Command-Line Arguments to Provide Inputs for Variables  What are Command-Line Arguments ?  Command-Line Arguments are the arguments specified at the command prompt with command or script to be executed  Command-Line arguments are accessed with $1,$2,n  $0 represents the script name  Note:  Best practice for variables or command-line arguments is using curly braces like: {2}……${n}  Command-Line Arguments are also called as positional arguments  $# /${#} will give the number arguments passed to script  $\* & $@ or ${\*} & ${@} Represents all arguments (there is a difference between these two arguments but will see in our upcoming sessions)

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Insertion and Deletion of lines With sed Command ➢ We can insert/delete a line based on line number or search string

➢ Inserting New Line: ➢ Syntax: ➢ sed [options] ‘commands’ fileName

➢ sed ‘1i NewLineInfo’ fileName → Defaults to stdout

Inserts new data before first line ➢ sed –i ‘1i NewLineInfo’ fileName → Insert a New Line in file ➢ sed –i.bak 1i NewLineInfo’ fileName → Take backup and edit ➢ sed –i ‘1a NewLineInfo’ fileName → Inserts data after first line ➢ sed –i ‘$a NewLineInfo’ fileName → Inserts data after last line ➢ sed –i ‘/SearchString/i NewLineInfo’ fileName ➢ sed –i ‘/SearchString/a NewLineInfo’ fileName

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Insertion and Deletion of lines With sed Command

➢ Deleting Lines: ➢ Syntax: ➢ sed [options] ‘commands’ fileName ➢ ➢ ➢ ➢ ➢ ➢ ➢ ➢ ➢

sed ‘1d’ fileName sed –i ‘1d’ fileName sed –i.bak ‘1d’ fileName sed -i ‘5,7d’ fileName sed -i ‘4,d’ fileName sed -i ‘$!d’ filename sed -i ‘1,+3d’ fileName sed -i ‘1~3d’ fileName

→ Defaults to stdout

Deletes the first line → Deletes the line from a file → Take backup and delete → deletes the lines from 5th to 7th → deletes the lines from 4th to last line → delete the last line → delete all lines except last line → delete 1st and 3 lines after that → delete 1st and every 3rd line after that

➢ sed -i ‘/SearchString/d’ fileName → Delete the lines which are having root string

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

shift Command with Command-Line Arguments  Syntax:  shift N ,Where N is a +ve integer number only  How shift Command works ?  The shift command is used to move command line arguments N positions to the left. During this move, the first N arguments are lost.  If N=1: , Then the shift command moves command line arguments one position to the left. During this move, the first argument is lost.  Default Value for N is 1  Shift Command don’t disturb the ${0} / $0  its always a script name only

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

tr (translate) Command  tr command is useful to translate or delete the characters then writes the output on stdout  syntax:  Command Output | tr [options] [set1] [set2]  cat fileName | tr [options] [set1] [set2]  echo “info” | tr [options] [set1] [set2]  tr [a-z] [A-Z]  tr [A-Z] [a-z]     

tr ‘{}’ ‘()’ tr ‘ ‘ ‘’ tr ‘’ ‘ ‘ tr -d ‘set’ tr -s ‘ ‘

or tr [:lower:] [:upper:] or tr [:upper:] [:lower:]

lowercase to uppercase translation upper to lower translation

 translates {} as ()  translates space with newline  translates newline with space  Deletes the given set , example: cat /etc/passwd | tr –d ‘[:digit:]’  squeeze-repeats

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Command Chaining Operators  Command Chaining Operators are useful to combine multiple commands, so that we can write simple and short shell scripts  They are:  Pipe Operator |  Semicolon Operator ;  Logical AND Operator &&  Logical OR Operator ||  Logical AND OR Operator && ||

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Comments && Logging  Comments:  Comment is a human readable explanation in the script  Adding comments to your Bash scripts will save you a lot of time and effort when you look at your code in the future.  Let’s say you want to change a script that you wrote a few months or years ago. The chances are that you will not remember why you wrote some complicated piece of code unless you added a comment.  # This is single line comment  << COMMENT  This line won’t execute  This line also won’t execute  COMMENT

