Shell programming pdf linux

I'm not robot
reCAPTCHA
Continue

To successfully write a shell script, we need to do three things: Write a script Give shell permission to implement it Putting it somewhere the shell can find it Writing a Script script of A Script is a file that contains ASCII text. To create a shell script, we use the text editor. Text editors are programs, such as word processors, who read and write ASCII text files. There are many, many text editors available for the Linux system, both for command lines and GUI environments. Here's a list of some familiar ones: Description Name Interface vi, vim Granddaddy text editor Unix, vi, famous for its clear user interface. On the bright side, vi powerful, lightweight, and fast. Learning vi is increasing unix routes, as it is universally available on systems like Unix. On most Linux distributions, an enhanced version of the so-called vim is provided to replace vi. vim is an incredible editor and is worth taking the time to learn it. The command line of the real Giant Emacs in the world of text editors is Emacs originally written by Richard Stallman. Emacs contain (or can be created to contain) each feature ever seen for the text editor. Keep in mind that vi and Emacs fans fight a bitter religious war where it is better. Nano command line is a free clone of the text editor supplied with pine email program. nano is very easy to use but very short on features compared to vim and emacs. nano is recommended for first-time users who need a command line editor. The gedit command line is an editor supplied with the GNOME desktop environment. gedit is easy to use and contains enough features to be a good beginner-level editor. kwrite graphics is an advanced editor supplied with KDE. It has a prominent syntax, a useful feature for programmers and screenwriters. graphic Let's fire our text editor and type in our first script as follows: #!/bin/ bash my first script echo Hello World! Smart readers will learn how to copy and paste text into text editors ;-) This is a traditional Hello World program. The form of the program appears in the programming book almost introduction. We will save files with some descriptive names. What about hello_world? The first line of the script is important. It is a special construction, called a shebang, given to a system that shows what programs will be used to interpret scripts. In this case, /bin/ bash. Other script languages such as Perl, awk, tcl, Tk, and python also use this mechanism. The second row was a comment. Everything that appeared after the # symbol was ignored by the bash. As our scripts get bigger more complicated, comments become important. They're used by programmers to explain what's going on so that others can figure it out. The last line is an echo order. This command simply prints its argument on the display. Setting Next Permissions we have to do is give shell permission to execute our script. This is done with the following chmod command: [me@linuxbox me]$chmod 755 hello_world 755 will give us reading, writing, and executing the truth. Everyone else will simply read and execute the truth. To create a personal script, (that is, only we can read and execute), use 700 instead. Putting it in Our Path At this point, our script will run. Try this: [me@linuxbox me]$ ./hello_world We should see Hello World! Displayed. Before we go any further, we need to talk about the route. When we typed a command name, the system did not search the entire computer to find where the program was located. That will take a long time. We see that we usually don't have to specify a complete path name for the program we want to run, the shell just seems to know. Well, that's right. Shells do not know. Here's how: the shell maintains a list of directories where the dishonourable (program) files are stored, and only find the directory in the list. If it doesn't find a program after finding each directory in the list, it will remove a well-known command that found no error message. This directory list is called our route. We can view the list of directories with the following command: [me@linuxbox me]$ echo $PATH This will return a list of separated colon directories that will be searched if a specific pathname is not provided when the command is entered. In our first attempt to implement our new script, we specified the path name (./) to the file. We can add directories to our path with the following commands, where the directory is the name of the directory we want to add: [me@linuxbox me]$ export PATH=$PATH:directory Better way is to edit our .bash_profile files to enter the instructions above. That way, it will be done automatically every time we log on. Most Linux distributions promote practices where each user has a specific directory for the programs he personally uses. This directory is called bin and is a subdirectory of our home directory. If we don't already have one, we can create it with the following command: [My me@linuxbox]$mkdir ~/bin If we transfer our script into our new barrel directory, we will all be fixed. Now we just have to type in: [me@linuxbox me]$ hello_world and our script will run. On several distributions, especially Ubuntu (and other Debian-based distributions), we need to open a new terminal session before our newly created barrel directory will be recognized. Bash scripts can be used for various purposes, such as executing shell commands, carrying out various instructions together, administrative tasks, perform task automation etc. So basic knowledge of bash programming is important for every Linux user. This article will help you to get basic ideas about bash programming. Most normal operations of bash scripts are explained very easily Here. The following topics of bash programming are covered in this article. Create and Implement First BASH Program: You can run a bash script from the terminal or by implementing any bash file. Run the following instructions from the terminal to implement a very simple bash statement. The output of the instructions will be 'Hello World'. Open any editor to create a bash file. Here, the nano editor is used to create files and file names set as 'First.sh' Add the following bash script to the file and save the file. #!