Write a script which:

1. Prompts the user for a directory name and then creates it with **mkdir**.
2. Changes to the new directory and prints out where it is using **pwd**.
3. Using **touch**, creates several empty files and runs **ls** on them to verify they are empty.
4. Puts some content in them using **echo** and redirection.
5. Displays their content using **cat**.
6. Says goodbye to the user and cleans up after itself.

Create a file named **testfile.sh**, with the content below:

**#!/bin/bash**

**# Prompts the user for a directory name and then creates it with mkdir.**

**echo "Give a directory name to create:"**  
**read NEW\_DIR**

**# Save original directory so we can return to it (could also just use pushd, popd)**

**ORIG\_DIR=$(pwd)**

**# check to make sure it doesn't already exist!**

**[[ -d $NEW\_DIR ]] && echo $NEW\_DIR already exists, aborting && exit**  
**mkdir $NEW\_DIR**

**# Changes to the new directory and prints out where it is using pwd.**

**cd $NEW\_DIR**  
**pwd**

**# Using touch, creates several empty files and runs ls on them to verify they are empty.**

**for n in 1 2 3 4**  
**do**  
 **touch file$n**  
**done**

**ls file?**

**# (Could have just done touch file1 file2 file3 file4, just want to show do loop!)**

**# Puts some content in them using echo and redirection.**

**for names in file?**  
**do**  
 **echo This file is named $names > $names**  
**done**

**# Displays their content using cat**

**cat file?**

**# Says goodbye to the user and cleans up after itself**

**cd $ORIG\_DIR**  
**rm -rf $NEW\_DIR**  
**echo "Goodbye My Friend!"**

Make it executable and run it:

**$ chmod +x testfile.sh**  
**./testfile.sh**

**Give a directory name to create:**

**/tmp/SOME\_DIR**

**/tmp/SOME\_DIR**  
**file1 file2 file3 file4**  
**This file is named file1**  
**This file is named file2**  
**This file is named file3**  
**This file is named file4**  
**Goodbye My Friend**

Write a script that takes exactly one argument, and prints it back out to standard output. Make sure the script generates a usage message if it is run without giving an argument.

Create a file named testarg.sh, with the content below:

**#!/bin/bash**  
**#**  
**# check for an argument, print a usage message if not supplied.**  
**#**  
**if [ $# -eq 0 ] ; then**  
 **echo "Usage: $0 argument"**  
 **exit 1**  
**fi**  
**echo $1**  
**exit 0**

Make it executable and run it:

**student:/tmp> chmod +x testarg.sh**  
**student:/tmp> ./testarg.sh Hello**

**Hello**

**student:/tmp>./testarg.sh**

**Usage: ./testarg.sh argument**  
**student:/tmp>**

Write a script which:

1. Asks the user for a number, which should be "1" or "2". Any other input should lead to an error report.
2. Sets an environmental variable to be "Yes" if it is "1", and "No" if it is "2".
3. Exports the environmental variable and displays it.

Create a file named **testenv.sh**, with the content below:

**#!/bin/bash**

**echo "Enter 1 or 2, to set the environmental variable EVAR to Yes or No"**  
**read ans**

**# Set up a return code**  
**RC=0**

**if [ $ans -eq 1 ]**  
**then**  
 **export EVAR="Yes"**  
**else**  
 **if [ $ans -eq 2 ]**  
 **then**  
 **export EVAR="No"**  
 **else**  
**# can only reach here with a bad answer**  
 **export EVAR="Unknown"**  
 **RC=1**  
 **fi**  
**fi**  
**echo "The value of EVAR is: $EVAR"**  
**exit $RC**

Make it executable and run it:

**student:/tmp> chmod +x testenv.sh**  
**student:/tmp> ./testenv.sh**

**Enter 1 or 2, to set the environmental variable EVAR to Yes or No**

**1**

**The value of EVAR is: Yes**

**student:/tmp> ./testenv.sh**

**Enter 1 or 2, to set the environmental variable EVAR to Yes or No**

**2**

**The value of EVAR is: No**

**student:/tmp> ./testenv.sh**

**Enter 1 or 2, to set the environmental variable EVAR to Yes or No**

**3**

**The value of EVAR is: Unknown**

Write a script which:

1. Asks the user for a number (1, 2 or 3).
2. Calls a function with that number in its name. The function should display a message with its name included.

Create a file named **testfun.sh**, with the content below:

**#!/bin/bash**

**# Functions (must be defined before use)**  
**func1() {**  
**echo " This message is from function 1"**  
**}**  
**func2() {**  
**echo " This message is from function 2"**  
**}**  
**func3() {**  
**echo " This message is from function 3"**  
**}**

**# Beginning of the main script**

**# prompt the user to get their choice**  
**echo "Enter a number from 1 to 3"**  
**read n**

**# Call the chosen function**  
**func$n**

Make it executable and run it:

**student:/tmp> chmod +x testfun.sh**  
**student:/tmp> ./testfun.sh**

**Enter a number from 1 to 3**

**2**

**This message is from function 2**

**$ ./testfun.sh**

**Enter a number from 1 to 3**

**7**

**./testfun.sh: line 21: func7: command not found**

Write a script that will act as a simple calculator for add, subtract, multiply and divide.