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Comments && Logging  Logging:  It is a way to store execution information about your script  Adding logging is a good practice  Note: Add logging for the scripts which you want to execute in the background

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Environment Setup To Practice With Bash Shell Scripting  We need any OS (Linux/Unix/Mac)  You can get those OS’s  Directly Install Required OS on Your PC/Laptop  Get it from any Cloud  Use WSL and Deploy Required OS on Windows  We will see how to Deploy CentOS on Windows WSL  And enabling systemctl on CentOS of WSL

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Thank you

Exit Status of a Command    

Each Linux Command return a status(integer value 0-255) when it is executed We can capture this return status using $? If ?) into a variable also  cmdStatus={?}  We use this $? Value in conditional statement(if, if else….)

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Here Document and Here String  Heredoc is a multi-lines of strings that will be redirected to a Command  Syntax:

 Mostly We use heredoc for cat and ssh commands

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Here-String  Syntax:  command <<< string  Note: this string value can be from a variable or from a command output

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

if elif elif else statement  if else:  Syntax:  if Command1 ; then Command2 Command3 else Command4 Command5 fi

 if elif elif elif …. else :  Syntax:  if Command1 ; then Command2 Command3 elif Command4 ; then Command5 Command6 elif Command7 ; then Command8 Command9 else Command10 Command11 fi

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Input’s & Output’s For Variables in Bash Shell Scripting  There are different ways to provide inputs for Variables in Bash Shell Scripts  Directly Defining Variable Value in Scripts, itself  Using Read Command  Using Source Concept  Using export command  Using Command line Arguments  Using Options with the help of while loop

 There are different ways to display/print Variable Values from bash shell scripts  echo  cat command with here-doc  printf

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Bash Arrays  Bash Array is also a variable but It is useful to store multiple values into one variable  Bash Arrays are also called as indexed Arrays or lists  Basics about Arrays:  Creating an Empty Array  declare -a myArray  myArray=()  Creating an Array with values  myArray=(1 2 3)  myArray=(1 “hi” “bash shell” 5 7 9.6)  Note:  Values are separated with space  Values may be of same type or different type  If your value is having space then keep your value inside quotes  So, Bash Array is a collection of Values/elements

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Automation with

Bash Shell Scripting (#!/bin/bash) VRTechnologies

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction  What is a Shell ?  The Shell is the Linux Command Line Interpreter, Shell executes the commands/scripts  There are different types of shells(interpreters) to execute commands/scripts in Unix/ Linux Environment  cat /etc/shells

 The most popular and advanced shell is “bash”

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction . . .  What is a Shell Script ?  Shell Script is a sequence of commands pasted in a text file.  Example: Versions info of java, git, Jenkins, nginx, Docker/Kubernets, Ansible etc…

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction . . .  Why we need to develop shell scripts ?  To Automate repetitive tasks in our Unix/Linux Environment.

 We do release/deployment/patching on DevOps Tools – Week Ends  So, just want to verify the all-DevOps Tools Versions info on every Monday

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction . . .  Examples:  Send some DevOps tools versions Report to compliance team @ Every Friday  Versions info of java, git, Jenkins, nginx, Docker/Kubernets, Ansible etc…  Find Log Files Between Two Dates  Monitoring Application(s) or Tool(s) or File System Usage  Monitor xyz Log Directory and if any logs are not generating in last 24hrs then trigger a mail  Monitor Micro- Services and Send Automatic Email Alerts When Application are Down  Monitor Jenkins and Send Automatic Email Alert When Application is Down  Monitoring File System Usage and Send Automatic Email Alert When Your File System Usage is more than threshold let say 90%

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction . . .

 Shell Scripting is a great way to Automate repetitive tasks in our Unix/ Linux Environment.

