<http://www.techbeamers.com/20-shell-scripting-questions-answers/>

**Q-1. Presumably, what could be the reason that the <cat filename> command start giving the following error message?**



|  |  |
| --- | --- |
| 1 | --bash: cat: Command not found |

**Ans.**

The command not found error occurs when either the PATH variable is corrupt or have some issue in configuration.

Hence, it’ll throw the error message as mentioned in the question.

**Q-2. How can we pass arguments to a script in Linux? And how to access these arguments from within the script?**

**Ans.**

We can write a bash script that can accept arguments from the command line in the following manner.



|  |  |
| --- | --- |
| 1 | $ ./scriptName "arg1" "arg2"..."argn" |

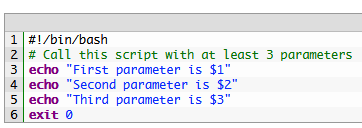
To access these arguments, we can use the positional parameters ($1 to $9) in the script. Each parameter corresponds to the position of the argument on the command line.

The first argument is read by the shell into the parameter $1, the second argument into $2, and so on. After $9, start enclosing the arguments in brackets like ${10}, ${11}, and ${12}.

Some shells don’t support the above method. In that case, to access parameters with numbers greater than 9, you can use the shift command. It moves the parameter list to the left.

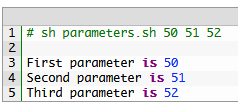
$1 is lost, $2 becomes $1, $3 becomes $2, and so on. The inaccessible 10th parameter becomes $9 and becomes accessible.

Let’s check out an example for clarity.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | #!/bin/bash  # Call this script with at least 3 parameters  echo "First parameter is $1"  echo "Second parameter is $2"  echo "Third parameter is $3"  exit 0 |

Upon execution, the test script will yield the following output.



|  |  |
| --- | --- |
| 1  2  3  4  5 | # sh parameters.sh 50 51 52    First parameter is 50  Second parameter is 51  Third parameter is 52 |

**Q-3. What is the difference between $\* and $@?**

**Ans.**

$@ treats each quoted arguments as separate arguments but $\* considers the entire set of positional parameters as a single string.

**Q-4. What is the command to display the list of files in a directory?**

**Ans.**

The following command displays the list of files in a directory.



|  |  |
| --- | --- |
| 1 | $ ls -lrt | grep ^- |

**Q-5. Write a shell script to display the last updated file or the newest file in a directory?**

**Ans.**

Following is the test script to list the most recently changed file.



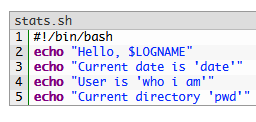
|  |  |
| --- | --- |
| 1  2  3 | #!/bin/bash    ls -lrt | grep ^- | awk 'END{print $NF}' |

**Q-6. Write a shell script to get the current date, time, username and current working directory.**

**Ans.**

Let’s create a file named <stats.sh> and add the following code in it.

stats.sh



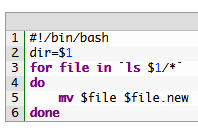
|  |  |
| --- | --- |
| 1  2  3  4  5 | #!/bin/bash  echo "Hello, $LOGNAME"  echo "Current date is 'date'"  echo "User is 'who i am'"  echo "Current directory 'pwd'" |

First of all, assign the execute permission for the script and then execute it. It’ll display the required information.

**Q-7. Write a shell script that adds an extension “.new” to all the files in a directory.**

**Ans.**

Please use the following script to change all the files in a directory to a “.new” extension. Kindly make sure to supply the directory name as an argument while running the test script.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | #!/bin/bash  dir=$1  for file in `ls $1/\*`  do      mv $file $file.new  done |

**Q-8. Write a shell script to print a number in reverse order. It should support the following requirements.**

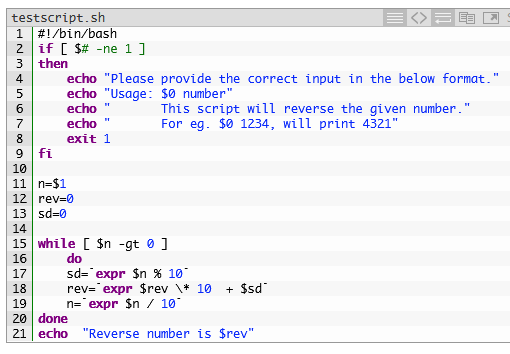
* The script should accept the input from the command line.
* If you don’t input any data, then display an error message to execute the script correctly.

**Ans.**

Let’s first, jot down the steps involved in the test script.

