* Theme: light + (default light)

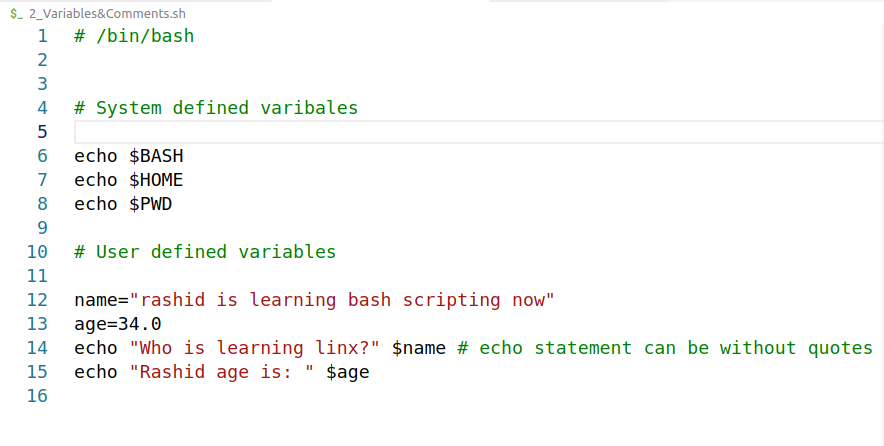
**1. Introduction**

* Shell scripts are interpreted not compiled
* To see installed shells in your machine
  + cat /etc/shells
* sh is borne shell….still used in unix system, the first shell which was used
* bash ...born again shell…. Standard gnu which is intutive and flexible
* to see where bash is located
  + which bash
* #! shebang/hashbang

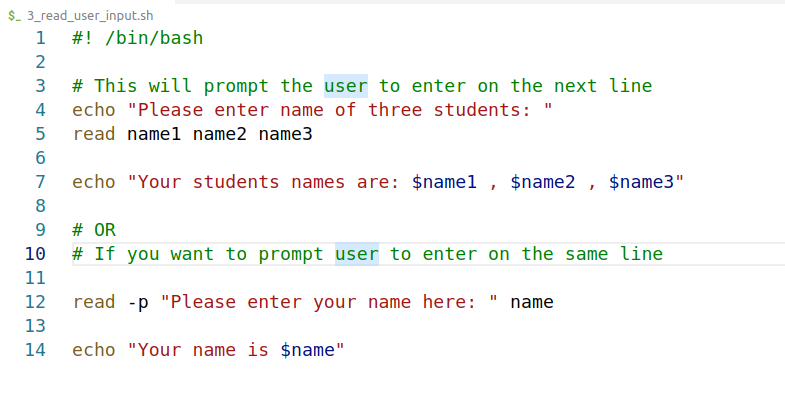
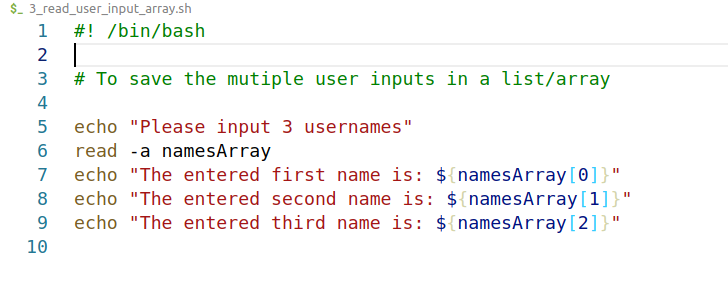
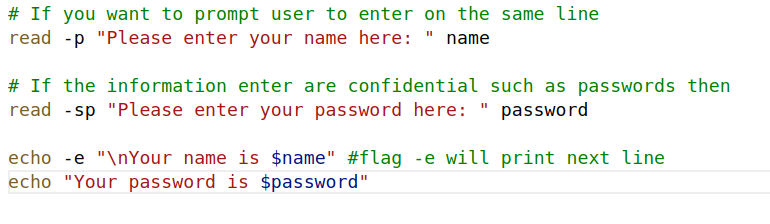


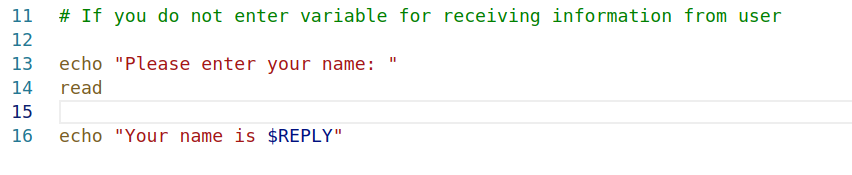
**2. Using Variables and Comments**

* Two types of variables
  + system variable (Capital case)
  + user defined variables

