Bash Shell Scripting Constructs

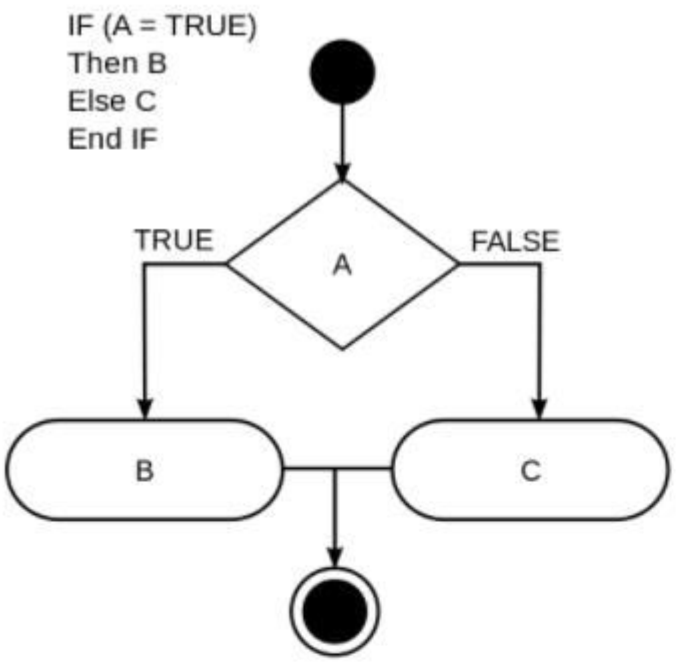
The if Statement:

1. Conditional decision making, using an if statement, is a basic construct that any useful programming or scripting language must have.
2. When an if statement is used, the ensuing actions depend on the evaluation of specified conditions, such as:
    - Numerical or string comparisons
    - Return value of a command (0 for success)
    - File existence or permissions.
3. In compact form, the syntax of an if statement is:

```
if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi
```

A more general definition is:

```
if condition
then
       statements
else
       statements
fi
```



The `if` Statement

Using the if Statement:

1. In the following example, an if statement checks to see if a certain file exists, and if the file is found, it displays a message indicating success or failure:

```
if [ -f "$1" ]
then
    echo file "$1 exists"
else
    echo file "$1" does not exist
fi
```

