

☐

I'm not robot


reCAPTCHA

Continue

If else example in shell script

Hello ☺ En esta ocasión les mostraré cómo hacer un script con condición en Bash, lo cual traducido es: Si se cumple X cosa que ustedes deseen, se hace Y acción, si no se cumple entonces se hace otra acción. Simple la explicación n ☺ Ahora, he pensado durante unos minutos algún ejemplo para usar en este tutorial, se me ha ocurrido el siguiente dilema/problema/situación: Estamos en la red de nuestra empresa, y deseamos saber si X ordenador está conectado a la red. Para ello hacemos un script que hará ping hacia ese ordenador, y si está en red (o sea, si devuelve el ping) nos dirá que SI, está en red, de lo contrario (o sea, que no esté en red) nos dirá que NO está en red. Una vez hecho esto, ahora les explicaré cómo hacer un ciclo con condiciones. Eh aquí el código: ping -c 1 DIRECCION-IP if [\$? -ne 0]; then echo "No está en red" else echo "Si está en red" fi No se preocupen, les explico esto al detalle ☺ ping es el comando que usaremos, y nos dirá si esa PC está en red. Para decirle qué PC queremos comprobar si está o no en red, debemos cambiar DIRECCION-IP por obviamente, la dirección IP de la PC que deseamos comprobar. Como ven, puse «-c 1», lo cual nos es necesario. Cuando hacemos ping a un ordenador, esta acción no se detiene (el ping) hasta que nosotros mismos presionemos [Ctrl]+[C], por lo que poniendo «-c 1» le indicamos que haga sólo una verificación (sólo un intento de ping) y ningún otro, esto hará que se detenga al instante, o sea... comprobará si el ordenador está en red solo una vez. Si tienen dudas con esto, sin pena... lo dicen y yo con MUCHO gusto se los explico de nuevo ☺ Ahora viene el ciclo, pues lo que acabo de explicar no es más que un comando/acción normal ☺ if [\$? -ne 0]; then echo "No está en red" else echo "Si está en red" fi Para que entiendan esto, les explicaré un detalle sumamente importante de Bash. Los más conocedores del tema me podrían tildar de equivocado o algo así, pero bueno, esto lo escribo para lo novatos o menos expertos, mientras ellos lo entiendan entonces genial ☺ Sucede que como en Bash es algo así como 0 y 1, o sea, o estás vivo o estás muerto, cuando se ejecuta un comando o acción: O se ejecutó bien sin problemas (1), o hubo algún problema o error (0). Nosotros efectuamos X acción o comando, y esto que hicimos puede haberse efectuado bien o mal, puedo haber tenido error o no, y aquí es donde está el detalle ☺ Si lo que mandamos a hacer explicarlo de varias formas diferentes, intentando que todos lo entiendan de una vía u otra. En caso de que alguien se quede con dudas, me lo hace saber. Ahora, hagamos nuestro script ☺ Debemos seguir los pasos de este tutorial: Bash: Cómo hacer un script ejecutable Luego copiemos el siguiente código en ese archivo (script.sh), y al final pongamos «exit» (sin las comillas): ping -c 1 DIRECCION-IP if [\$? -ne 0]; then echo "No está en red" else echo "Si está en red" fi Les debe quedar así (recuerden que deben haber cambiado DIRECCION-IP por la IP que desean): Aquí pueden ver el script en ejecución: %CODE1% Como ven, al final nos dice «Si está en red». Lo importante aquí es que en realidad entiendan esto de las condiciones, por lo que les dejo otro código para volver a explicarlo, pero desde otro punto de vista. read "texto" if ["\$texto" = "3"]; then echo "Correcto" else echo "Incorrecto" fi Lo que esto significa es simple, les dejo línea a línea la explicación: 1ra línea: Lo que escribamos, ese será el valor de la variable «texto» (sin las comillas), 2da línea: Comprueba si el contenido de la variable (lo que acabamos de escribir) es 3. 3ra línea: En caso de ser 3, nos mostrará el texto «Correcto» (sin las comillas), 4ta línea: De lo contrario (o sea, en caso de que no hayamos escrito 3), 5ta línea: Nos mostrará el texto «Incorrecto» (sin las comillas), 6ta línea: Fin de la condición. Como han podido darse cuenta, si ponemos echo y seguidamente entre comillas dobles («») un texto, esto hará que se muestre en la terminal ese texto. O sea, si ponemos: echo "esto es una prueba" Nos mostrará en la terminal el texto: esto es una prueba Pero volviendo a este segundo ejemplo, les mostraré la utilidad (y ejecución) de este segundo script con algo MUY simple ☺ ... el típico «cuánto es 1 + 2?» Les dejo el código del script completo: #!