1. Each operation should have its own function.
2. Any of the three methods for bash arithmetic, (**$((..))**, **let**, or **expr**) may be used.
3. The user should give 3 arguments when executing the script:  
   - The first should be one of the letters **a**, **s**, **m**, or **d** to specify which math operation.  
   - The second and third arguments should be the numbers that are being operated on.
4. The script should detect for bad or missing input values and display the results when done.

Create a file named **testmath.sh**, with the content below:

**#!/bin/bash**

**# Functions. must be before the main part of the script**

**# in each case method 1 uses $((..))**  
**# method 2 uses let**  
**# method 3 uses expr**

**add() {**  
 **answer1=$(($1 + $2))**  
 **let answer2=($1 + $2)**  
 **answer3=`expr $1 + $2`**  
**}**  
**sub() {**  
 **answer1=$(($1 - $2))**  
 **let answer2=($1 - $2)**  
 **answer3=`expr $1 - $2`**  
**}**  
**mult() {**  
 **answer1=$(($1 \* $2))**  
 **let answer2=($1 \* $2)**  
 **answer3=`expr $1 \\* $2`**  
**}**  
**div() {**  
 **answer1=$(($1 / $2))**  
 **let answer2=($1 / $2)**  
 **answer3=`expr $1 / $2`**  
**}**  
**# End of functions**  
**#**  
**# Main part of the script**  
**# need 3 arguments, and parse to make sure they are valid types**  
**op=$1 ; arg1=$2 ; arg2=$3**  
**[[ $# -lt 3 ]] && \**  
 **echo "Usage: Provide an operation (a,s,m,d) and two numbers" && exit 1**  
**[[ $op != a ]] && [[ $op != s ]] && [[ $op != d ]] && [[ $op != m ]] && \**  
 **echo operator must be a, s, m, or d, not $op as supplied**  
**# ok, do the work!**

**if [[ $op == a ]] ; then add $arg1 $arg2**  
**elif [[ $op == s ]] ; then sub $arg1 $arg2**  
**elif [[ $op == m ]] ; then mult $arg1 $arg2**  
**elif [[ $op == d ]] ; then div $arg1 $arg2**  
**else**  
**echo $op is not a, s, m, or d, aborting ; exit 2**  
**fi**  
**# Show the answers**  
**echo $arg1 $op $arg2 :**  
**echo 'Method 1, $((..)),' Answer is $answer1**  
**echo 'Method 2, let, ' Answer is $answer2**  
**echo 'Method 3, expr, ' Answer is $answer3**

Make it executable and run it:

**student:/tmp> chmod +x testmath.sh**  
**student:/tmp> ./testmath.sh**  
**student:/tmp> for n in a s m d x ; do ./testmath.sh $n 21 7 ; done**

**21 a 7 :**  
**Method 1, $((..)), Answer is 28**  
**Method 2, let,**  
**Answer is 28**  
**Method 3, expr,**  
**Answer is 28**  
**21 s 7 :**  
**Method 1, $((..)), Answer is 14**  
**Method 2, let,**  
**Answer is 14**  
**Method 3, expr,**  
**Answer is 14**  
**21 m 7 :**  
**Method 1, $((..)), Answer is 147**  
**Method 2, let,**  
**Answer is 147**  
**Method 3, expr,**  
**Answer is 147**  
**21 d 7 :**  
**Method 1, $((..)), Answer is 3**  
**Method 2, let,**  
**Answer is 3**  
**Method 3, expr,**  
**Answer is 3**  
**operator must be a, s, m, or d, not x as supplied**  
**x is not a, s, m, or d, aborting**

Write a script which reads two strings as arguments and then:

1. Tests to see if the first string is of zero length, and if the other is of non-zero length, telling the user of both results.
2. Determines the length of each string, and reports on which one is longer or if they are of equal length.
3. Compares the strings to see if they are the same, and reports on the result.