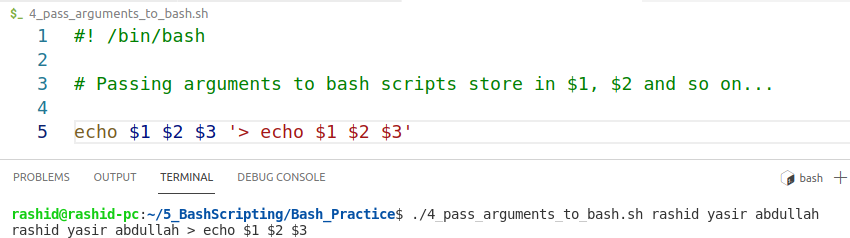


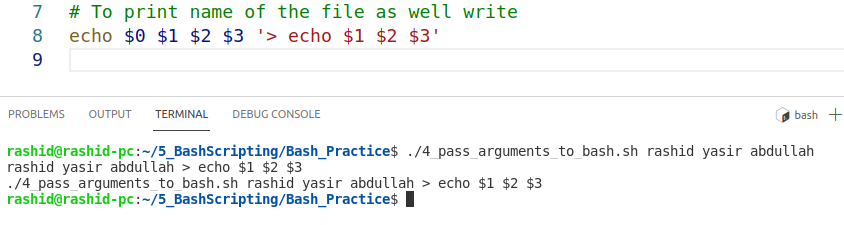
**3. Read User Input**

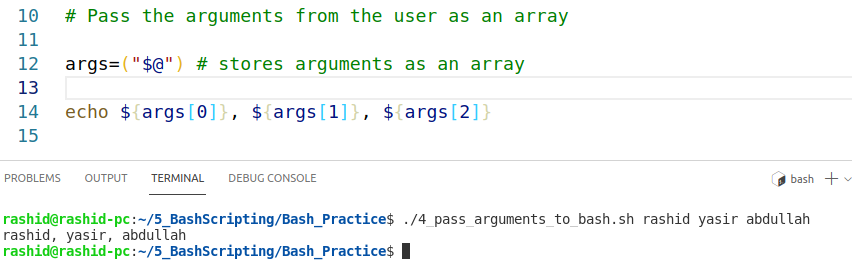


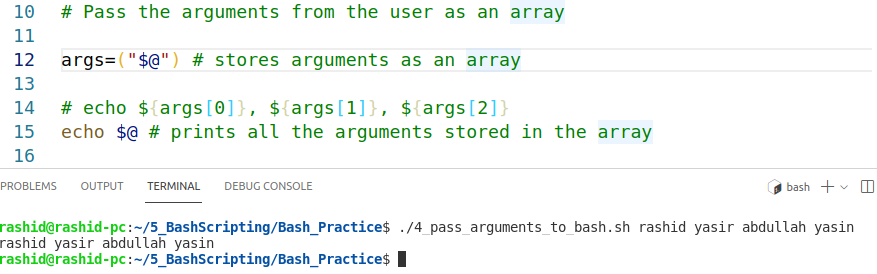


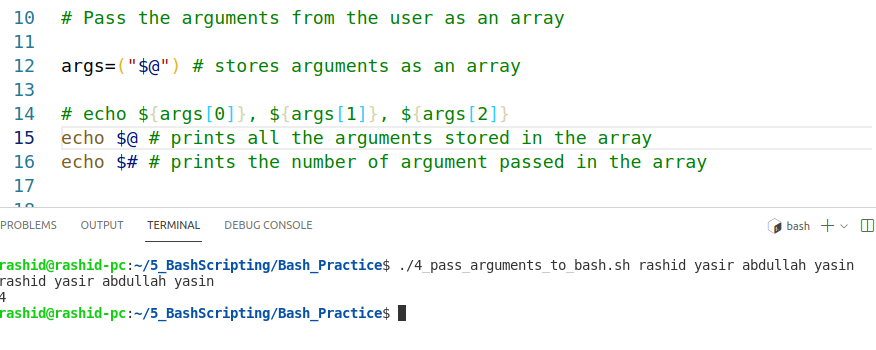
**4. Pass Arguments to a Bash Scripting**



