can be used to interact.

Syntax

autoexpect ./script.sh

Ex:
autoexpect ./script.sh

DEMO - Expect

edureka!

Demo - Expect

Generating the expect script from autoexpect

```
ubuntu@ubunt#autoexpect ./script.sh
autoexpect started, file is script.exp
What is your name?
Tom
Enter your password?
Tom123
autoexpect done, file is script.exp
ubuntu@ubunt#
```

Demo - expect (continued)

Generated Script

```
set timeout -1
spawn ./script.sh
match_max 100000
expect -exact "What is your name?\r
"
send -- "t "
expect -exact "
send -- "Tom\r"
expect -exact "Tom\r
Enter your password?\r
"
send -- "Tom123\r"
expect eof
ubuntu@ubunt#cat script.exp
```

Executing the script

```
ubuntu@ubunt#chmod 755 script.exp
ubuntu@ubunt#./script.exp
spawn ./script.sh
What is your name?
Tom
Enter your password?
Tom123
ubuntu@ubunt#
```

Python

edureka!

Python

- Python is an object-oriented programming language created by Guido Rossum in 1989.
- It provides easier to read syntax than other programming languages.
- It is a platform independent scripted language with full access to operating system API's.
- Python can be compiled to byte-code for embedded or large applications.

Python

There is no compilation step, so testing and debugging is faster.

Use command apt-get install python to install python in Linux.

Python has two versions 2 & 3. For beginners there is not much difference than some basic syntax.

Python Scripting

- Type ‘python’ go into the interactive command line mode. One can enter the basic python commands and execute them.
- Python scripts ends with .py extension.
- To run a file use ‘python <filename>

  # python file.py
- Python uses indentation to indicate a block of code.
- Appending text after ‘#’ will comment that part of text.
- To have multi-line comment use tripe quotes.

edureka!

Python Variables

A variable name must start with a letter or the underscore character.

A variable name can only contain alpha-numeric characters and underscores.

Variable names are case-sensitive.

Ex:
age = 21
name = “Tom”

Casting Data Types

Casting is done by using pre-defined functions such as :

int()	: constructs an integer from integer, float or string.
float()	: constructs a float from integer, float or string.
str()	: constructs a string from integer, float or string.

Ex:

```
temp = int(“48”)
temp = int(5.0)
temp = float(48)
temp = str(7.3)
```

String Operations

String in python can be surrounded by single quotes or double quotes.

Some of the operation that can be performed on a string variable ‘A’ are :

A[2]	: get the character at 2nd position.
A[2:5]	: get character from position 2 to 5.
A.strip()	: removes whitespace from beginning and the end.
len(A)	: returns the length of the string.
A.lower()	: returns string in lowercase.
A.upper()	: returns string in uppercase.
A.split(“,”)	: split strings in substring if finds the particular seperator.
A.replace(“M”, “N”)	: replace all M with N.

List

It is a collection which is ordered, allows modification and allows duplicate members.

Some options to be used on the list ‘alist’ are :

alist [2]	: to access or modify the 3rd element of the list. List starts from ‘0’.
alist.append(“four”)	: to add an entry in the list.
alist.remove(“one”)	: to remove the entry one from the list.
len(alist)	: returns the number of elements in the list.

Ex:
list = [‘one’, ‘two’ , ‘three’]

Dictionaries

A dictionary is a collection which is unordered, changeable and indexed and have keys and values.

Some options to be used on the list ‘adict’ are :

adict[‘name’]	: to access or modify the value at key name.
adict[“member”]=“No”	: adds a new key member if not already present.
del(adict[“location”])	: remove the entry with key ‘location’.
len(adict)	: returns the length of the dictionary.

Ex: # dict = { “name” : “tom” , “age” : “21” , “location” : “Antarctic” }

Conditionals

Python supports usual logical conditions from mathematics.

The conditions are used in multiple ways but most generally with if statements.

Ex:

```
#if a > b:
        print(“a is greater than b")
elif a == b:
        print("a and b are equal")
else:
        print(“b is greater than a")
```

edureka!

Loops

Python supports for and while loops.

Ex:

```
# for x in range(2, 6):
        print(x)
# Numbers = [“one”, “two", “three"]
    for x in Numbers:
        print(x)
# num = 2
```

Ex: while num < 16:
print(num)
num += 3

Functions

Function is a block of code which runs only when it is called.

It is defined using the ’def’ keyword.

We can pass a parameter and assign it default values just like a ‘C’ program.

```
Ex:  # def print_hello():
                print("Hello")
```

Import

Import in python is similar to #include header file in C.

Python modules can get access to code from another module by importing the block of code.