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Thank you

Bash Dictionary Variables  Bash Dictionaries are same as bash array variables but these are useful to store values based on custom keys instead of index values  These are also called as Associative arrays/Hash Tables  declare -A myDict  myDict=([name]=docker [version]=1.13.1 )  Or  myDict[name]=docker  myDict[version]=1.13.1  Retrieve Values:  All Value: echo “{myDict[name]}”  Get Keys:  echo “${!myDict[@]}”

 Delete a value based on its key: unset myDict[name]

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Functions  Function is a block of code  We can re-use this block of code any number of time in scripts  Synatx:  function Name\_of\_the\_function { command1 ; command2 ; ….. Command\_N; }  Or  function Name\_of\_the\_function { Command1 Command2 . . Command\_N }  Or  Name\_of\_the\_function() {command1 ; command2 ; ….. Command\_N; }  Or Name\_of\_the\_function() { Command1 Command2 . Command\_N }

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Functions…  Points to remember:  Function is a block of code with custom name  We can call/re-use this block with given custom name with any number of times  Function is a like a script / command  If it is a script then we can pass command line argument to a function  Syntax: functionName [arguments]  Arguments can be accessed using ${1} (functionName [arguments])  If it is a command then we can also find the exit status of a function using $?, return is the key send the exit status from a function  Function should be defined before its calling

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Loops  Example:  Just Assume that - Nginx is installed  Write a Shell Script to start the nginx if it is not running and monitor it for 2mins with intervals of 30 sec

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Loops  Loop is a statement which allows to re-run command or series of commands or any logic based on requirement  We have different types of loops  For  While  Until  Select

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Regex (OR) Regular Expressions  Regex or Regular Expression is a string pattern that represents more than one string or several sequence of characters  Regular Expressions are made up of two types of characters  They are:  Ordinary Characters or Literals  such as space, underscore(\_), A-Z, a-z, 0-9  Special Characters or Meta Characters  |, ^ , $, . , ? , + , \* , ( ) and [ ] and  
 There are two types of regular expressions  They are:  BRE – Basic Regular Expressions (vi, sed, grep)  ERE – Extended Regular Expressions (awk)  We can use extended regex in grep and sed just by passing -E option

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to Variables VRTechnologies

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

What is a variable & How to declare and define it ?    

Variables plays an important role in any programming or scripting language Variables are used to store data/value Variable data/value can be a number / character / string / path/ command output Defining Variables:  Examples:  p=1  q=2.5  r=y  s=bash  name=‘bash shell scritping’  myPath=‘/home/VRTech’  myCmdOut=$(whoami)

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Assigning One Variable Value to Another Variable  Let say a=4 then b=${a} means a value is assigned to b

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Displaying Variable Value  We can use echo, cat

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Thank you

Executing Commands on Remote Servers  Now a Days , we are using configuration management tools like ansible, cheff, puppet & salt to work with remote servers.    

We use ssh to work with remote servers There are two ways to execute commands on remote servers Note: Your servers may be configured with password-less or with password authentication First Way – Execute your commands by logging into remote server  ssh remoteServerUser@remoteServerHostName  Provide the password if servers are configured with password authentication  Then run your command and exit from remote server  This is not good for automation  Second Way – without logging into remove server:  ssh remoteServerUser@remoteServerHostName “command”  ssh remoteServerUser@remoteServerHostName “command1; command2”

 sshpass is used to provide password for ssh command

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Basic Operations on Strings

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Basic Operations on Strings  Finding the length of a string  Concatenation of strings  Converting a string into lower/upper  ${myStr^^}  Upper  ${myStr,,}  Lower  Alternative way is using tr command  Slicing of strings  ${strVar:startingIndex:Length}  Alternative way is using cut command  Replacing the part of the string  ${strVar/oldStr/NewStr}  Alternative way is using sed command

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Thank you

printf command  echo and printf both commands are useful to display output  printf is also useful to format the output  Syntax:  printf “format”

arguments

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Scheduling Jobs With at and crontab  at and crontab commands are used to schedule a job , nothing but running shell script automatically at a given time in the background  at:  To run job only once  crontab :  To run jobs with intervals based on requirement – basically more than once  Note: Use always Complete Paths for commands in script when we want to execute a script using at/crontab