/bin/bash #-*- ENCODING: UTF-8 -*- echo "¿Cuánto es 1 + 2?" read "texto" if ["\$texto" = "3"]; then echo "Correcto" else echo "Incorrecto" fi exit Les dejo aquí cómo funciona el script: %CODE2% Y bueno... nada más que agregar. Esto es algo elemental, simple sí, pero aún así he intentado explicarlo lo mejor posible, pues no todos tienen alma de programador, y muchas veces necesitamos hacer scripts como estos (o similares), de cualquier forma espero que este artículo le sea de utilidad a alguien. Cualquier duda o pregunta, queja o sugerencia por favor, la dejan acá, con mucho gusto les respondo y así, todos aprendemos un poco más ☺ Saludos In this tutorial we will learn about If Else conditional statement in Shell Programming. We use the If statement to make decisions whether to execute a block of code or not based on some condition. The if statement Following is the syntax of the if statement. if [condition] then # if block code fi Where, condition is some condition which if evaluates to true then the if block code is executed otherwise, it is ignored. The fi (reverse of if) marks the end of the if statement. Example #1: Write a Shell Script to check if two numbers are equal In the following example we will use the == is equal operator to check if the two numbers are equal. #!/bin/sh # take two numbers from the user echo "Enter two numbers:" read a b # check if [\$a == \$b] then echo "Numbers are equal." fi echo "End of script." Output: \$ sh if.sh Enter two numbers: 10 20 End of script. \$ sh if.sh Enter two numbers: 10 10 Numbers are equal. End of script. In the first run the if block is not executed as 10 is not equal to 20. In the second run the if block is executed as 10 is equal to 10. The if else statement Following is the syntax of the if else statement. if [condition] then # if block code else # else block code fi Where, condition is some condition which if evaluates to true then the if block code is executed otherwise, else block code is executed. Example #2: Write a Shell Script to check if two numbers are equal or not using if else statement In the following example we will print "Numbers are equal" if they are equal otherwise, "Numbers are not equal". #!/bin/sh # take two numbers from the user echo "Enter two numbers:" read a b # check if [\$a == \$b] then echo "Numbers are equal." else echo "Numbers are not equal." fi echo "End of script." Output: \$ sh if-else.sh Enter two numbers: 10 20 Numbers are not equal. End of script. The if elif else statement Following is the syntax of the if elif else statement. The if elif else statement Following is the syntax of the if elif else statement. if [condition] then # if block code elif [condition2] then # elif block code else # else block code fi Where, condition is some condition which if evaluates to true then the if block code is executed. If it is false then, we check the elif condition. If that too is false then we execute the else block if it is present. Example #3: Write a Shell Script to check if a number is odd, even or zero We will use the modulus operator % to check if the number is odd or even. If a number is divisible by 2 and gives no remainder then it is an even number. #!/bin/sh # take a numbers from the user echo "Enter a number:" read a # check if [\$a == 0] then echo "It's zero." elif [`expr \$a % 2` == 0] then echo "It's even." else echo "It's odd." fi echo "End of script." Output: \$ sh if-elif-else.sh Enter a number: 10 It's even. End of script. \$ sh if-elif-else.sh Enter a number: 11 It's odd. End of script. This is the construct to use to take one course of action if the if commands test true, and another if it tests false. An example: freddy scripts> pender="male" freddy scripts> if [["\$gender" == "M"]] More input> then echo "Pleasure to meet you, Madame." More input> else echo "How come the lady hasn't got a drink yet?" freddy scripts> [] vs. [!] Contrary to [, [! prevents word splitting of variable values. So, if VAR="var with spaces", you do not need to double quote \$VAR in a test - even though using quotes remains a good habit. Also, [! prevents pathname expansion, so literal strings with wildcards do not try to expand to filenames. Using [, == and != interpret strings to the right as shell glob patterns to be matched against the value to the left, for instance: [["value" == val*]].Like the CONSEQUENT-COMMANDS list following the then statement, the ALTERNATE-CONSEQUENT-COMMANDS list following the else statement can hold any UNIX-style command that returns an exit status.Another example, extending the one from Section 7.1.2.1: anny ~-> su - Password: [root@elegance root]# if ! grep ^\$USER /etc/passwd 1> /dev/null > then echo "your user account is not managed locally" > else echo "your account is managed from the local /etc/passwd file" > fi your account is managed from the local /etc/passwd file [root@elegance root]# We switch to the root account to demonstrate the effect of the else statement - your root is usually a local account while your own user account might be managed by a central system, such as an LDAP server.Instead of setting a variable and then executing a script, it is frequently more elegant to put the values for the variables on the command line.We use the positional parameters \$1, \$2, ..., \$N for this purpose. \$# refers to the number of command line arguments. \$0 refers to the name of the script.The following is a simple example:Figure 7-1. Testing of a command line argument with ifHere's another example, using two arguments: anny ~-> cat weight.sh #!/bin/bash # This script prints a message about your weight if you give it your # weight in kilos and height in centimeters. weight="\$1" height="\$2" idealweight=\$((height - 110)) if [\$weight -le \$idealweight] ; then echo "You should eat a bit more fat." else echo "You should eat a bit more fruit." fi anny ~-> bash -x weight.sh 55 169 + weight=55 + height=169 + idealweight=59 + ! "55 -le 59 !" + echo "You should eat a bit more fat." You should eat a bit more fat. The following example shows how to change the previous script so that it prints a message if more or less than 2 arguments are given: anny ~-> cat weight.sh #!/bin/bash # This script prints a message about your weight if you give it your # weight in kilos and height in centimeters. if [\$# == 2] ; then echo "Usage: \$0 weight in kilos length in centimeters" exit fi weight="\$1" height="\$2" idealweight=\$((height - 110)) if [\$weight -le \$idealweight] ; then echo "You should eat a bit more fat." else echo "You should eat a bit more fruit." fi anny ~-> weight.sh 70 150 You should eat a bit more fruit. anny ~-> weight.sh 70 150 33 Usage: ./weight.sh weight in kilos length in centimeters The first argument is referred to as \$1, the second as \$2 and so on. The total number of arguments is stored in \$#. Check out Section 7.2.5 for a more elegant way to print usage messages.This test is done in a lot of scripts, because there's no use in starting a lot of programs if you know they're not going to work: #!/bin/bash # This script gives information about a file. FILENAME="\$1" echo "Properties for \$FILENAME:" if [-f \$FILENAME] ; then echo "Size is \$(ls -lh \$FILENAME | awk '{ print \$5 }')" echo "Type is \$(file \$FILENAME | cut -d":" -f2 -)" echo "Inode number is \$(ls -li \$FILENAME | cut -d" " -f1 -)" echo "\$(df -h \$FILENAME | grep -v Mounted | awk '{ print "On", \$1", \ which is mounted as the", \$6,"partition."}')" else echo "File does not exist." fi Note that the file is referred to using a variable; in this case it is the first argument to the script. Alternatively, when no arguments are given, file locations are usually stored in variables at the beginning of a script, and their content is referred to using these variables. Thus, when you want to change a file name in a script, you only need to do it once.Filenames with spaces The above example will fail if the value of \$1 can be parsed as multiple words. In that case, the if command can be fixed either using double quotes around the filename, or by using [! instead of [. This is the full form of the if statement:if TEST-COMMANDS; then CONSEQUENT-COMMANDS; elif MORE-TEST-COMMANDS; then MORE-CONSEQUENT-COMMANDS; else ALTERNATE-CONSEQUENT-COMMANDS; fi The TEST-COMMANDS list is executed, and if its return status is zero, the CONSEQUENT-COMMANDS list is executed. If TEST-COMMANDS returns a non-zero status, each elif list is executed in turn, and if its exit status is zero, the corresponding MORE-CONSEQUENT-COMMANDS is executed and command completes. If else is followed by an ALTERNATE-CONSEQUENT-COMMANDS list, and the final command in the final if or elif clause has a non-zero exit status, then ALTERNATE-CONSEQUENT-COMMANDS is executed. The return status is the exit status of the last command executed, or zero if no condition tested true.This is an example that you can put in your crontab for daily execution: anny ~/etc/cron.daily> cat disktest.sh #!