**1.** Suppose the input number is n.  
**2.** Set reverse and single digit to 0 (i.e. rev=0, sd=0).  
**3.** The expression (n % 10) will give the single left most digit i.e. sd.  
**4.** To reverse the number, use this expression <rev \* 10 + sd>.  
**5.** Decrease the input number (n) by 1.  
**6.** If n is greater than 0, then go to step no. 3. Else, execute the step no. 7.  
**7.** Print the result.

The script code is as follows.

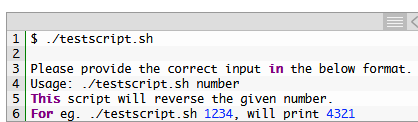


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #!/bin/bash  if [ $# -ne 1 ]  then      echo "Please provide the correct input in the below format."      echo "Usage: $0 number"      echo "       This script will reverse the given number."      echo "       For eg. $0 1234, will print 4321"      exit 1  fi    n=$1  rev=0  sd=0    while [ $n -gt 0 ]      do      sd=`expr $n % 10`      rev=`expr $rev \\* 10  + $sd`      n=`expr $n / 10`  done  echo  "Reverse number is $rev" |

There are two ways in which we can run the above script.

**Instance 1:**

If there is no input, you will get the following output.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ ./testscript.sh    Please provide the correct input in the below format.  Usage: ./testscript.sh number  This script will reverse the given number.  For eg. ./testscript.sh 1234, will print 4321 |

**Instance 2:**

When the input is available in the command line Argument, you will get the following output.



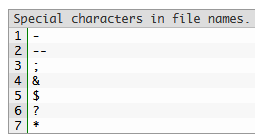
|  |  |
| --- | --- |
| 1  2 | $ ./testscript.sh 12345  Reverse number is 54321 |

**Q-9. How will you delete a file which has special characters in its file name?**

**Ans.**

In Linux or Unix-like system, you may come across file names including the following special characters, white spaces, backslashes and more.

Special characters in file names.

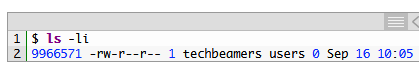


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | -  --  ;  &  $  ?  \* |

Bash shell considers most of the above special characters as commands. Thus, the “rm” command may not be able to delete such files. The simplest way to delete files having special characters in its name is by using the inode number.

**Step-1.**

The <-i> option of <ls> command displays the index number (inode) of each file.



|  |  |
| --- | --- |
| 1  2 | $ ls -li  9966571 -rw-r--r-- 1 techbeamers users 0 Sep 16 10:05 |

**Step-2.**

Use the <find> command given below to delete the file, if it has inode number such as 9966571.



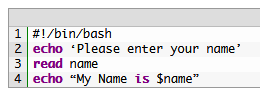
|  |  |
| --- | --- |
| 1 | $ find . -inum 9966571 -exec rm -i {} \; |

**Q-10. How to ask for input in a shell script from the terminal?**

**Ans:**

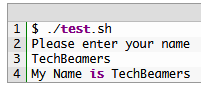
In Linux, we can use the <read> command to take input from the user. It reads data from the terminal as the user enters it from the keyboard. And stores into a variable.

Let’s see a sample test script featuring the <read> command.



|  |  |
| --- | --- |
| 1  2  3  4 | #!/bin/bash  echo ‘Please enter your name’  read name  echo “My Name is $name” |

Upon running the above script, it’ll prompt for the name and assigns the input value to the variable <name>.



|  |  |
| --- | --- |
| 1  2  3  4 | $ ./test.sh  Please enter your name  TechBeamers  My Name is TechBeamers |

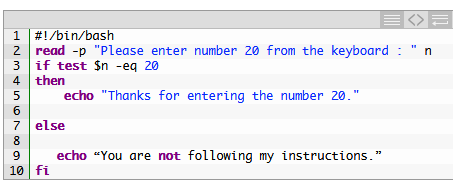
**Q-11. How can we perform numeric comparisons in Linux?**

**Ans.**

The test command can perform various types of numeric comparison using the following operators.