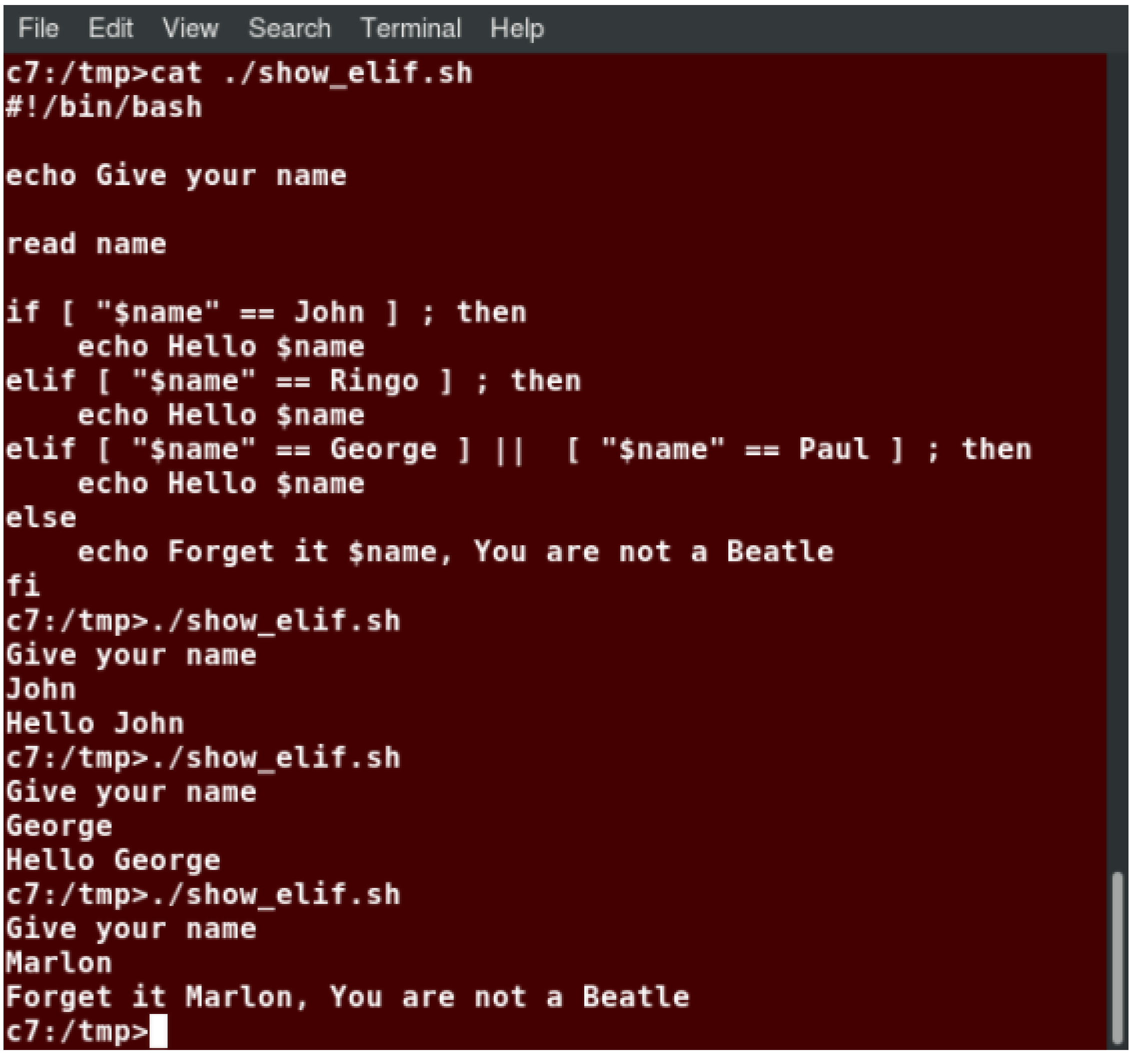
2. We really should also check first that there is an argument passed to the script ($1) and abort if not.
3. Notice the use of the square brackets (**[]**) to delineate the test condition. There are many other kinds of tests you can perform, such as checking whether two numbers are equal to, greater than, or less than each other and make a decision accordingly; we will discuss these other tests.
4. In modern scripts, you may see doubled brackets as in [[-f /etc/passwd]]. This is not an error. It is never wrong to do so and it avoids some subtle problems, such as referring to an empty environment variable without surrounding it in double quotes; we will not talk about this here.

The elif Statement:

1. You can use the **elif** statement to perform more complicated tests, and take action appropriate actions. The basic syntax is:

```
if [ sometest ] ; then
    echo Passed test1
elif [ somothertest ] ; then
    echo Passed test2
fi
```

2. In the example shown we use strings tests which we will explain shortly, and show how to pull in an environment variable with the read statement.



```
c7:/tmp>cat ./show_elif.sh
#!/bin/bash

echo Give your name

read name

if [ "$name" == John ] ; then
    echo Hello $name
elif [ "$name" == Ringo ] ; then
    echo Hello $name
elif [ "$name" == George ] ||  [ "$name" == Paul ] ; then
    echo Hello $name
else
    echo Forget it $name, You are not a Beatle
fi
c7:/tmp>./show_elif.sh
Give your name
John
Hello John
c7:/tmp>./show_elif.sh
Give your name
George
Hello George
c7:/tmp>./show_elif.sh
Give your name
Marlon
Forget it Marlon, You are not a Beatle
c7:/tmp>
```

The `elif` Statement

Testing for Files:

1. bash provides a set of file conditionals, that can be used with the if statement, including those in the table.
2. You can use the if statement to test for file attributes, such as:
   - File or directory existence
   - Read or write permission
   - Executable permission.
3. For example, in the following example:

```
if [ -x /etc/passwd ] ; then
    ACTION
fi
```

the `if` statement checks if the file `/etc/passwd` is executable, which it is not. Note the very common practice of putting:

```
; then
```

on the same line as the `if` statement.