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

at Command  Verify at command first  If not installed, then use :  yum install at –y  systemctl start atd  systemctl enable atd  Scheduling a job using:  echo “completePathOfTheScript” | at  Example:  echo “/root/automation/practice\_6.sh” | at 17:57  atq  To list the scheduled jobs  atrm  To Remove job  requiredTime:

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

test command  test command is used to validate/judge conditions  What is a condition ?  Example: -e file  Syntax:  test condition or [ condition ] or [[ condition ]]  [[ ]] is the latest and advanced usage of test command  test -e file  We can combine test commands with logical AND , OR operators using this concept  If file exist then test command return or exits with status as zero(true) else non-zero(false)  We have two types of operators to create conditions  File Test Operators  Comparison Operators

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

File Test Operators to create conditions       

test -f file test –d file test -e file test -s file test -r file test -w file test -x file

or [[ -f file ]]  To validate file or path is a file or not or [[ -d file ]]  To validate file or path is a directory or not or [[ -e file ]]  To validate file exist or not or [[ -s file ]]  To validate file size is greater than zero or not or [[ -r file ]]  To validate file is having read permissions for the current user or [[ -w file ]]  To validate file is having write permissions for the current user or [[ -x file ]]  To validate file is having execution permissions for the current user

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Comparison Operators to create conditions  String Comparison:  test -z string or [[ -z string ]]  To validate string is null or not  test -n string or [[ -n string]]  To validate string is not null or not  test str1 == str2 or [[ str1 == str2 ]]  To validate two strings are equal or not  test str1 = str2 or [[ str1 = str2 ]]  test str1 != str2 or [[ str1 != str2 ]]  To validate two strings are not equal or not  Integer Numbers Comparison:  test num1 –eq num2 or [[ num1 –eq num2 ]]  test num1 –ne num2 or [[ num1 –ne num2 ]]  test num1 –lt num2 or [[ num1 –lt num2 ]]  test num1 –le num2 or [[ num1 –le num2 ]]  test num1 –gt num2 or [[ num1 –gt num2 ]]  test num1 –ge num2 or [[ num1 –ge num2 ]]

 To validate two numbers or equal or not  To validate two numbers or not equal or not  To validate num1 is less than num2 or not  To validate num1 is less than or equal to num2  To validate num1 is greater than num2 or not  To validate num1 is greater than or equal to num2

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Case Statement  Case Statement is another form of if elif elif elif else statement  if elif elif elif …. else :  Syntax:  if Command1 ; then Command2 Command3 elif Command4 ; then Command5 Command6 elif Command7 ; then Command8 Command9 else Command10 Command11 fi

 Case Syntax:  case AnyValue in value1 ) Command1 Command2 ;; value2 ) Command3 Command4 ;; value3 ) Command5 Command6 ;; \*) Command7 Command8 esac

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Command Chaining Operator – Pipe (|)  Pipe Operator (|)  Syntax  Command1 | Command2  Command1 | Command2 | Command3  Note:  This operator is useful to pass output of one command as an input to next command  By Default, Only Success Output from first command will be passed as an input to second command  If you want to pass error also to second command then:  Command1 2>&1 | Command2

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

echo Command usage  Syntax :  echo [-neE] [Arguments]  Options:  -n do not append a newline or trailing newline is suppressed  -e enable interpretation of the following backslash escapes  -E explicitly suppress interpretation of backslash escapes  Notes:  Without Options & without Arguments echo just prints/displays an empty line  Arguments :  Only String(s)  Only Variable(s) (Variable can have any kind of data like: strings,chars,numbers,path,cmd output)  Wildcard \*  Command Output also  Combination of all these  Although not necessary, it is a good practice to enclose the arguments in double or single quotes  If we use single quotes Variables will not be expanded and wildcard wont work

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

For Loop  For Loop Syntax1: for variable in strings/numbers/files/any Command output which are separated with space/newline do Logic to repeat # a single line or multi-lines done  Note:  This loop is going to repeat or re-execute or iterate the logic with N number of times  Where N is : how many strings, numbers, files are passing to for loop