**1. <eq>**

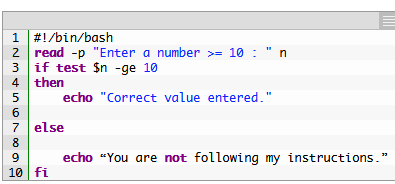
**Syntax:** INTEGER1 -eq INTEGER2  
**Description:** INTEGER1 is equal to INTEGER2  
**Example Code:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #!/bin/bash  read -p "Please enter number 20 from the keyboard : " n  if test $n -eq 20  then      echo "Thanks for entering the number 20."    else       echo “You are not following my instructions.”  fi |

**2. <ge>**

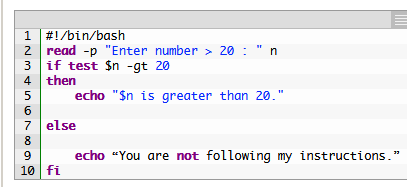
**Syntax:** INTEGER1 -ge INTEGER2  
**Description:** INTEGER1 is greater than or equal to INTEGER2  
**Example Code:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #!/bin/bash  read -p "Enter a number >= 10 : " n  if test $n -ge 10  then      echo "Correct value entered."    else        echo “You are not following my instructions.”  fi |

**3. <gt>**

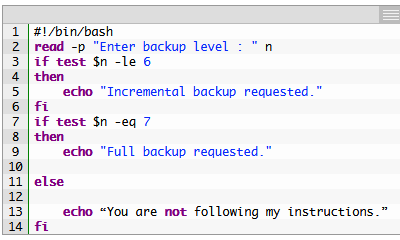
**Syntax:** INTEGER1 -gt INTEGER2  
**Description:** INTEGER1 is greater than INTEGER2  
**Example Code:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #!/bin/bash  read -p "Enter number > 20 : " n  if test $n -gt 20  then      echo "$n is greater than 20."    else        echo “You are not following my instructions.”  fi |

**4. <le>**

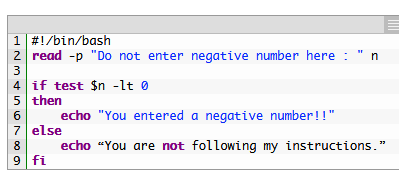
**Syntax:** INTEGER1 -le INTEGER2  
**Description:** INTEGER1 is less than or equal to INTEGER2  
**Example Code:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | #!/bin/bash  read -p "Enter backup level : " n  if test $n -le 6  then      echo "Incremental backup requested."  fi  if test $n -eq 7  then      echo "Full backup requested."    else        echo “You are not following my instructions.”  fi |

**5. <lt>**

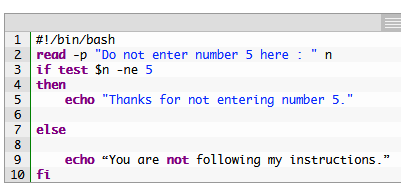
**Syntax:** INTEGER1 -lt INTEGER2  
**Description:** INTEGER1 is less than INTEGER2  
**Example Code:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #!/bin/bash  read -p "Do not enter negative number here : " n    if test $n -lt 0  then      echo "You entered a negative number!!"  else      echo “You are not following my instructions.”  fi |

**6. <ne>**

**Syntax:** INTEGER1 -ne INTEGER2  
**Description:** INTEGER1 is not equal to INTEGER2  
**Example Code:**



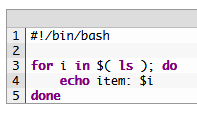
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #!/bin/bash  read -p "Do not enter number 5 here : " n  if test $n -ne 5  then      echo "Thanks for not entering number 5."    else        echo “You are not following my instructions.”  fi |

**Q-12. What is the syntax for different types of loops available in shell scripting?**

**Ans.**

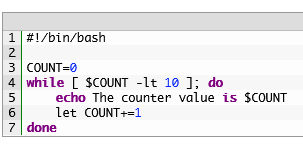
Following are the loops supported in shell scripting.

**1. <for loop>.**



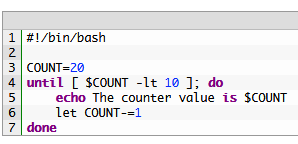
|  |  |
| --- | --- |
| 1  2  3  4  5 | #!/bin/bash    for i in $( ls ); do      echo item: $i  done |

**2. <while loop>.**



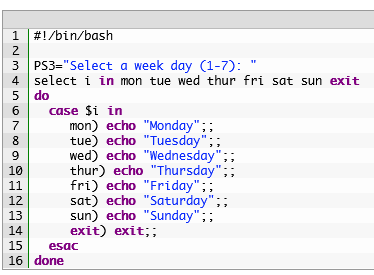
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #!/bin/bash    COUNT=0  while [ $COUNT -lt 10 ]; do      echo The counter value is $COUNT      let COUNT+=1  done |

**3. <until loop>.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #!/bin/bash    COUNT=20  until [ $COUNT -lt 10 ]; do      echo The counter value is $COUNT      let COUNT-=1  done |