You can view the full list of file conditions typing:

`man 1 test`.

Condition	Meaning
`-e file`	Checks if the file exists.
`-d file`	Checks if the file is a directory.
`-f file`	Checks if the file is a regular file (i.e. not a symbolic link, device node, directory, etc.)
`-s file`	Checks if the file is of non-zero size.
`-g file`	Checks if the file has **sgid** set.
`-u file`	Checks if the file has **suid** set.
`-r file`	Checks if the file is readable.
`-w file`	Checks if the file is writable.
`-x file`	Checks if the file is executable.

Boolean Expressions:

1. Boolean expressions evaluate to either **TRUE** or **FALSE**, and results are obtained using the various Boolean operators listed in the table.

Operator	Operation	Meaning
&&	**AND**	The action will be performed only if both the conditions evaluate to true.
||	**OR**	The action will be performed if any one of the conditions evaluate to true.
!	**NOT**	The action will be performed only if the condition evaluates to false.

2. Note that if you have multiple conditions strung together with the **&&** operator, processing stops as soon as a condition evaluates to false. For example, if you have **A && B && C** and **A** is true but **B** is false, **C** will never be executed.
3. Likewise, if you are using the **||** operator, processing stops as soon as anything is true. For example, if you have **A || B || C** and **A** is false and **B** is true, you will also never execute **C**.

Tests in Boolean Expressions:

1. Boolean expressions return either TRUE or FALSE. We can use such expressions when working with multiple data types, including strings or numbers, as well as with files. For example, to check if a file exists, use the following conditional test:

```
[ -e <filename> ]
```

2. Similarly, to check if the value of **number1** is greater than the value of **number2**, use the following conditional test:

```
[ $number1 -gt $number2 ]
```

The operator `-gt` returns TRUE if `number1` is greater than `number2`.