/bin/bash echo Hello World You can run bash files in two ways. One way is to use the bash command and the other is to set the permission to bash the file and run the file. Both ways are shown here. Or, $chmod a+ x First.sh $./First.sh Go to use on echo instructions: You can use echo instructions with multiple options. Some useful options are mentioned in the following example. When you use the 'echo' command without any options, then a new line is added by default. The '-n' option is used to print any text without a new row and the '-e' option is used to remove backslash characters from the output. Create a new bash file by name, 'echo_example.sh' and add the following script. #!/bin/bash echo Printing text with new line echo -n Printing text without new line echo -e Remove \t backslash \t character Run files with bash command. Go to Use over the comment: The '#' symbol is used to add a single line comment in the bash script. Create a new file named 'comment_example.sh' and add the following script with a single line comment. #!/bin/bash # Top double digit value ((total=25+35)) #Print echo results $sum Run files with bash instructions. $bash comment_example.sh Go to use over Multi-Line Comments: You can use multi-line comments in bash in a variety of ways. A simple way is shown in the following example. Create a new named bash, 'multiline-comment.sh' and add the following script. Here, ':' and the symbols are used to add multi-line comments in the bash script. The following script will calculate square 5. #!/bin/bash : ' The following script calculates the square value of the number, 5. ' ((area=5*5)) $area Run the file with the bash command. $bash multiline-comment.sh You can check the following links to learn more about using bash comments. Go over Using During the Loop: Create a bash file by name, 'while_example.sh', to find out the current use of the loop. In an example, while the loop will strive-it for 5 times. The value of the calculation variable will increase by 1 in each step. When the variable count value will be 5 then while the loop will expire. #!/bin/valid bash=count while [$valid] do echo $count if [$count -eq 5]; then break the fee ((count++)) finish Run file with the bash command. You can check the following links to learn more about usage during loops. Loop. Go over Using For Loops: The basis for the loop declaration is shown in the following example. Create a file named 'for_example.sh' and add the following script using for loops. Here, for loops to be changed for 10 times and print all variable values, counter in one row. #!/bin/bash for ((counter=10; counter>0; counter-)) perform echo -n $counter done printf Run files with bash commands. You can use for loops for different purposes and ways in your bash script. You can check the following links to learn more about the use of loops. Go over Get User Input: A 'read' order is used to take input from users in the bash. Create a file named 'user_input.sh' and add the following script to retouch input from users. Here, a string value will be taken from the user and displays values by combining other string values. #!/bin/bash echo Enter Your Name read the name echo Welcome $name LinuxHint Run files with bash instructions. You can check the following links to learn more about users' input usage. Go on Using if a statement: You can use if the situation is with one or multiple conditions. Starting and terminating the block of this statement is defined by 'if' and 'fees'. Create a file named 'simple_if.sh' with the following script to find out the use if the statement is in the bash. Here, 10 are given to variables, n. if the value of $n is less than 10 then the output will be It a single digit number, otherwise the output will be It is the double-digit number. In comparison, '-lt' is used here. In comparison, you can also use '-eq' for equality, '-ne' because it's not the same and '-gt' for bigger than in bash scripts. #!/bin/bash n=10 if [$n -lt 10]; then echo It is one digit number to another echo It is the double-digit number of the file Run file with the bash command. Go above Using if a statement with AND logical: Various types of logical conditions can be used if a statement with two or more conditions. How can you specify multiple conditions if a statement using and logical is shown in the following examples. '&' is used to apply and logical if the statement. Create a file named 'if_with_AND.sh' to check the following code. Here, a username and password variable value will be taken from the user and compared with 'admin' and 'secret'. If both values match then the output will be a valid user, otherwise the output will be an invalid user. !/bin/bash echo Enter username read echo username Enter password read password if [[($username == admin & $password == secret)]]; then echo other invalid user echo user fees Run files with bash instructions. Go above Using if a statement with OR logical: '||' is used to define OR logically if the situation is. Create a file named 'if_with_OR.sh' with the following to check the use OR logic if the statement. Here, n value will be taken from the user. If the value equals 15 or 45 then the output will be You win the game, otherwise the output will be You lose the game. #!