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

For Loop  For Loop Syntax2: for (( Initializer; Condition ; step )) do Logic to repeat – a single line or multi-lines done  Syntax2: one more form: for (( ; ; )) do Logic to repeat – a single line or multi-lines – # It will execute infinite number of times done

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Accessing Array Elements/Values  How to access Values from an Array ?  Its based on Index Values  ${myArray[0]}  First Element/Value  ${myArray[2]}  Second Element/Value  ${myArray[@]}  All Elements/Values  ${myArray[\*]}  All Elements/Values  ${myArray[@]:i:n}  Retrieving n values starting from index i

 Get Index Numbers of an Array:  ${!myArray[@]}  Array Index Numbers

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Operations on Arrays  Finding the Length of an Array  ${#myArray[@]}  Array Length  Overwriting Values/Elements:  myArray[2]=newValue  Overwriting 2nd index value  Appending Values/Elements  myArray+=(8)  Appending Value  myArray+=(34 56)  Appending Values  Delete or unset a value in an array  unset myArray[2]

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Prerequisites to Enroll this Course

  

Basic knowledge on Linux/Unix/Mac OS and Commands Basic Knowledge to use vi/vim command line editors You Should have Linux/Unix/Mac OS to practice  (We practice on CentOS, just to send mails )

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Redirection Operators & File Descriptors for STDIN, STDOUT and STDERR    

Redirection Operators: Some Linux Commands needs input and some command take default inputs and it results output/error By Default input is given with the keyboard and output/error is displaying on the terminal (default) Sometimes we may need to store output/error into a file or we will pass output as an input to another command  So,  How we can provide inputs for command if required ?  How we store output/error into a file ?  How to pass one command output as an input to another command ?  Solution is: using redirection operators  We have different types of redirection operators , based on input/output redirection  They are:  Input Redirection (< & <<<) [ < for files and <<< for strings/text ]  Output Redirection (> & >> ) [ > to create a file, >> to append a data to a file ]  Combining Redirection ( | ) [ By Default it consider only success output from first command and pass as an input to second command ]

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Redirection Operators & File Descriptors for STDIN, STDOUT and STDERR…  File Descriptors for STDIN, STDOUT and STDERR  A File Descriptor is an integer number to identify STDIN, STDOUT & STDERR  0 : STDIN ( 0 < )  1: STDOUT ( > or 1 > )  2: STDERR ( 2 > )

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Redirection Operators & File Descriptors for STDIN, STDOUT and STDERR  Redirection Operators:  Input Redirection (< & <<<) [ < for files and <<< for strings/text ]  Output Redirection (> & >> ) [ > to create a file, >> to append a data to a file ]  Combining Redirection ( | ) [ By Default it consider only success output from first command and pass as an input to second command ]

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Scheduling Jobs With crontab  Syntax: 

 Execute at 6pm daily  0 18 \* \* \* /path/to/script.sh

 Or  0 18 \* \* \* /bin/bash /path/to/script.sh

 Execute at 6am and 6pm  0 6,18 \* \* \* /path/to/script.sh

 https://crontab-generator.org/

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Scheduling Jobs With crontab       

Examples: \* \* \* \* \* /path/to/script.sh  Every Minute @hourly /path/to/script.sh @daily /path/to/script.sh @reboot /path/to/script.sh @weekly /path/to/script.sh @yearly /path/to/script.sh

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Basic String Operations on Paths  dirname :  It will delete any suffix beginning with the last / (slash) character from path and return the result  If the path has no slash in it, it will output a dot , means current directory  We can use dirname for multiple paths also  basename:  Strips directory information or Strips suffixes from file names

Learn How to Automate Common Tasks with Bash Shell Scripting

exit Command and Its Usage  exit Command is used to exit/terminates a shell script.  We use this exit command to terminates the shell script based on some condition  Syntax:  exit / exit 0  exit N (where N is Integer Number )

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Command Chaining Operator - Semicolon Operator (;)  Semicolon Operator ;  This operator is useful to write and execute multiple commands in one line  Syntax:  Command1 ; Command2 ; Command3 ;  Note: All Commands Executes Independently in sequence