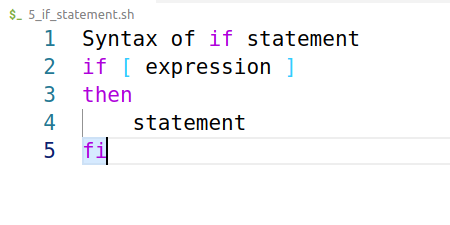








**5. Conditional Statements**



## Comparison Operators

### Integer Comparison

**-eq** is equal to **if [ "$a" -eq "$b" ]**

**-ne** is not equal to **if [ "$a" -ne "$b" ]**

**-gt** is greater than **if [ "$a" -gt "$b" ]**

**-ge** is greater than or equal to **if [ "$a" -ge "$b" ]**

**-lt** is less than **if [ "$a" -lt "$b" ]**

**-le** is less than or equal to **if [ "$a" -le "$b" ]**

**<** is less than (within [double parentheses](https://tldp.org/LDP/abs/html/dblparens.html)) **(("$a" < "$b"))**

**<=** is less than or equal to (within double parentheses) **(("$a" <= "$b"))**

**>** is greater than (within double parentheses) **(("$a" > "$b"))**

**>=** is greater than or equal to (within double parentheses) **(("$a" >= "$b"))**

### String Comparison

**=** is equal to **if [ "$a" = "$b" ]**

|  |  |
| --- | --- |
|  | Note the [whitespace](https://tldp.org/LDP/abs/html/special-chars.html" \l "WHITESPACEREF) framing the **=**.  **if [ "$a"="$b" ]** is not equivalent to the above. |

**==** is equal to **if [ "$a" == "$b" ]**

This is a synonym for =.

|  |  |  |
| --- | --- | --- |
|  | The == comparison operator behaves differently within a [double-brackets](https://tldp.org/LDP/abs/html/testconstructs.html" \l "DBLBRACKETS) test than within single brackets.   |  | | --- | | [[ $a == z\* ]] # True if $a starts with an "z" (pattern matching).  [[ $a == "z\*" ]] # True if $a is equal to z\* (literal matching).  [ $a == z\* ] # File globbing and word splitting take place.  [ "$a" == "z\*" ] # True if $a is equal to z\* (literal matching).  # Thanks, Stéphane Chazelas | |

**!=** is not equal to **if [ "$a" != "$b" ]**

This operator uses pattern matching within a [[[ ... ]]](https://tldp.org/LDP/abs/html/testconstructs.html" \l "DBLBRACKETS) construct.

**<** is less than, in [ASCII](https://tldp.org/LDP/abs/html/special-chars.html" \l "ASCIIDEF) alphabetical order

**if [[ "$a" < "$b" ]]**

**if [ "$a" \< "$b" ]**

Note that the "<" needs to be [escaped](https://tldp.org/LDP/abs/html/escapingsection.html" \l "ESCP) within a **[ ]** construct.

**>** is greater than, in ASCII alphabetical order

**if [[ "$a" > "$b" ]]**

**if [ "$a" \> "$b" ]**

Note that the ">" needs to be escaped within a **[ ]** construct.

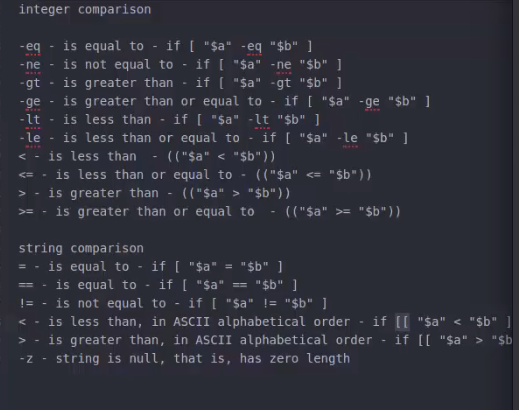
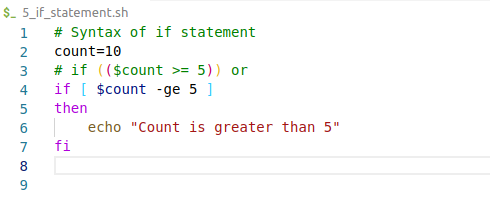
See [Example 27-11](https://tldp.org/LDP/abs/html/arrays.html" \l "BUBBLE) for an application of this comparison operator.

**-z** string is *null*, that is, has zero length

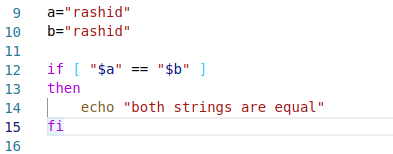
|  |
| --- |
| String='' # Zero-length ("null") string variable.  if [ -z "$String" ]  then  echo "\$String is null."  else  echo "\$String is NOT null."  fi # $String is null. |