Create a file named **teststrings.sh**, with the content below:

**#!/bin/bash**

**# check two string arguments were given**

**[[ $# -lt 2 ]] && \**  
 **echo "Usage: Give two strings as arguments" && exit 1**  
**str1=$1**  
**str2=$2**

**#------------------------------------**  
**## test command**

**echo "Is string 1 zero length? Value of 1 means FALSE"**  
**[ -z "$str1" ]**  
**echo $?**  
**# note if $str1 is empty, the test [ -z $str1 ] would fail**  
**# but [[ -z $str1 ]] succeeds**  
**# i.e., with [[ ]] it works even without the quotes**  
**echo "Is string 2 nonzero length? Value of 0 means TRUE;"**  
**[ -n $str2 ]**  
**echo $?**

**## comparing the lengths of two string**

**len1=${#str1}**  
**len2=${#str2}**  
**echo length of string1 = $len1, length of string2 = $len2**

**if [ $len1 -gt $len2 ]**  
**then**  
 **echo "String 1 is longer than string 2"**  
**else**  
 **if [ $len2 -gt $len1 ]**  
 **then**  
 **echo "String 2 is longer than string 1"**  
 **else**  
 **echo "String 1 is the same length as string 2"**  
 **fi**  
**fi**

**## compare the two strings to see if they are the same**

**if [[ $str1 == $str2 ]]**  
**then**  
 **echo "String 1 is the same as string 2"**  
**else**  
 **if [[ $str1 != $str2 ]]**  
 **then**  
 **echo "String 1 is not the same as string 2"**  
 **fi**  
**fi**

**student:/tmp> chmod +x teststrings.sh**  
**student:/tmp> ./teststrings.sh str1 str2**

**Is string 1 zero length? Value of 1 means FALSE**  
**1**  
**Is string 2 nonzero length? Value of 0 means TRUE;**  
**0**  
**length of string1 = 4, length of string2 = 4**  
**String 1 is the same length as string 2**  
**String 1 is not the same as string 2**

**student:/tmp>./teststrings.sh str1 str2long**

**Is string 1 zero length? Value of 1 means FALSE**  
**1**  
**Is string 2 nonzero length? Value of 0 means TRUE;**  
**0**  
**length of string1 = 4, length of string2 = 8**  
**String 2 is longer than string 1**  
**String 1 is not the same as string 2**

**student:/tmp>**

Write a script that takes as an argument a month in numerical form (i.e. between 1 and 12), and translates this to the month name and displays the result on standard out (the terminal).

If no argument is given, or a bad number is given, the script should report the error and exit.

Create a file named **testmonths.sh**, with the content below:

**#!/bin/bash**

**# Accept a number between 1 and 12 as**  
**# an argument to this script, then return the**  
**# the name of the month that corresponds to that number.**

**# Check to see if the user passed a parameter.**  
**if [ $# -eq 0 ]**  
**then**  
 **echo "Error. Give as an argument a number between 1 and 12."**  
 **exit 1**  
**fi**

**# set month equal to argument passed for use in the script**  
**month=$1**

**################################################**  
**# The example of a case statement:**  
  
**case $month in**

**1) echo "January" ;;**  
 **2) echo "February" ;;**  
 **3) echo "March" ;;**  
 **4) echo "April" ;;**  
 **5) echo "May" ;;**  
 **6) echo "June" ;;**  
 **7) echo "July" ;;**  
 **8) echo "August" ;;**  
 **9) echo "September" ;;**  
 **10) echo "October" ;;**  
 **11) echo "November" ;;**  
 **12) echo "December" ;;**  
 **\*)**  
 **echo "Error. No month matches: $month"**  
 **echo "Please pass a number between 1 and 12."**  
 **exit 2**  
 **;;**  
**esac**  
**exit 0**

Make it executable and run it:

**student:/tmp> chmod +x testcase.sh**  
**student:/tmp> ./testcase.sh 5**

**May**

**student:/tmp> ./testcase.sh 12**

**December**

**student:/tmp> ./testcase.sh 99**

**Error. No month matches: 99**  
**Please pass a number between 1 and 12**  
**student:/tmp>**

Write a script which:

1. Takes a word as an argument.
2. Appends a random number to it.
3. Displays the answer.

Create a file named **testrandom.sh**, with the content below:

**#!/bin/bash**  
**##**  
**# check to see if the user supplied in the parameter.**

**[[ $# -eq 0 ]] && echo "Usage: $0 word" && exit 1**

**echo "$1-$RANDOM"**  
**exit 0**

Make it executable and run it:

**student:/tmp> chmod +x testrandom.sh**  
**student:/tmp> ./testrandom.sh strA**

**strA-29294**

**student:/tmp>./testrandom.sh strB**

**strB-23911**

**student:/tmp>./testrandom.sh strC**

**strC-27782**  
**student:/tmp>**

1. Check to see if the enscript package has been installed on your system, and if not, install it.
2. Using enscript, convert the text file **/var/dmesg** to PostScript format and name the result **/tmp/dmesg.ps**. As an alternative, you can use any large text file on your system. Make sure you can read the PostScript file (for example with evince) and compare to the original file. ***NOTE****: On some systems, evince may have problems with the PostScript file, but the PDF file you produce from it will be fine for viewing.*
3. Convert the PostScript document to PDF format, using **ps2pdf**. Make sure you can read the resulting PDF file. Does it look identical to the PostScript version?
4. Is there a way you can go straight to the PDF file without producing a PostScript file on the disk along the way?
5. Using **pdfinfo**, determine what is the PDF version used to encode the file, the number of pages, the page size, and other metadata about the file. If you do not have **pdfinfo** you probably need to install the **poppler-utils** package.
6. Try:  
   **which enscript**  
   **/usr/bin/enscript**  
   If you do not get a positive result, install with whichever command is appropriate for your Linux distribution:  
   **apt-get install enscript**  
   **yum install enscript**  
   **zypper install enscript**
7. **enscript -p /tmp/dmesg.ps /var/log/dmesg**  
   **evince /tmp/dmesg.ps**
8. **ps2pdf /tmp/dmesg.ps**  
   **ls -lh /var/log/dmesg /tmp/dmesg.ps /tmp/dmesg.pdf**  
   **-rw-rw-r-- 1 coop coop 28K Apr 22 13:00 /tmp/dmesg.pdf**  
   **-rw-rw-r-- 1 coop coop 80K Apr 22 12:59 /tmp/dmesg.ps**  
   **-rw-r--r-- 1 root root 53K Apr 22 11:48 /var/log/dmesg**  
   **evince /tmp/dmesg.ps /tmp/dmesg.pdf**  
   Note the difference in sizes. PostScript files tend to be large, while PDF is a compressed format.
9. You may want to scan the man pages for **enscript** and **ps2pdf** to figure out how to use standard input or standard output instead of files.  
   **student:/tmp> enscript -p - /var/log/dmesg | ps2pdf - dmesg\_direct.pdf**  
   **[ 15 pages \* 1 copy ] left in -**  
   **85 lines were wrapped**  
   **student:/tmp> ls -l dmesg\*pdf**  
   **-rw-rw-r-- 1 coop coop 28177 Apr 22 13:20 dmesg\_direct.pdf**  
   **-rw-rw-r-- 1 coop coop 28177 Apr 22 13:00 dmesg.pdf**
10. **student:/tmp> pdfinfo dmesg.pdf**  
    **Title: Enscript Output**  
    **Author: Theodore Cleaver**  
    **Creator: GNU Enscript 1.6.6**  
    **Producer: GPL Ghostscript 9.07**  
    **CreationDate: Wed Apr 22 13:00:26 2015**  
    **ModDate: Wed Apr 22 13:00:26 2015**  
    **Tagged: no**  
    **Form: none**  
    **Pages: 15**  
    **Encrypted: no**  
    **Page size: 612 x 792 pts (letter)**  
    **Page rot: 0**  
    **File size: 28177 bytes**

You can convert two text files (you can create them or use ones that already exist since this is non-destructive) into PDFs, or you can use two pre-existing ones. Combine them into one PDF, and view the result. Do this using three different methods:

1. **qpdf**
2. **qdftk**
3. **gs**

If **pdftk** is not installed, you can try to install. However, if you are on a system for which it is no longer available, you will have to use **qpdf** or **gs**.

First, we can create two PDFs to play with, using **enscript** and then **ps2pdf**:

**cd /var/log**  
**enscript -p dmesg.ps /var/log/dmesg**  
**enscript -p boot.ps /var/log/boot.log**  
**ps2dpf dmesg.ps**  
**ps2dpf boot.ps**

Of course, you may use two pre-existing PDF files and substitute their names below.

* Method 1: Using **qpdf**:  
  **qpdf --empty --pages dmesg.pdf boot.pdf -- method1.pdf**
* Method 2: Using **pdftk**:  
  **pdftk dmesg.pdf boot.pdf cat output method2.pdf**
* Method 3: Using **gs**:  
  **gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=method3.pdf dmesg.pdf boot.pdf**

Now view them:

**ls -l method1.pdf method2.pdf**  
**evince method1.pdf method2.pdf**

How do the files compare?

1. Create a new user, using **useradd**, and give the user an initial password with **passwd**.
2. Configure this user to be able to use **sudo**.
3. Login as or switch to this new user and make sure you can execute a command that requires root privilege.

For example, a trivial command requiring root privilege could be:

**$ ls /root**

1. **sudo useradd newuser**  
   **sudo passwd newuser**  
   (give the password for this user when prompted)
2. With root privilege, (use **sudo visudo**) add this line to **/etc/sudoers**:  
   **newuser ALL=(ALL) ALL**  
   Alternatively, create a file named **/etc/sudoers.d/newuser** with just that one line as content.
3. You can login by doing:  
   **sudo su newuser**  
   or  
   **ssh newuser@localhost**  
   which will require giving **newuser**'s password, and is probably a better solution. Instead of **localhost** you can give your hostname, IP address or **127.0.0.1**. Then as **newuser** just type:  
   **sudo ls /root**