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Providing Password for SSH with SSHPASS  sshpass is not a built-in or default command, so we have to check before using it  Install:  Redhad/CentOS: yum install sshpass –y  Debian/Ubuntu: apt-get install sshpass –y  There are different ways for sshpass to provide password for ssh  Run Help: sshpass –h  Provide Password directly in command itself:  sshpass -p ‘test123’ ssh -E /dev/null -o StrictHostKeyChecking=No ec2-user@54.224.154.78 “uname“  Export the password and use it:  export SSHPASS=‘test123’  sshpass -e ssh -E /dev/null -o StrictHostKeyChecking=No ec2-user@54.224.154.78”uname“  Store the password in a file and use it:  sshpass -f passwordfile ssh -E /dev/null -o StrictHostKeyChecking=No ec2-user@54.224.154.78 “uname“

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Executing Commands on Remote Servers  How to pass automatically the required password to ssh command ?  sshpass command  expect command

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Simple if & if – else Statements Syntaxes  if and if-else statements are another form of command chaining operators ( && and || )

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Simple if Statement  Syntax;  Command1 && Command2

 Command1 || Command2

 if command1 then Command2 fi

 if command1 ; then Command2 fi

 if ! command1 then Command2 fi

 if ! command1 ; then Command2 fi

 This is Called simple if Statement

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Simple if Statement…  Syntax;  Command1 && { Command2 ; Command 3 ; }  if command1 then Command2 Command3 fi  Command1 || { Command2 ; Command 3 ; }  if ! command1 then Command2 Command3 fi

 if command1 ; then Command2 Command3 fi

 if ! command1 ; then Command2 Command3 fi

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Simple if and if-else statements …  Syntax;  Command1 && Command2 || command3  if else statement  if command1  if command1 ; then then Command2 Command2 else else Command3 Command3 fi fi  Command1 && { Command2 ; Command3 ; } || { command4; Command5 ; }

 if command1 then Command2 Command3 else Command4 Command5 fi

 if command1 ; then Command2 Command3 else Command4 Command5 fi

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Types of Variables  Basically, there are three types:  User-defined-Variables  These are the variables defined by us while developing shell scripts  System-defined-Variables  System-defined-Variables are default variables  System-defined-Variables can get using set command  System-defined-Variables are defined in CAPITAL LETTERS  Note: Its better to avoid user-defined-variables in terms of CAPITAL LETEERS  Special Variables  Special variables are like @,$\*….

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Command Chaining Operators Logical AND and OR Operators (&& ||)  Logical AND Operator &&  Syntax:  Command1 && Command2  Command1 && Command2 && Command3  Note:  AND Operator (&&) executes the second command only if first command executes successfully  Logical OR Operator ||  Syntax:  Command1 || Command2  Command1 || Command2 || Command3  Note:  OR Operator (||) is completely Opposite to AND Operator  OR Operator executes the second command only if first command Fails

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Command Chaining Operators Logical AND and OR Operators (&& ||)  Logical AND - OR Operator && ||  Syntax:  Command1 && Command2 || Command3  Note:  If command1 is succeeded then command2 executes else command3 executes  Its like If else logic

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Gmail Setup On CentOS(7 or 9)  Two Steps:  Step1: Get Gmail Id and Gmail App Password  Step2: Configure Gmail Setup on CentOS

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Get Gmail Id and App Password  Gmail ID: xyz from xyz@gmail.com  Gmail App Password:  Open: Gmail  Open: Mange Your Google Account  Select Security  Goto How you sign in to Google  Enable 2-Step Verification  Open: Mange Your Google Account  Select Security  Goto How you sign in to Google  Select 2-Step Verification and Generate App Password

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Configure Gmail Setup On CentOS  Step1: Enable systemctl if you are using CentOS from WSL  Step2: Small Difference between CentOS 7 and 9  Step3: Follow the Steps from Document

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Debugging a Bash Shell Script  Debugging is the process of finding and fixing errors or bugs in the script  There are different types of  set -n or bash -n  set -u or bash -u  set -v or bash –v  set -x or bash -x

options  Syntax Check  To Know if any undefined variables are using  Verbose Mode  Display each step after substitution