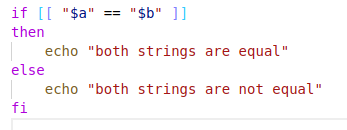
**-n** string is not *null.*

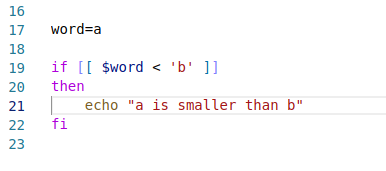
|  |  |
| --- | --- |
|  | The **-n** test requires that the string be quoted within the test brackets. Using an unquoted string with *! -z*, or even just the unquoted string alone within test brackets (see [Example 7-6](https://tldp.org/LDP/abs/html/comparison-ops.html" \l "STRTEST)) normally works, however, this is an unsafe practice. Always quote a tested string. |



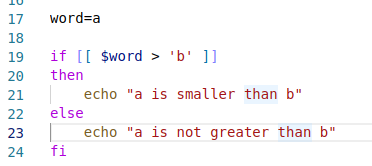
OR

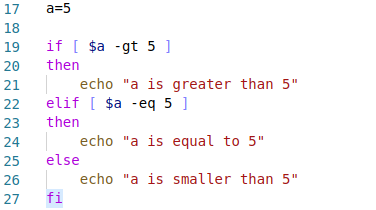




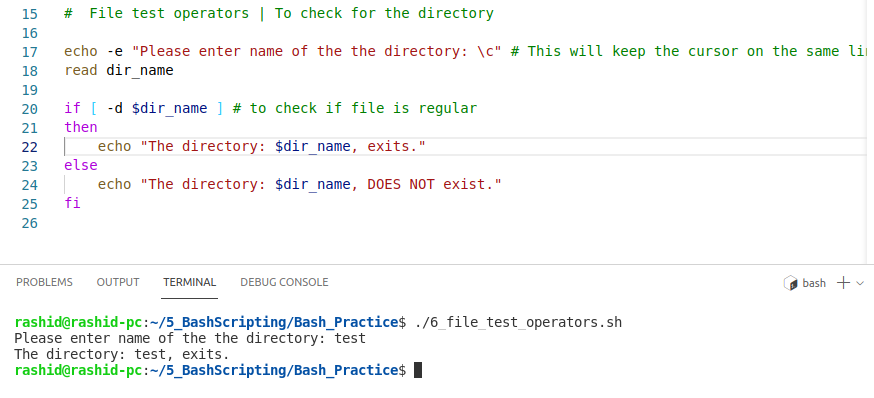
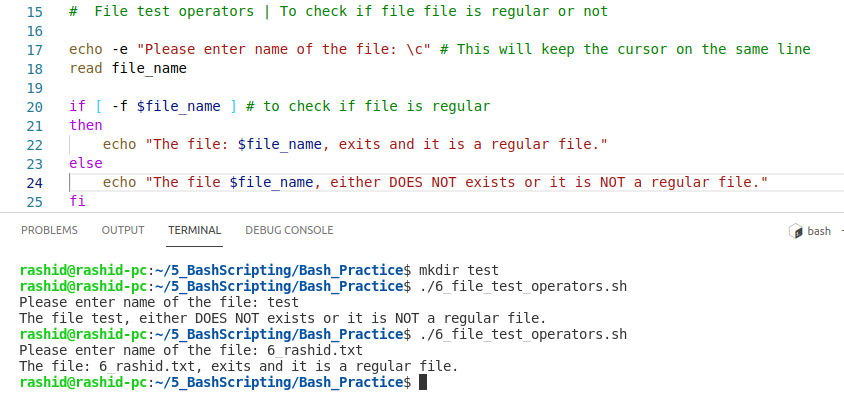
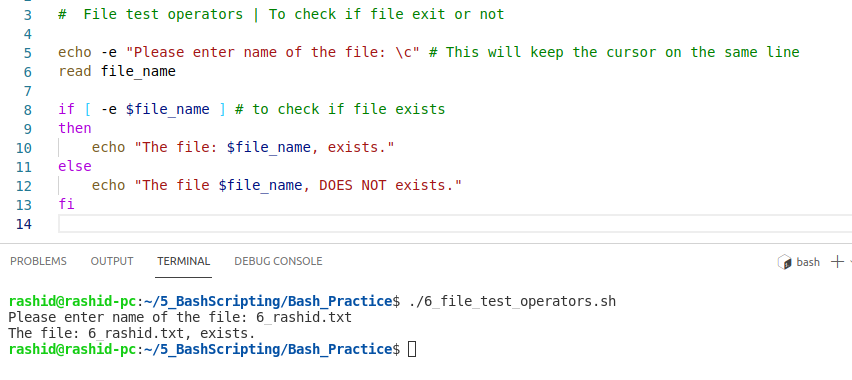


* use double brackets when you are using angle brackets (>,<) i.e., (( )) in case of comparing integers, and [[]] in case of comparing strings
* **Convention** that I will follow….
  + For strings, always use [[ ]] with comparison operators like ==, > etc
  + for integers, always use [] and use urinary operator like -eq, -gt etc.