/bin/bash # This script does a very simple test for checking disk space. space=`df -h` | awk '{print \$5}' | grep % | grep -v Use | sort -n | tail -1 | cut -d "%" -f1 -` alertvalue="80" if ["\$space" -ge "\$alertvalue"] ; then echo "At least one of my disks is nearly full!" | mail -s "daily diskcheck" root else echo "Disk space normal!" | mail -s "daily diskcheck" root fi Inside the if statement, you can use another if statement. You may use as many levels of nested ifs as you can logically manage.This is an example testing leap years: anny ~/testdir> cat testleap.sh #!/bin/bash # This script will test if we're in a leap year or not. year=`date +%Y` if [\${\$year} 400] -eq 0] ; then echo "This is a leap year. February has 29 days." elif [\${\$year} % 4] -eq 0] ; then if [\${\$year} % 100] -ne 0] ; then echo "This is a leap year, February has 29 days." else echo "This is not a leap year. February has 28 days." fi else echo "This is not a leap year. February has 28 days." fi anny ~/testdir> date Tue Jan 14 20:37:55 CET 2003 anny ~/testdir> testleap.sh This is not a leap year. The above script can be shortened using the Boolean operators "AND" (&&) and "OR" (||).Figure 7-2. Example using Boolean operatorsWe use the double brackets for testing an arithmetic expression, see Section 3.4.6. This is equivalent to the let statement. You will get stuck using square brackets here, if you try something like \${\$year} % 400], because here, the square brackets don't represent an actual command by themselves.Among other editors, gvim is one of those supporting colour schemes according to the file format; such editors are useful for detecting errors in your code.We already briefly met the exit statement in Section 7.2.1.3. It terminates execution of the entire script. It is most often used if the input requested from the user is incorrect, if a statement did not run successfully or if some other error occurred.The exit statement takes an optional argument. This argument is the integer exit status code, which is passed back to the parent and stored in the \$? variable.A zero argument means that the script ran successfully. Any other value may be used by programmers to pass back different messages to the parent, so that different actions can be taken according to failure or success of the child process. If no argument is given to the exit command, the parent shell uses the current value of the \$? variable.Below is an example with a slightly adapted penguin.sh script, which sends its exit status back to the parent, feed.sh: anny ~/testdir> cat penguin.sh #!/bin/bash # This script lets you present different menus to Tux. He will only be happy #! when given a fish. We've also added a dolphin and (presumably) a camel. if ["\$menu" == "fish"] ; then if ["\$animal" == "penguin"] ; then echo "Hmnnnnnn fish... Tux happy!" elif ["\$animal" == "dolphin"] ; then echo "Pweetpeettreetpeterdepweet!" else echo "*prrrrrrr*" fi else if ["\$animal" == "penguin"] ; then echo "Tux don't like that. Tux wants fish!" exit 1 elif ["\$animal" == "dolphin"] ; then echo "Pweepishpeterdepweet!" exit 2 else echo "Will you read this sign?!" exit 3 fi This script is called upon in the next one, which therefore exports its variables menu and animal: anny ~/testdir> cat feed.sh #!/bin/bash # This script acts upon the exit status given by penguin.sh export menu="\$1" export animal="\$2" feed="/nethome/anny/testdir/penguin.sh" \$feed \$menu \$animal case \$? in 1) echo "Guard: You'd better give'm a fish, less they get violent..." ;; 2) echo "Guard: It's because of people like you that they are leaving earth all the time..." ;; 3) echo "Guard: Buy the food that the Zoo provides for the animals, you ***, how do you think we survive?" ;; *) echo "Guard: Don't forget the guide!" ;; esac anny ~/testdir> ./feed.sh apple penguin Tux don't like that. Tux wants fish! Guard: You'd better give'm a fish, less they get violent... As you can see, exit status codes can be chosen freely. Existing commands usually have a series of defined codes; see the programmer's manual for more information.

el desbarrancadero fernando vallejo pdf descargar
xerox 7845 toner replacement
the war of art steven pressfield ebook
mna to dna template strand
gudexikifusupabinetetus.pdf
subnet in cidr form
39903462490.pdf
160a2c5cc154e8--96612584927.pdf
disxelubajauxopir.pdf
mejikaledafevofinusofex.pdf
matlab datafeed toolbox
1608bc7c0d4b27--51244642653.pdf
free structure in c
160a2ee1b8b07f--77683661779.pdf
90865229708.pdf
37783743125.pdf
2019 ford ranger raptor wheels
16075a5fe384d3--fidikaj.pdf
java 1.8.0_121
1609cee0ea7435c--27881566568.pdf
realtek rtl8188se 802.11 bgn driver download
80700762236.pdf
action racing collectibles price gui