When import is used, it searches for the module initially in the local scope by calling __import__() function.

The value returned by the function are then reflected in the output of the initial code.

```
Ex: # import math
    print(math.pi)
```

DEMO - Python

edureka!

Demo - Python

```
ubuntu@ubuntu#
ubuntu@ubuntu#vi sum.py
ubuntu@ubuntu#cat sum.py
num1 = 1.5
num2 = 6.3

sum = float(num1) + float(num2)

print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
ubuntu@ubuntu#
ubuntu@ubuntu#python sum.py
The sum of 1.5 and 6.3 is 7.8
ubuntu@ubuntu#
ubuntu@ubuntu#python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello"
Hello
>>> exit()
ubuntu@ubuntu#
```

edureka!

Demo - Python (continued)

Script to measure average CPU percentage consumed by specific process

```
ubuntu@ubuntu#
ubuntu@ubuntu#cat usage.py
import os

CPU_Pct=str(round(float(os.popen('''grep 'cpu ' /proc/stat | \
                awk '{usage=($2+$4)*100/($2+$4+$5)} END {print usage }' ''').readline()),2))

    #print results
print("CPU Usage = " + CPU_Pct)
ubuntu@ubuntu#
ubuntu@ubuntu#
ubuntu@ubuntu#python usage.py
CPU Usage = 15.99
ubuntu@ubuntu#
```

edureka!

Tcl / Tk

edureka!

Tcl

Tool Command Language is shell application that reads TCL command from its standard input or from a file and gives desired results.

TCL program should have a .tcl extension.

The source code is compiled into bytecode, which is later interpreted by the Tcl interpreter.

Tcl is a string based scripting language.

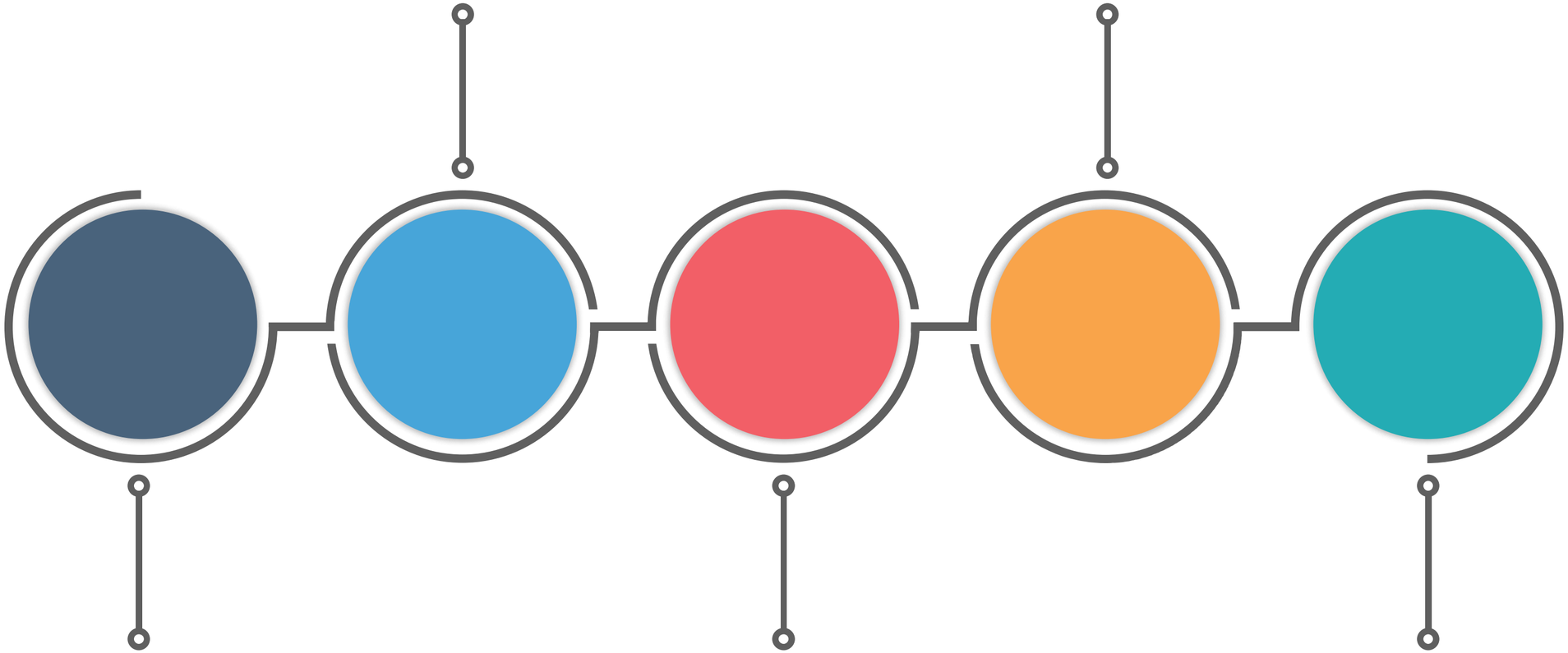
Tk is an open source, cross-platform widget toolkit that provides a library of basic elements for building a graphical user interface (GUI).

edureka!