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

How to provide input values for Array Variables ?  There are different ways to provide inputs for Array Variables in Bash Shell Scripts  Directly Defining Array Variable Values in Scripts, itself  Using Command line Arguments  Note:  oldIFS=${IFS}  IFS=‘delimiter’  myCmdOutArray =( {oldIFS}  Creating An Array from Command Output  Using read Command  Using readarray command  Using mapfile command

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Rules to define variables     

   



Variable Name should contain only a-z or A-Z, 0-9 and \_ characters. Variable Name Should not start with number Variable Name length should be less than or equal to 20 characters. Variable Names are case sensitive. Means x and X are different. Don’t Provide space on either sides of equal symbol while defining variables  Ex: x=4 is valid  x =4 or x = 4 or x= 4 are invalid No need to declare variable type, Automatically it will take care while executing commands or scripts. Use quotes for the data if data consist of spaces We can store the output of a command into a variable as follows:  anyVariable=$(command)  anyVariable=`command` We can assign one variable value/data into another using:  Name=“Shell Scripting”  NewName=$Name  NewName=${Name} Better to avoid CAPTIAL Letters For Variables as system variables are in CAPITALS, if you want to use check once before creating it

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

How to use SCP ?  Password-less Authentication Env:  scp -o StrictHostKeyChecking=No FileToTranster ec2-user@54.234.43.144:/tmp/.  Password Authentication Env:  sshpass -p ‘password’ scp -o StrictHostKeyChecking=No remoteServers.txt ec2-user@54.234.43.144:/tmp/.

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

awk Command  AWK command is useful to select specific column(s)/field(s) from a given file/text/string or from Command Output based on delimiter(s)  Note:  Default delimiter is space and tab and use -F option for any other delimiters  Delimiter could be any number of characters  Columns/Field are $1 $2 $3 …..  $0  Entire Row/Line and $NF is last field  Number of the record(Line Number) can get using NR  Syntax:  awk ‘{action}’ fileName  To get required field  awk -F ‘[delimiters]’ ‘{action}’ fileName  Note: action is with print, example: print or print requiredField  awk ‘/SearchString/’ fileName or awk ‘condition’ fileName  awk ‘/SearchString/ {action}’ fileName or awk ‘condition {action}’ fileName  CommandOutput | required awk command

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Simple Info about expect command  Install expect if not exist:  Redhad/CentOS: yum install expect –y  Debian/Ubuntu: apt-get install expect –y  Expect itself is a separate scripting  How to use expect with SSH & SCP Command ?

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Basics of grep Command ➢ grep is used to select the lines/rows based on required search string from a given file(s)/Command Output ➢ grep syntax: ➢ grep [options] ‘searchString’ file(s) ➢ commandOutput | grep [options] ‘searchString’ ➢ cat file | grep [options] ‘searchString’ ➢ echo “some string” | grep [options] ‘searchString’ ➢ Note: if no match found then exist status of a grep is non-zero ➢ Options: ➢ Basic Options: -i -w -v -n -c -r -l -h –o -A -B -C ➢ Note: ➢ grep [options] ‘^searchString’ file(s) : Search Must Occur at the beginning of the line ➢ grep [options] ‘searchString$’ file(s) : Search Must Occur at the end of the line ➢ Advanced Options: -e -f and –E

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Please have a look once and start using basic options

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Basic Steps To Develop/Write And Execute Bash Shell Script  Step1: Get Requirement & its required commands  Step2: Choose Shell Type ( in our case bash) and find the location of shell : which bash  Step3: Open file using vi/vim editors (ex: vi firstScript.sh) on Your Linux/Unix/Mac OS  Note: .sh extension is not mandatory  Step5: Write Shebang line ( #!/bin/bash) as a very first line in script  Step6: Place List of Commands in Sequence and save the file  Step7: Provide the execution permissions for the script using chmod (example: chmod u+x scriptName)  Step8: Run the script as follows  ./scriptName  Or  /completePathOfTheScript example: /home/devopsScripts/scriptName