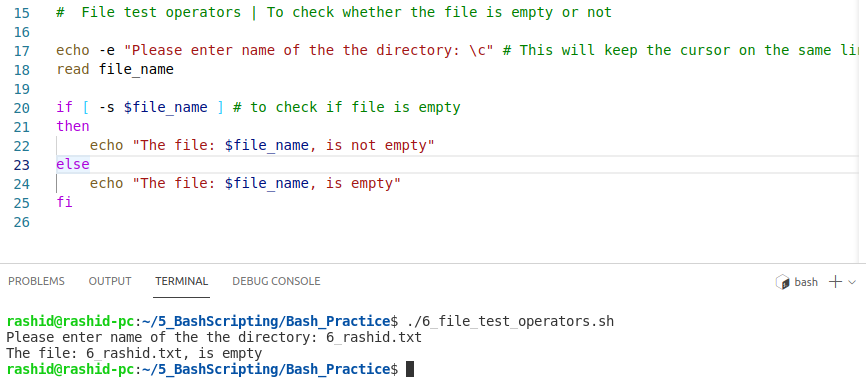




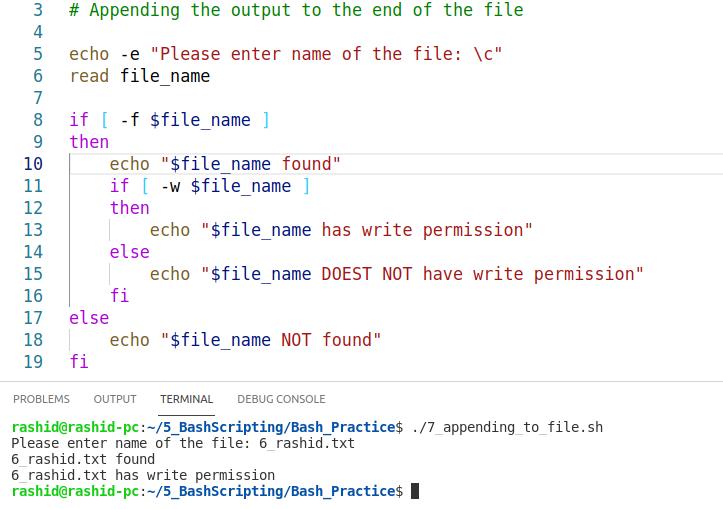
**6. File Test Operators**



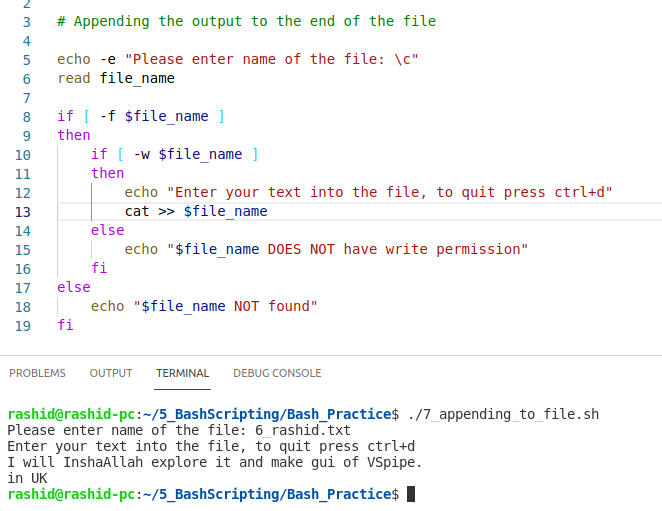
* Two types of files
  + character special file: normal file (**-b**)
  + block special file: binary file (**-c**)
* To check whether file is empty or not (**-s**)
* To check whether file has read, write, executable permission (**-r**, **-w, -x**)