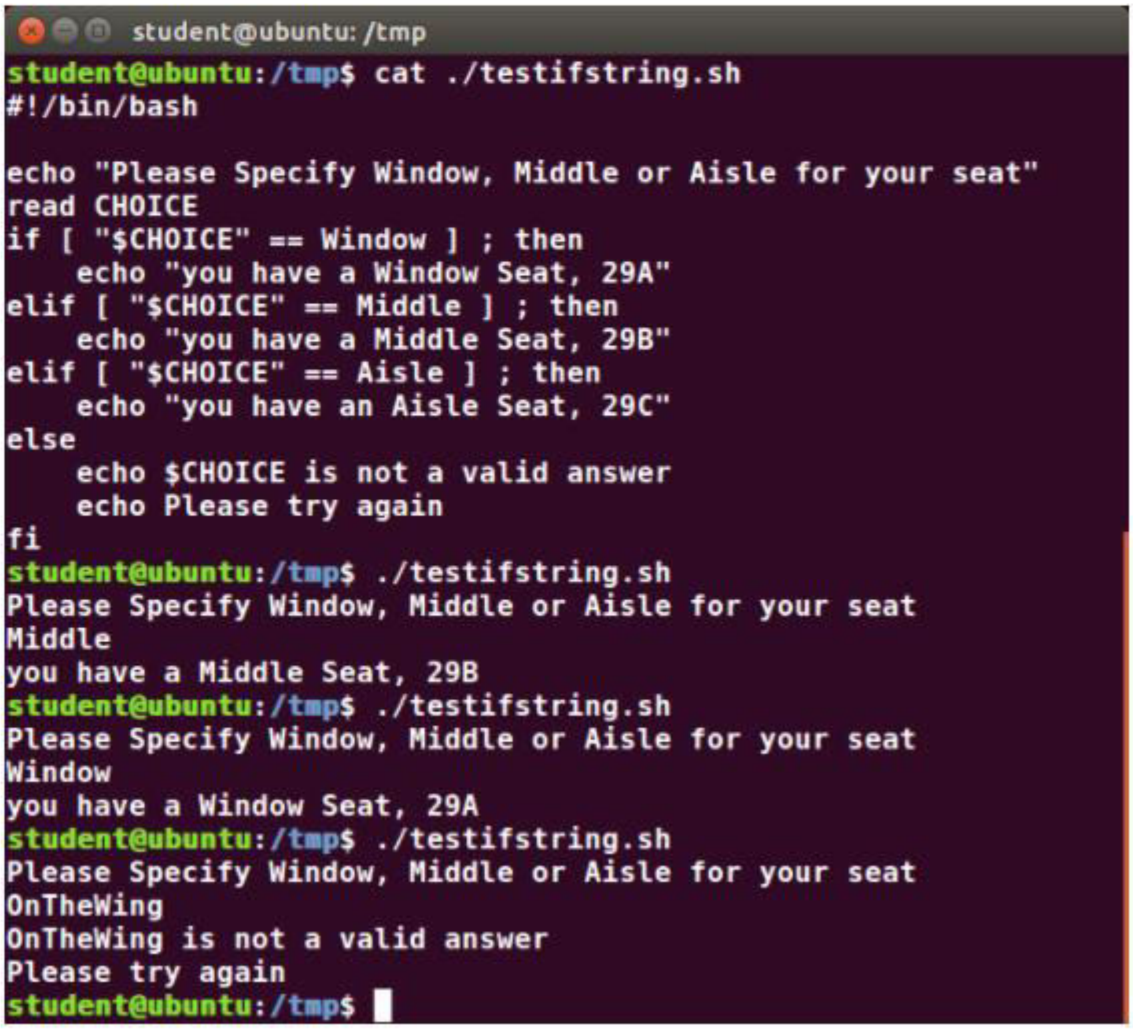
Example of Testing of Strings:

1. You can use the if statement to compare strings using the operator == (two equal signs). The syntax is as follows:

```
if [ string1 == string2 ] ; then
   ACTION
fi
```

2. Note that using one = sign will also work, but some consider it deprecated usage. Let’s now consider an example of testing strings.

3. In the example illustrated here, the if statement is used to compare the input provided by the user and accordingly display the result.