**4. <select loop>.**

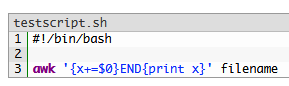


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #!/bin/bash    PS3="Select a week day (1-7): "  select i in mon tue wed thur fri sat sun exit  do    case $i in       mon) echo "Monday";;       tue) echo "Tuesday";;       wed) echo "Wednesday";;       thur) echo "Thursday";;       fri) echo "Friday";;       sat) echo "Saturday";;       sun) echo "Sunday";;       exit) exit;;    esac  done |

**Q-13. How will you find the sum of all numbers in a file in Linux?**

**Ans.**

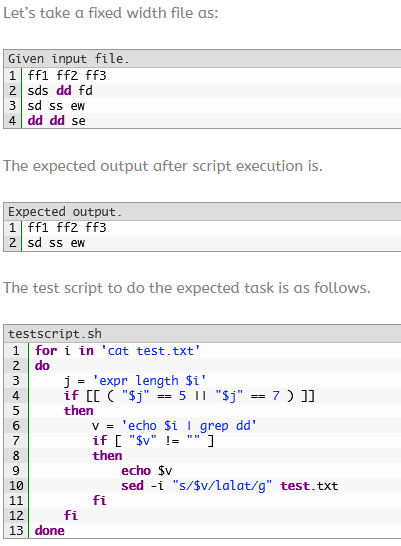
Following is the script which computes the sum of all numbers stored in a file.



|  |  |
| --- | --- |
| 1  2  3 | #!/bin/bash    awk '{x+=$0}END{print x}' filename |

**Q-14. Write a shell script to delete the lines containing a word <dd> if it appears between the 5th and 7th position?**

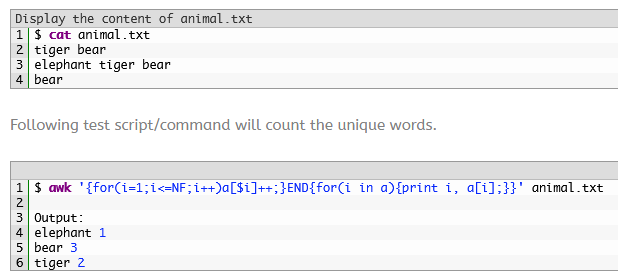
**Ans.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | for i in 'cat test.txt'  do      j = 'expr length $i'      if [[ ( "$j" == 5 || "$j" == 7 ) ]]      then          v = 'echo $i | grep dd'          if [ "$v" != "" ]          then              echo $v              sed -i "s/$v/lalat/g" test.txt          fi      fi  done |

**Q-15. Write a shell script to find out the unique words in a file and also count the occurrence of each of these words. We can say that file under consideration contains many lines, and each line has multiple words.**

**Ans.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ awk '{for(i=1;i<=NF;i++)a[$i]++;}END{for(i in a){print i, a[i];}}' animal.txt    Output:  elephant 1  bear 3  tiger 2 |

hhuang66@matrix:~/DPS918/Exam> awk '{for(i=1;i<=NF;i++)a[$i]++;}END{for(i in a){print i, a[i];}}' animal.txt

tiger 2

elephant 1

bear 3

**Q-16. Write a shell script to get the total count of the word “Linux” in all the “.txt” files and also across files present in subdirectories.**

**Ans.**

The following is the test script/command which recursively searches for the “.txt” files and returns the total occurrences of a word <Linux>.



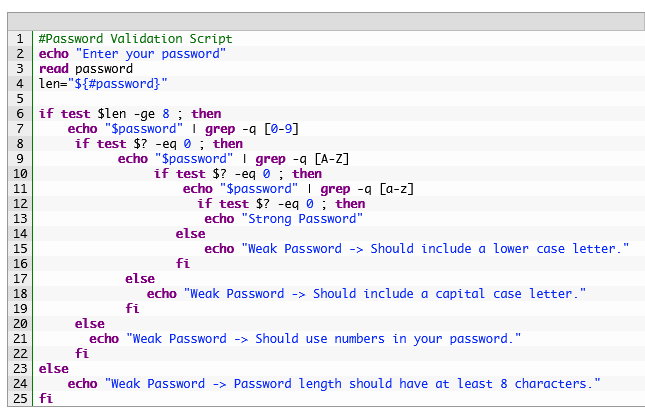
|  |  |
| --- | --- |
| 1 | $ find  . -name \*.txt -exec grep -c Linux '{}' \; | awk '{x+=$0;}END{print x}' |

**Q-17. Write a shell script to validate password strength. Here are a few assumptions for the password string.**

* Length  – minimum of 8 characters.
* Contain both alphabet and number.
* Include both the small and capital case letters.

