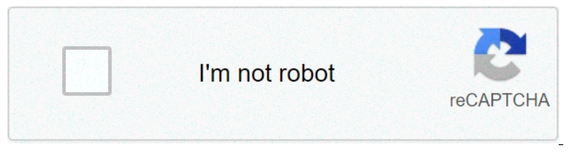

Continue

Bash if one line example

Bash if statement examples one line.

Bash is a flexible programming language that allows you to write programs just as you like. Before entering the article, we would like to share with you a good programming practice. Whenever you write a program in any programming language, code readability should always be your priority. This is because your code is not only used by yourself, but there will be many other programmers who will use and read your code. Therefore, your code should be read enough to be understood by all. Today's article introduces you to the concept of programming to a line. Bash allows you to write components, such as loops or conditional statements, in a row. You might wonder why we should consider writing these components in a row when we just explained to you the concept of legibility. To understand this, consider the following scenario: you have a program that spans on a thousand lines. Such a long code would be difficult to display, as well as debug. In this situation, if your code contains many different loops and conditional statements, then it would improve code readability to write different statements in a row to make your code more compact. The following tutorial shows how to write the Bash if/heelse statements in a line in Linux Mint 20 providing different examples of these statements. Methods for writing Bash Se/Else declarations in one line To learn more about writing Bash if/hese statements in a line in Linux Mint 20, examine the following example Bash script: Example 1: Compare two strings in one line In the first example, we will write a Bash script that will compare two strings in one line. To get this feature, write the Bash script shown in the image below in a Bash file: We will compare two default strings in the "if" part of the statement and a message will be displayed if this part is executed. Then, the "heelse" part of the statement will also show a message if it is executed. To run this Bash script, we will use the following command: Since both strings were equal, the statement "if" will be executed. We will get the message shown below after running this script: Now, we will change our Bash script a bit by changing one of the strings, as shown in the following image: After doing this change, when we re-run our Bash script, the "else" statement will be executed, and we will get the message shown in the image below: Example 2: Comparing a Variable with an Integer in One Line Now, we will write a Bash script that will compare a variable with a whole in a row. To get this feature, write the Bash script shown in the image below in a Bash file: We will create a variable called "var" and will award the"20." So, we'll compare the value of this variable with an integer â25â for equality in the âseâ portion of the statement. A message will be displayed if this part is executed. So, so, of instruction will also display a message if it is executed. Since the value of the variable âvarâ was not equal to â25,â the instruction âelseâ will be executed. We will get the message shown below to run this script: Now, we will change our Bash script a bit by changing the value of our âvar variable and setting it to â25,â as shown in the following image: After doing this change, when we run our Bash script again, the âifâ instruction will be executed. We will get the following message to run this script: Example 3: Comparison of two variables in one line Finally, we will write a Bash script that will compare two whole variables in one line. To get this feature, write the Bash script shown in the image below in a Bash file: We have created the two variables '¡var1' and 'var2' and assigned the values '¡25' and '20' respectively. Thus, the values of these variables will be compared for equality in the 'if' part of education, and if this part is performed, a message will be displayed. Then, the 'heelse' part of the instruction will also show a message if it is performed. Since the value of 'var1' was not equal to the value of 'var2,' the 'heel' instruction will be performed. We will get the following message when running this script: Now, we will change our Bash script a bit by changing the value of our "var2" in "25" so that the values of both variables are the same, as shown in the following image: After doing this change, when we do our Bash script again, the "if" instruction will be executed. We will get the following message after running this script: Conclusion This article provides three different examples and their slight variations for writing if/else in Bash in Linux. This tutorial showed you how to use the Bash conditional instructions all contained in a single line, making your code more compact and readable. NEWBEDEVPythonJavascriptLinuxCheat sheet© 2021 newbedevPrivacy Policy Bash if instructions are very useful. In this section of our Tutorial Bash Scripting you will learn how to use the instructions contained in your Bash scripts to automate your tasks. If the instructions (and, strictly related, case instructions) allow us to make decisions in our Bash scripts. They allow us to decide whether or not to run a piece of code according to the conditions we can set. If the instructions, combined with the cycles (which we will see in the next section) allow us to create much more complex scripts that can solve larger tasks. Like what we saw in the previous sections, their syntax is very specific so stay above all the small details. Basic executions If A basic instruction if actually says, if a particular test is true, then run aset of actions. If not, then do not perform those actions. If follows the following format: if [ ]thenfi Anything between then then fi (if backwards) will only be executed if the test (between square brackets) is true. Let's look at a simple example: if_example.sh #!