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

break and continue statements for loops

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

break and continue statements  These two statements must be used only inside loops  break:  break is used to stop the loop ; we can use this with or without condition  Note: Be clear it will stop only loop not a shell script  Continue:  It will skip current iteration/execution and move to the next iteration

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

How to replace search string with a variable for grep command ?  grep syntax:  grep [options] ‘searchString’ file(s)  reqS=“xyz”  grep [options] “${reqS}” file(s)

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Read Command to provide inputs for Variables  Syntax :  read [options] Variable(s)  Options:  -p : Outputs the prompt string before reading user input  -r : Disables the execution of escaped characters  -s : Does not echo the user input on terminal while passing to script

 read [options] variable  Note: Default Variable is REPLY

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Read Command to provide input for multiple Variables  Syntax :  read [options] Variable(s)  Options:  -p : Outputs the prompt string before reading user input  -r : Disables the execution of escaped characters  -s : Does not echo the user input on terminal while passing to script  read [options] var1 var2 …  Note:  Variables must separate with space if more variables are used in one read command  The read command takes the user input and splits the strings into fields, assigning each new field to a Variable. If there are fewer variables than fields, read stores the remaining fields to the final Variables  Default IFS is space  read command with custom IFS  IFS=‘:’ read -rp “Enter inputs: “ var1 var2

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

While loop  Syntax1: while : do echo “this is infinite loop” done

 Syntax2: while [[ condition ]] do echo “this logic will execute as long as condition is true” done

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Introduction to sed Command  Sed Command Stands for Stream Editor, basically it edits the stream of data.  Sed Command Performs lot of operations/functions on a file or on command output  Reading/Displaying/Viewing the file content  Searching  Finding and Replace  Insertion or deletion  Note: sed Also Supports Regular Expressions like grep  Syntax:  sed [options] ‘commands’ fileName  commandOutput | sed [options] ‘commands’

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Displaying/Printing/Viewing the file Content with sed based on Line-Numbers and Search String  Syntax:  sed [options] ‘commands’ fileName

        

sed ’’ fileName sed -n ’’ fileName sed -n ‘2p’ fileName sed -n ‘5,7p’ fileName sed -n ‘4,p’ fileName sed -n ‘$!p’ filename sed -n ‘1,+3p’ fileName sed -n ‘1~3p’ fileName

        

display File Content stop Default Display print 2nd line print 5th to 7th lines print 4th to last line print last line print except last line print 1st and 3 lines after that print 1st and every 3rd line after that

 sed -n ‘/root/p’ /etc/passwd

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Source Command to Provide Inputs for Variables  We define inputs into a file, then we use them in scripts using source command  Syntax :  source fileName  How Source Command Works ?  Source Command reads and executes the file content in the current shell.  Note: When we use this concept?  Common Variables across environment  Common Variables based on environment  Common Variables for multiple-scripts

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

While loop  Read Command Output/file content line by line  Syntax3: while read [-r] line : do echo “${line}” done < <(command)  Or

command | while read [-r] line do echo “${line}” done

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

While loop  Read File Content line by line  Syntax3: while read [-r] line : do echo “${line}” done < fileName

Note: while read or while IFS=“:”

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

export Command to Provide Inputs for Variables  export command exports the variables from a shell, making them global and available in each sub-shell  Syntax: export VarName=VarValue   Note: Variables will get vanish once you close the terminal

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Find and Replace With sed Command  Syntax:  sed [options] ‘commands’ fileName  sed ‘s/SearchString/ReplaceString/g’ fileName  Defaults to stdout  sed –i ‘s/SearchString/ReplaceString/g’ fileName  Edits the original file  sed –i.bak ‘s/SearchString/ReplaceString/g’ fileName  Take backup and edit

 Note: we can take any delimiter in find and replace of sed command but make sure that it’s a single character  Example: take | in place of /  sed ‘s|SearchString|ReplaceString|g’ fileName

Learn How to Automate Repetitive Tasks with Bash Shell Scripting

Until loop  Syntax: until [[ condition ]] do echo “this logic will execute as long as condition is false” done

Learn How to Automate Repetitive Tasks with Bash Shell Scripting