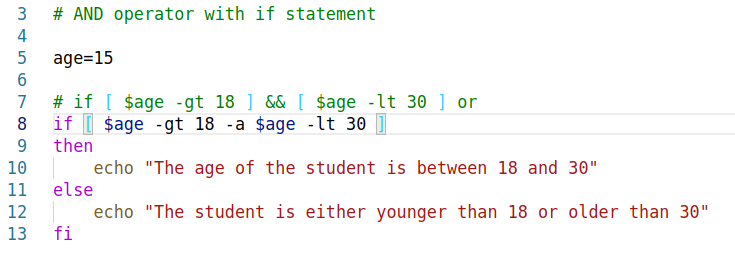
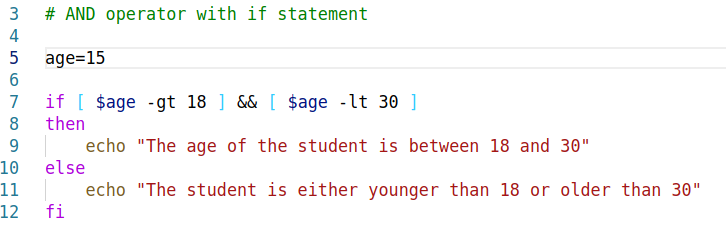
**7. How to append output of the end of the file**

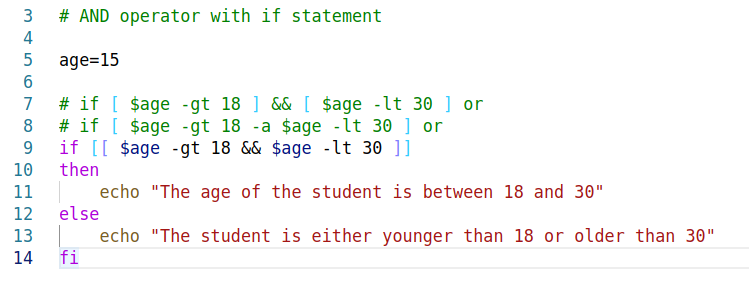


* cat > : will over-write the file
* cat >> : will append the data at the end of the file