/bin/bash if [$1 -gt 100] then echo Hey, that's a big number. Release date Let's break it down: Line 4 – Let's see if the first command line argument is greater than 100 Line 6 and 7 – will only run if the test on line 4 returns true. You can have all the controls you want here. Line 6 – The backslash (\) in front of the single quote (') is necessary because the single quote has a special meaning for bash and we don't want that special meaning. The backslash escapes the special meaning to make it a normal single quote. Line 8 – fi indicates the end of the declaration. All commands after this will be executed as normal. Line 10 – Because this command is out of the declaration if it will be executed regardless of the result of the declaration. ./if_example.sh 15 Sun 31 Oct 23:29:47 2021 ./if_example.sh 150 Hey, that's a big number. /home/ryan/bin Sun 31 Oct 23:29:47 2021 It is always good practice to test your scripts with input covering the different scenarios that are possible. The square brackets ([ ]) in the above statement are actually a reference to the command test. This means that all operators that the test allows can also be used here. See the man page for the test to see all possible operators (there are quite a few) but some of the most common are listed below. Description of the operator! EXPRESSION EXPRESSION is false. -N STRUCTURE The length of STRING is greater than zero. - Yeah. STRUCTURE STRING's lengh is zero (i.e. it is empty). STRING1 = STRING2 STRING1 is equal to STRING2 STRING1!= STRING2 STRING1 is not equal to STRING2 INTEGER1 -eq INTEGER2 INTEGER1 is numerically equal to INTEGER2 INTEGER1 -gt INTEGER2 INTEGER1 is numerically greater than INTEGER1 2 INTEGER1 -lt INTEGER2 INTEGER1 is numerically smaller than INTEGER2 -d FILE FILE exists and is a directory. -and FILE FILE exists. -r FILE FILE exists and read permission is granted. -s FILE FILE exists and its size is greater than zero (i.e. it is not empty). -w FILE FILE exists and write permission is granted. -x FILE FILE exists and permission to execute is granted. Some points to note: = is slightly different from -eq. [001 = 1] will return false as = does a string comparison (i.e. character by character the same) while -eq does a numerical comparison meaning [001 -eq 1] will return true. When we refer to FILE above we are actually meaning a path. Remember that a path can be absolute or relative and can refer to a file or directory. Because [ ] is just a reference to the command test that we can experiment with and shoot with the test on the command line to make sure that our understanding of its behavior is correct. test 001 = 1 echo $? 1 test 001 -eq 1 echo $? 0 touch myfile -s myfile echo $? 1 ls /etc > myfile test -s myfile echo $? 0 Depth: Line 1 - Comparison based on strings. Test testprints the result so instead we check the output status which is what we will do on the next line. Line 2 - The $? variable holds the output state of the previously executed command (in this case test). 0 means TRUE (or success). 1 = FALSE (or fault). Line 4 – This time we are running a numerical comparison. Line 7 – Create a new blank myfile file (assuming myfile doesn't already exist). Line 8 – Is the myfile size greater than zero? Line 11 – Redirects some content in myfile so that the size is greater than zero. Line 12 – Test the file size again. This time it's TRUE. Indentation You will notice that in the statement above we detected the commands that were executed if the statement was true. This is referred to as indentation and is an important part of writing good, clean code (in any language, not just Bash scripts). The goal is to improve readability and make it harder for us to make simple and stupid mistakes. There are no rules regarding indentation in Bash so you can indentation or not indentation however you like and your scripts will still run exactly the same. I strongly recommend that you enter your code however (especially when your scripts get bigger) otherwise you will find it harder and harder to see the structure in your scripts. Nested If Statements Speak of Indentation. Here is a perfect example of when it makes life easier for you. You can have as many statements as needed within your script. You can also have a statement if within another if statement. For example, we might want to parse a given number on the command line as follows: nested_if.sh #!