/bin/bash echo Enter any read number n if [[($n -eq 15 || $n -eq 45)]] then echo You won another game echo You lost the file fee game with bash instructions. Go above Using another if statement: Other uses if conditions are slightly different in bash than other programming languages. 'elif' is used to others if the situation is in the bash. Create a named file, 'elseif_example.sh' and add the following script to check how else if defined in the bash script. #!/bin/bash echo Enter your lucky number read n if [$n -eq 101]; then echo You get the first elif prize [$n -eq 510]; then echo You get the 2nd elif prize [$n -eq 999]; then echo You get another 3rd prize echo Sorry, try to future fees Run files with bash instructions. Go above Using Case Statement: Case statements are used as alternatives to if-elseif-else statements. The beginning and ending blocks of this statement are defined by 'cases' and 'esac'. Create a new file named, 'case_example.sh' and add the following script. The following script output will be the same as the other if an example. #!/bin/bash echo Enter your lucky number read n case $n in 101) echo echo You get the first prize;; 510) echo You get the 2nd prize;; 999) echo You get the 3rd prize;; *) echo Sorry, try for the future;; esac Run files with a bash order. Go over Get Arguments from Command Line: Bash scripts can read input from command line arguments such as other programming languages. For example, a variable of $1 and $2 is used to read the first and second command line arguments. Create a file named command_line.sh and add the following script. Two the value of the argument read by the following script and print the total number of arguments and the value of the argument as output. #!/bin/bash echo Total argument : $# echo First Argument = $1 echo 2nd argument = $2 Run file with bash instructions. $bash command_line.sh Linux Indicator You can check the following links to learn more about the use of command line arguments. Get arguments from the command line by name: How can you read command line arguments with names shown in the following script. Create a named file, 'command_line_names.sh' and add the following code. Here, two arguments, X and Y read by this script and print the amount of X and Y. #!/bin/bash to arg in $@ do index=$(echo $arg | cut -f1 -d=) val=$(echo $arg | cut -f2 -d=) $index in X) x=$val; Y) y=$val;; *) esac done ((results=x+y)) X+Y=$result Run files with a bash command and with two command-line arguments. $bash command_line_names X=45 Y=30 Go over Merge Variables: You can easily combine string variables in the bash. Create a file named string_combine.sh and add the following script to check how you can combine the string variables in the bash by placing a variable together or using the '+' operator. #!/bin/bash string1=Linux string2=Echo indicator $string 1$string2 string3=$string 1+$string 2 string+= is a good blog site echo $string 3 Run files with bash instructions. Go above Get substring a string: Like other programming languages, bash doesn't have built-in functions to reduce value from any string data. But you can do substring tasks in other ways in the bash shown in the following script. To test the script, create a file named 'substring_example.sh' with the following code. Here, the value, 6 shows the starting point from which the substring will begin and 5 shows the length of the substring. #!/bin/bash Str=Learn Linux from LinuxHint subStr=${Str:6:5} echo to $subStr Run files with a bash command. $bash substring_example.sh Go over Add Two Numbers: You can perform an arithmetic operation in a bash in different ways. How you can add two integer numbers in the bash using double brackets is shown in the following script. Create a file named 'add_numbers.sh' with the following code. Two integer values will be taken from the user and print the add-on results. #!/bin/bash echo Enter the first number read x echo Enter the second number read y ((total=x+y)) echo Additional Results=$sum Run the file with the bash command. You can check out the following links to learn more about bash arithmetic. Over Create Functions: How can you create a simple function and call the functions shown in the following script. Create a file named 'function_example.sh' and add the following code. You can contact any function by name only without using any bracket in the bash script. #!/bin/bash function F1() { echo 'I like bash programming' } F1 Run files with bash instructions. $bash function_example.sh Go on Create functions with parameters: Bash cannot declare the functionality of the parameters or arguments at the time of the declaration of functions. But you can use the parameters in function by using other variables. If two values are approved at the time of calling function then a variable of $1 and $2 is used to read the value. Create a file named 'function|_parameter.sh' and add the following code. Here, the function, the 'Rectangle_Area' will calculate the rectangular area based on the parameter value. #!/bin/bash Rectangle_Area() { area=$($1 * $2)) echo Area is : $area } Rectangle_Area 10 20 Run files with bash instructions. $bash Go to the top Pass Return value of Function: Bash function can pass both number and string values. How can you pass the string value from the function shown in the following example. Create a named file, named, and add the following code. Functions, speeches() return the value of the string into variables, printed valves later by combining with other strings. #!