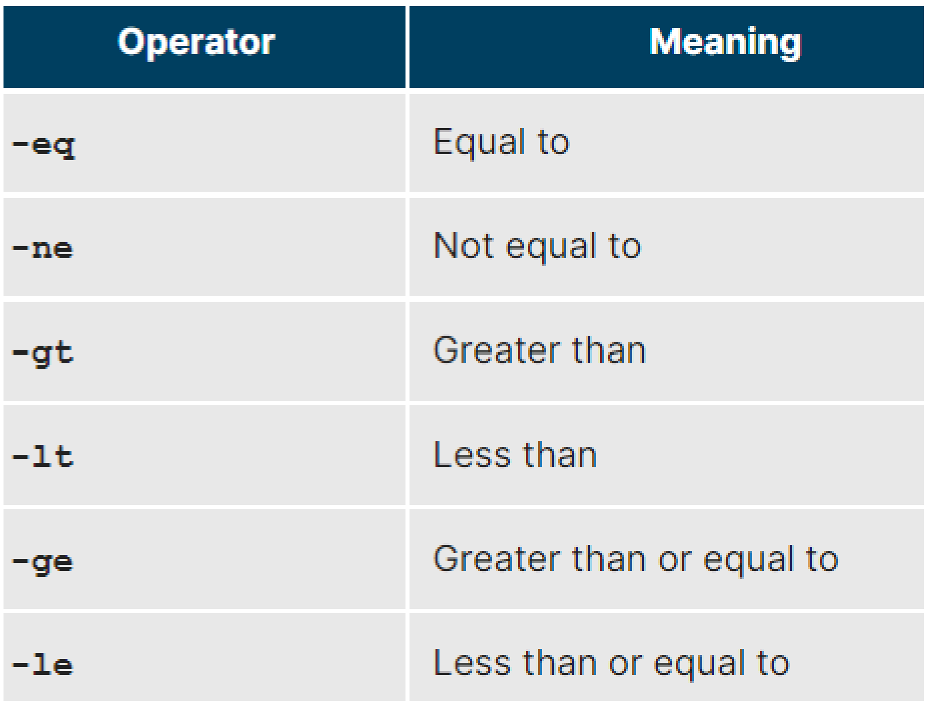
```
student@ubuntu:/tmp$ cat ./testifstring.sh
#!/bin/bash

echo "Please Specify Window, Middle or Aisle for your seat"
read CHOICE
if [ "$CHOICE" == Window ] ; then
    echo "you have a Window Seat, 29A"
elif [ "$CHOICE" == Middle ] ; then
    echo "you have a Middle Seat, 29B"
elif [ "$CHOICE" == Aisle ] ; then
    echo "you have an Aisle Seat, 29C"
else
    echo $CHOICE is not a valid answer
    echo Please try again
fi
student@ubuntu:/tmp$ ./testifstring.sh
Please Specify Window, Middle or Aisle for your seat
Middle
you have a Middle Seat, 29B
student@ubuntu:/tmp$ ./testifstring.sh
Please Specify Window, Middle or Aisle for your seat
Window
you have a Window Seat, 29A
student@ubuntu:/tmp$ ./testifstring.sh
Please Specify Window, Middle or Aisle for your seat
OnTheWing
OnTheWing is not a valid answer
Please try again
student@ubuntu:/tmp$
```

Example of Testing of Strings

Numerical Tests:

1. You can use specially defined operators with the if statement to compare numbers. The various operators that are available are listed in the table:



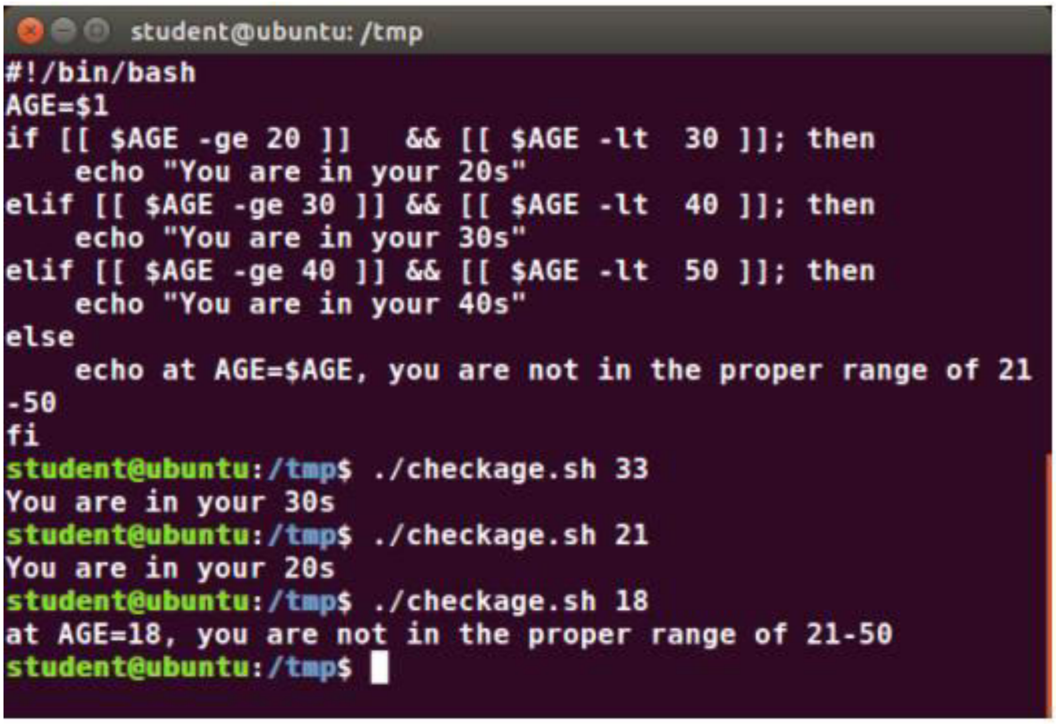
Operator	Meaning
`-eq`	Equal to
`-ne`	Not equal to
`-gt`	Greater than
`-lt`	Less than
`-ge`	Greater than or equal to
`-le`	Less than or equal to