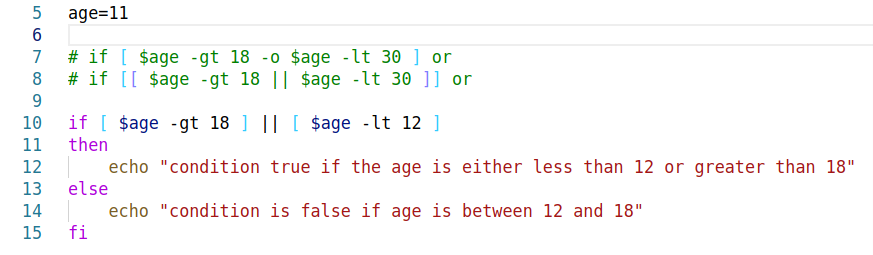


**8. Logical ‘AND’ Operator**

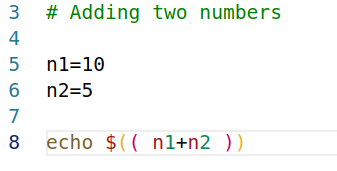


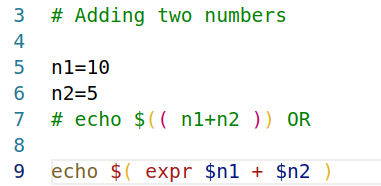


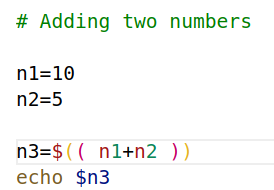
**9. Logical ‘OR’ Operator**

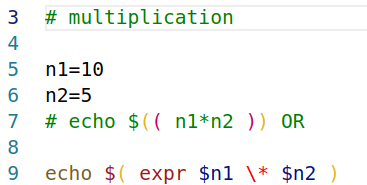


**10. Perform Arithmetic Operations**



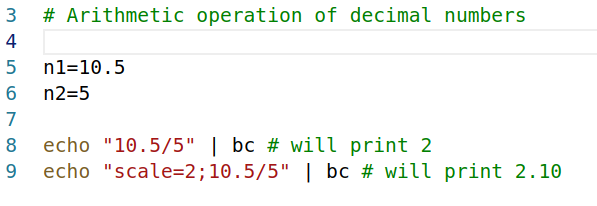


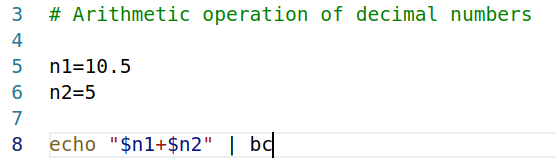


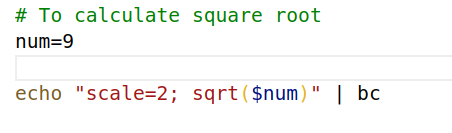


**11. Floating point math operations in bash | bc command**

* **echo "10.5+5" | bc**

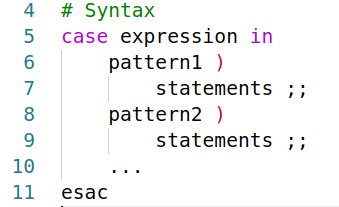
****

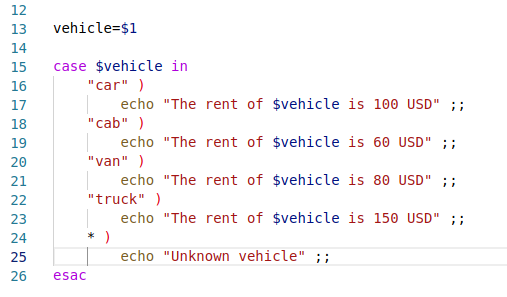


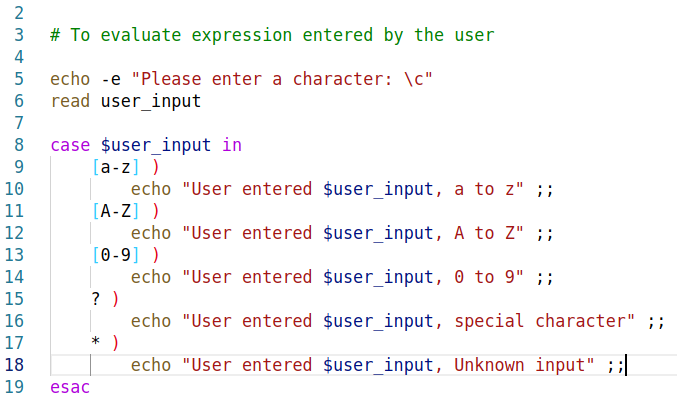


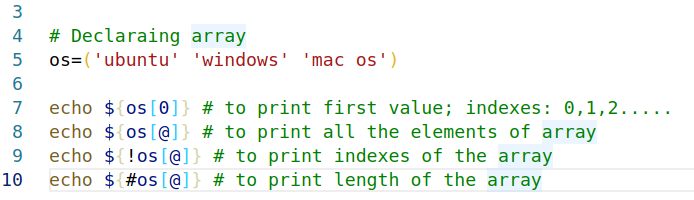
* echo "scale=2; 3^3" | bc

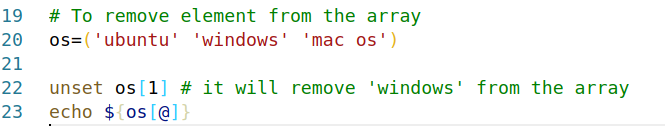
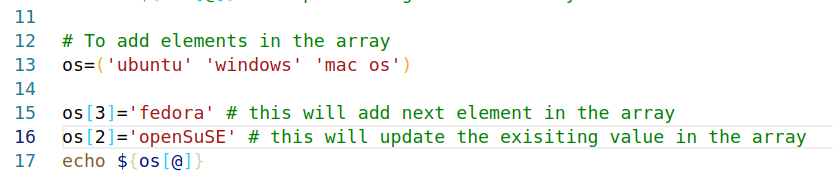
**12. The Case Statement / 13. The Case Statement Examples**