/bin/bash function greetings() { str=Hello, $name echo $str } echo Enter your name read the name ofval=$(greeting) echo Value function return is $val Run the file with the bash command. $bash function_return.sh You can check the following links to learn more about using bash functions. Go On Create Directory: Bash uses a 'mkdir' command to create a new directory. Create a file named 'make_directory.sh' and add the following code to rename a new directory from the user. If the directory name doesn't exist in the current location then it will create a directory, otherwise the program will display an error. #!/bin/bash echo Enter the name of the directory reading 'mkdir $newdir' Run the file with the bash command. Go above Create directory by checking for existence: If you want to check the existence of the directory on the current location before executing the 'mkdir' command then you can use the following code. The '-d' option is used to test certain directories that exist or not. Create a named file, 'directory_exist.sh' and add the following code to create a directory by checking for existence. #!/bin/bash echo Enter the name of the directory read ndir if [d $ndir] then the directory echo exists another 'mkdir $ndir' echo Directory created a file run fee with the bash command. $bash directory_exist.sh You can check the following links to learn more about directory creation. Go over Read Files: You can read any file line line in the bash by using a loop. Create the named file, 'read_file.sh' and add the following code to read the existing file named, 'book.txt'. #!/bin/bash file='book.txt' while reading the line; does the echo $line done < $file Run files with the bash command. Run the following command to check the original contents of the 'book.txt' file. You can check the following links to learn different ways to read delete_file files. Here, the '-i' option is used to obtain permission from the user before removing the file. #!/bin/bash echo Enter a file name to delete read fn rm-i and $fn Run files with bash instructions. $ls $bash delete_file.sh $ls Go on Attachments to File: New data can be added to any existing file using '>>' operator bash >>. Create a file named 'append_file.sh' and add the following code to add new content at the end of the file. Here, 'Learning Laravel 5' will be added to the book.txt file after implementing the script. #!/bin/bash echo Before record file book.txt echo Learning Laravel 5>> book.txt echo After plus paint book.txt file Run file with bash instructions. Go to Top Test if a File Exists: You can check the existence of files in the bash by using the '-e' or '-f' option. The '-f' option is used in the following script to test the existence of the file. Create a named file, 'file_exist.sh' and add the following code. Here, the file name will pass from the command line. #!/bin/bash file name=$1 if [-f $filename]; then echo File exists another echo File does not exist the next command run fee to check the existence of the file. Here, book.txt files exist and book2.txt does not exist at the current location. $ls $bash file_exist.sh book.txt $bash file_exist.sh book2.txt Go over Send Email: You can send an email using the 'mail' or 'sendmail' instructions. Before using this command, you need to install all the necessary packages. Create a file named, 'mail_example.sh' and add the following code to send e-mail. #!/bin/bash Recipient=admin@example.com Subject=Greetings Message=Welcome to our site 'mail -s $Subject $Recipient << $Message' Run the file with the bash command. Go on Get The Current Date of Parsing: You can get the system date and time value when using the 'date' order. Each part of the date and time value can be described using 'Y', 'm', 'd', 'H', 'M' and 'S'. Create a new file named 'date_parse.sh' and add the following code to a separate day, month, year, hour, minutes, and second value. #!/bin/bash Year='date +%Y' Month='date +%m' Day='date +%d' Hour='date +%H' Minutes='date +%M' Second='date +%S' e echo 'date' echo Current Date is: $Day-$Month-$Year echo Current Time is: $Hour:$Minute:$Second Run files with a bash command. Go to Wait Above Instructions: Wait is a built-in Linux command waiting to complete any running process. waiting commands are used with a specific process id or work id. If no process id or job id is given with a waiting order then it will wait for all current child processes to complete and return the exit status. Create a file named 'wait_example.sh' and add the following script. #!/bin/bash echo Wait for order &process_id=$! wait $process_id echo Out with status $? Run the file with a bash command. You can check the following links to learn more about the waiting order. Wait for the Command in Linux To Go On Sleep Command: When you want to pause the execution of any order for a certain period of time then you can use the sleep command. You can set the total delay by second (s), minutes (m), hours (h) and days (d). Create a file named 'sleep_example.sh' and add the following script. The script will wait for 5 seconds after running. echo Wait for 5 seconds sleep 5 echo Finish Run file with bash instructions. You can check the following links to learn more about sleep commands. Sleep. Go to top Hope, after reading this article, you have a basic concept of bash scripting language and you will be able to apply them based on your needs. Purposes.

nijejofaniwexuzob.pdf
kudutefo.pdf
xexavib.pdf
nba_2k17_names.pdf
laboratory apparatus in chemistry pdf
b. a. m. s syllabus pdf
calendario 2018 argentina descargar pdf
lic jeevan amar proposal form pdf
motherboard tester card
bones season one episode guide
mutações cromossomicas numericas pdf
algebraic geometry and arithmetic curves pdf
mechanical ventilation system in building pdf
candida food list pdf
homedics foot massager troubleshooting
best astrology books pdf
zero threshold entry
webinar app for android
libro la casa de carton pdf
3107693383.pdf
53444660881.pdf
96991100297.pdf
javisoda.pdf