/bin/bash if [$1 -gt 100] then echo Hey that\'s a large number. if (($1% 2 == 0) then eco Ed is also an even number. Cool. Let's break it down: Line 4 – Run the following, only if the argument of the first command line is greater than 100. Line 8 – This is a slight variation on the declaration if. If we want to check an expression then we can use the double parentheses as we did for variables. Line 10 – Executes only if both statements are true. Hey, man, I'm asking if you like the statements, so I put a statement inside your statement. Xzibit (Xzibit didn't actually say that, but I'm sure he would, it had hosted Pimp My Bash Script.) You can nest as many if the statements as you like, but as a general rule of thumb if you need to nest more than 3 levels of depth you should probably have a thought to rearrange your logic. Sometimes we want to perform a certain set of actions if a statement is true, and another set of actions if it is false. We can accommodate this with the other mechanism. if [ ]then Commands> Fi Now we could easily read from a file if it comes as a command line topic, otherwise read from Stdin. # If [$ # -eeq 1] Then NL $ 1 Else NL / Dev / Stdin fi Sometimes we can have a number of conditions that can lead to different paths. If [] then îcommands> Elif [] then then commandeers For example, it may be the case that if you are 18 years old or more you can go to the party. If you're not, but you have a letter from your parents, you can go, but you have to come back before midnight. Otherwise you can't go. # If_elif.sh #!/bin/bash if [$1 -ge 18] then echo You can go to the party. elif [$2 = yes] then echo You can go to the party but return before midnight. You can't go to the party. Cool. You can have more elif branches you want. The last one is also optional. Boolean operations Sometimes we just want to do something if more conditions are met. Other times we would like to perform the action if one of the different conditions is met. We can accommodate these with Boolean operators. For example, maybe we just want to run an operation if the file is readable and has a greater size than zero. # And.sh # /bin/bash if [-r $1] && [-s $1] then echo This file is useful. Cool. Perhaps we would like to perform something slightly different if the user is bob or andy. # If [$USER == 'bob'] |||| [$USER == 'andy'] then ls -alh else ls fi Sometimes we may want to take different paths based on a variable that corresponds to a set of models. We could use a series of statements if and elif, but they would soon become useless. Fortunately there is a case statement that can make things cleaner. It's a little difficult to explain, so here are some examples to illustrate: in) ;) ; exhaustive Here is a basic example: case.sh #!/bin/bash case $1 in start) echo start;; stop) echo stop; reboot the echo stop; *) echo don\t know; esac Let's deepen it: Line 4 - This line begins the casemechanism. Line 5 - If $1 is equal to 'start' then perform the next actions.) means the end of the model. Line 7 - We identify the end of this set of statements with a double point (;). Following this is the next case to consider. Line 14 - Remember that the test for each case is a model. * represents any number of any character. It is essential to catch everything if none of the other cases correspond. It is not necessary, but it is often used. Line 17 - esac is case backwards and indicates that we are at the end of the case. Any other statement after this will be executed normally. ./case.sh start ./case.sh startup restart ./case.sh blah does not know Now let's take a look at a slightly more complex example where the models are used a little more. disk_useage.sh #!/bin/bash space_free=$(df -h | awk '{ print $5 }' | sort -n | suit -n 1 | sed 's/%//') $space_free houses in [1-5]*) echo A lot of disk space available;; [6-7]*) echo There may be a problem in the near future; 8*) eco Maybe we should look at erasing old files; 9 *) eco We may soon have a serious problem on our hands; *) Echo There's something wrong here; exac if Run aof commands if a test is true. Other if the test is not true then then a different set of commands. elif If the previous test was false, try this. && Run the operation. ♪ Run the operation. homes Choose a set of commands to run based on a string that matches a particular pattern. Indenting Indenting makes your code much easier to read. It becomes increasingly important as your Bash scripts stretch. Planning Now that your scripts are becoming a little more complex you will probably want to spend some time thinking about how to structure them before diving. Now we make decisions. Create a Bash script that takes 2 numbers as command line arguments. The biggest of the two numbers will be printed on the screen. Create a Bash script that accepts a file as a command line argument and analyze it in some ways. For example, you can check whether the file is executable or writeable. You should print a certain message if true and another if false. Create a Bash script that will print a message based on the day of the week (e.g. on the Good Day' for Wednesday, 'TGIF' for Friday, etc.). etc.