Tcl Scripting

Use ‘exit’ keyword to come out of it.

The script should have ‘.tcl’ extension and should have executable permission.



Type ‘tlsch’ to enter the interactive mode of tcl.

Every script should have ‘#!/usr/bin/tclsh’ at the top.

‘puts’ is used to print on the output console.

Sample Tcl Script

Example of hello.tcl :

```
#!/usr/bin/tclsh
puts "Hello World")
```

Variable

A variable is an identifier which holds a value

```
//To set a value use ‘set’ keyword.
# set name “tom”

//To print a value use ‘puts’ and a ‘$’ symbol before the variable.
# puts $name

//Curly Braces ‘{}’ have special meaning and substitution of words is
disabled inside braces.
# puts {$name}                                  //prints ‘$name’

//Square brackets, [], are used to create nested commands.
# puts “a : [set b [set c 10]]“                 //  prints  ‘a : 10’ , b and c
//also have value as 10.
```

Conditionals

If statement consists of Boolean expression followed by one or more statements.

```
Ex:
        # set num 10
        if {$num < 20}
        { puts “Number is less than 20" }
        else
        { puts “Number is greater than or
equal to 20” }
```

The switch statement enables a variable to be tested for equality against a list of values.

```
Ex:
        # switch $alphabet {
                a { puts “character a" }
                b { puts “character b" }
                default { puts “character
unknown" }
        }
```

edureka!

Loops

For command executes a sequence of statements multiple times based upon a counter value.

Ex:

```
# for {set i 0} {$i < 5} {incr i}
{ put $i }
```

In while loop, a given block of code repeat till condition is true.

Ex:

```
# set a 10
While {$a < 5} {
        Puts "a is $a"
        incr a }
```

DEMO - Tcl

edureka!

Demo - tcl

```
ubuntu@ubuntu#
ubuntu@ubuntu#vi tcl_script.tcl
ubuntu@ubuntu#chmod 755 tcl_script.tcl
ubuntu@ubuntu#
ubuntu@ubuntu#./tcl_script.tcl
42
You are an adult now
ubuntu@ubuntu#
ubuntu@ubuntu#
ubuntu@ubuntu#cat tcl_script.tcl
#!/usr/bin/tclsh

set age 42
puts $age

if {$age < 18} {
    puts "You are a child"
} else {
    puts "You are an adult now"
}
ubuntu@ubuntu#
ubuntu@ubuntu#
```

edureka!

Quiz

1. How would you show process hierarchy in forest format?

   a. ps –ef tree

   b. ps –f

   c. ps –ef - - forest

   d. ps –ef -t

Answers

A

1. How would you show process hierarchy in forest format?

   a. ps –ef tree

   b. ps –f

   c. ps –ef - - forest

   d. ps –ef -t

Answer C: - - forest is used to display process hierarchy in forest fprmat.

Quiz

2. What does the 2>&1 mean in this command? *find -name fred.txt > names 2>&1*

   a. Append standard error to same place as the standard output
   b. Append standard error to a file called &1
   c. Send standard error to a file called &1
   d. Send the output of the find command to /dev/null.

Answers

A

2. What does the 2>&1 mean in this command? *find -name fred.txt > names 2>&1*

   a. Append standard error to same place as the standard output

   b. Append standard error to a file called &1

   c. Send standard error to a file called &1

   d. Send the output of the find command to /dev/null.

Answer A: 2> is used to re-direct standard error. 1 is for standard output.

Quiz

3. Which of the following function checks in a string that all characters are in upper-case?

   a. isupper()

   b. Join()

   c. for(a-z)

   d. notlower()

Answers

A

3. Which of the following function checks in a string that all characters are in upper-case?

   a. isupper()

   b. Join()

   c. for(a-z)

   d. notlower()

Answer A: isupper() – Returns true if string has at least 1 character and all characters are in uppercase.

Summary

- In this module you should have learnt:
  - Process Management
  - Basic Linux Commands
  - System related commands
  - Basic shell scripting
  - Basic python and its usage

edureka!

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co