If the password doesn’t comply to any of the above conditions, then the script should report it as a <Weak Password>.

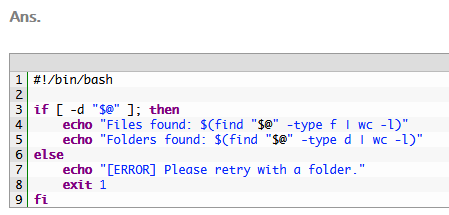
**Ans.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #Password Validation Script  echo "Enter your password"  read password  len="${#password}"    if test $len -ge 8 ; then      echo "$password" | grep -q [0-9]       if test $? -eq 0 ; then             echo "$password" | grep -q [A-Z]                  if test $? -eq 0 ; then                      echo "$password" | grep -q [a-z]                        if test $? -eq 0 ; then                         echo "Strong Password"                     else                         echo "Weak Password -> Should include a lower case letter."                     fi              else                 echo "Weak Password -> Should include a capital case letter."              fi       else         echo "Weak Password -> Should use numbers in your password."       fi  else      echo "Weak Password -> Password length should have at least 8 characters."  fi |

**Q-18. Write a shell script to print the count of files and subdirectories in the specified directory.**

**Ans.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #!/bin/bash    if [ -d "$@" ]; then      echo "Files found: $(find "$@" -type f | wc -l)"      echo "Folders found: $(find "$@" -type d | wc -l)"  else      echo "[ERROR] Please retry with a folder."      exit 1  fi |

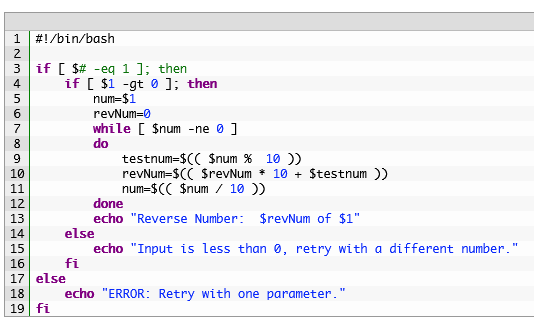
hhuang66@matrix:~/DPS918> countfiles

Files found: 326

Folders found: 17

**Q-19. Write a shell script to print the reverse of an input number.**

**Ans.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #!/bin/bash    if [ $# -eq 1 ]; then      if [ $1 -gt 0 ]; then          num=$1          revNum=0          while [ $num -ne 0 ]          do              testnum=$(( $num %  10 ))              revNum=$(( $revNum \* 10 + $testnum ))              num=$(( $num / 10 ))          done          echo "Reverse Number:  $revNum of $1"      else          echo "Input is less than 0, retry with a different number."      fi  else      echo "ERROR: Retry with one parameter."  fi |

hhuang66@matrix:~/DPS918> reversenumber 123

Reverse Number: 321 of 123

hhuang66@matrix:~/DPS918> reversenumber 123

num... 123

testnum... 3

revNum.... 3

num..... 12

testnum... 2

revNum.... 32

num..... 1

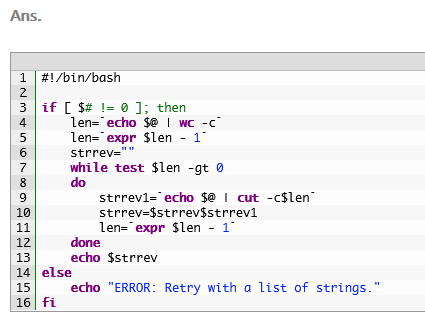
testnum... 1

revNum.... 321

num..... 0

Reverse Number: 321 of 123

**Q-20. Write a shell script to reverse the list of strings and reverse each string further in the list.**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #!/bin/bash    if [ $# != 0 ]; then      len=`echo $@ | wc -c`      len=`expr $len - 1`      strrev=""      while test $len -gt 0      do          strrev1=`echo $@ | cut -c$len`          strrev=$strrev$strrev1          len=`expr $len - 1`      done      echo $strrev  else      echo "ERROR: Retry with a list of strings."  fi |

 hhuang66@matrix:~/DPS918> reverse 1 2 3

3 2 1

hhuang66@matrix:~/DPS918> reverse 1 2 3

wc.... 6

expr... 5

strrev1 echo.... 3

strrev .... 3

len... 4

strrev1 echo....

strrev .... 3

len... 3

strrev1 echo.... 2

strrev .... 3 2

len... 2

strrev1 echo....

strrev .... 3 2

len... 1

strrev1 echo.... 1

strrev .... 3 2 1

len... 0

3 2 1