The syntax for comparing numbers is as follows:

```
exp1 -op exp2
```

Example of Testing for Numbers:

1. Let us now consider an example of comparing numbers using the various operators:



```
student@ubuntu: /tmp
#!/bin/bash
AGE=$1
if [[ $AGE -ge 20 ]]   && [[ $AGE -lt  30 ]]; then
    echo "You are in your 20s"
elif [[ $AGE -ge 30 ]] && [[ $AGE -lt  40 ]]; then
    echo "You are in your 30s"
elif [[ $AGE -ge 40 ]] && [[ $AGE -lt  50 ]]; then
    echo "You are in your 40s"
else
    echo at AGE=$AGE, you are not in the proper range of 21
-50
fi
student@ubuntu:/tmp$ ./checkage.sh 33
You are in your 30s
student@ubuntu:/tmp$ ./checkage.sh 21
You are in your 20s
student@ubuntu:/tmp$ ./checkage.sh 18
at AGE=18, you are not in the proper range of 21-50
student@ubuntu:/tmp$
```

Example of Testing for Numbers

Arithmetic Expressions:

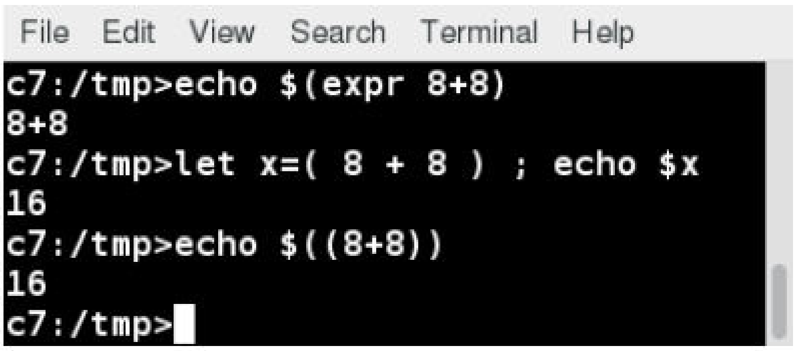
1. Arithmetic expressions can be evaluated in the following three ways (spaces are important!):

- Using the `expr` utility
  `expr` is a standard but somewhat deprecated program. The syntax is as follows:
  ```
  expr 8 + 8
  echo $(expr 8 + 8)
  ```

- Using the `$((...))` syntax
  This is the built-in shell format. The syntax is as follows:
  ```
  echo $((x+1))
  ```

- Using the built-in shell command `let`. The syntax is as follows:
  ```
  let x=( 1 + 2 ); echo $x
  ```

2. In modern shell scripts, the use of **expr** is better replaced with **var=$((...)).**



```
c7:/tmp>echo $(expr 8+8)
8+8
c7:/tmp>let x=( 8 + 8 ) ; echo $x
16
c7:/tmp>echo $((8+8))
16
c7:/tmp>
```

Arithmetic Expressions

Lab 15.6: Arithmetic and Functions

Bookmark this page

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
```