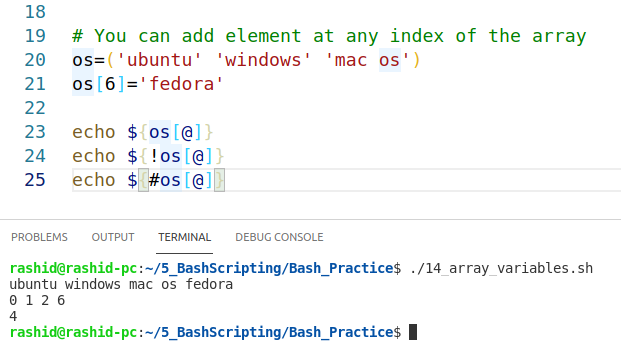
****

****

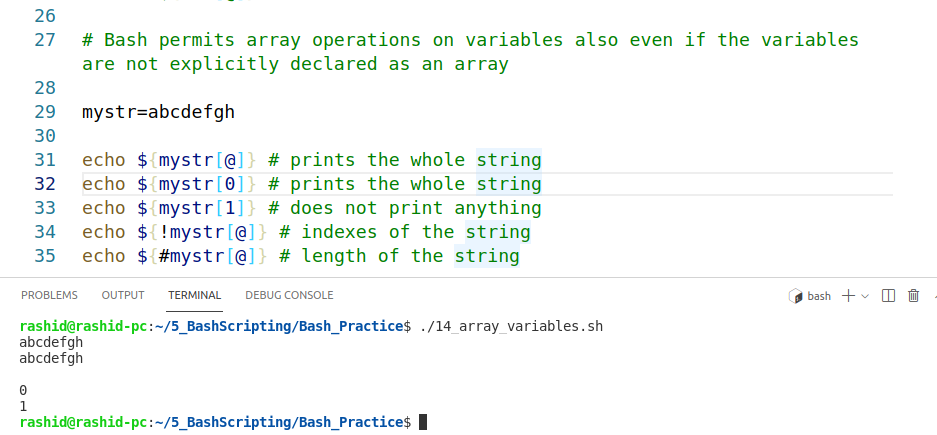
**14. Array Variables**

****

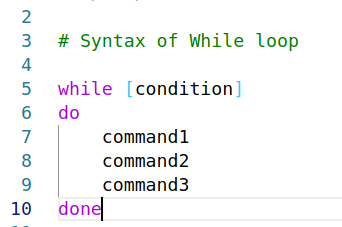


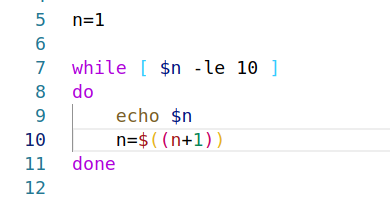
****

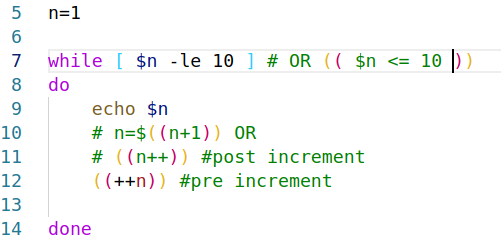
* Above code means that some indexes in the array may be left UN-initialized and gaps in the array are OK.



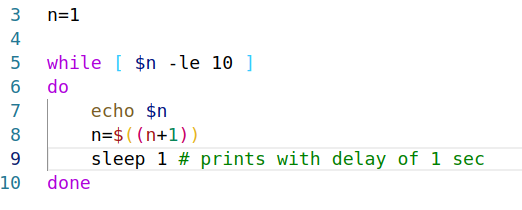
**15. WHILE Loops**

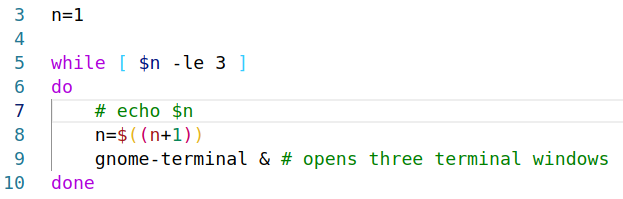
****

****

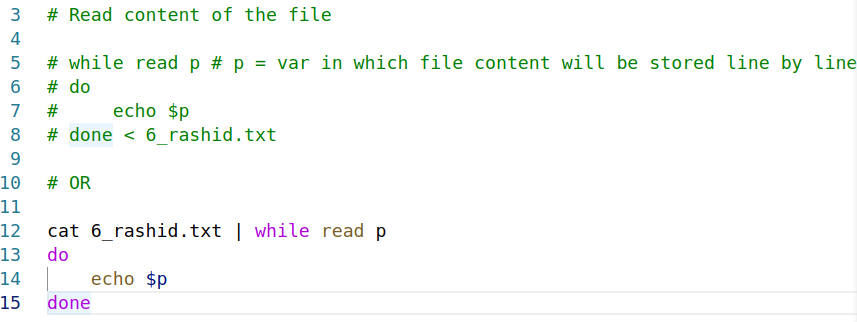
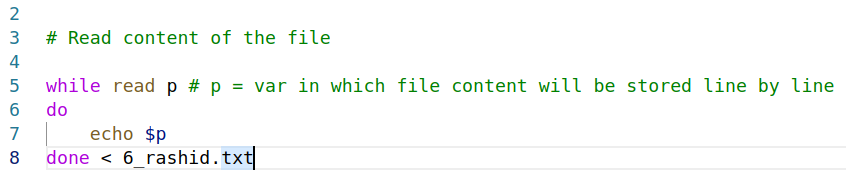
****

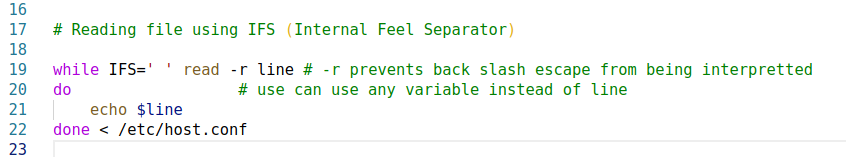
**16. Using sleep and open terminal with WHILE Loops**

****

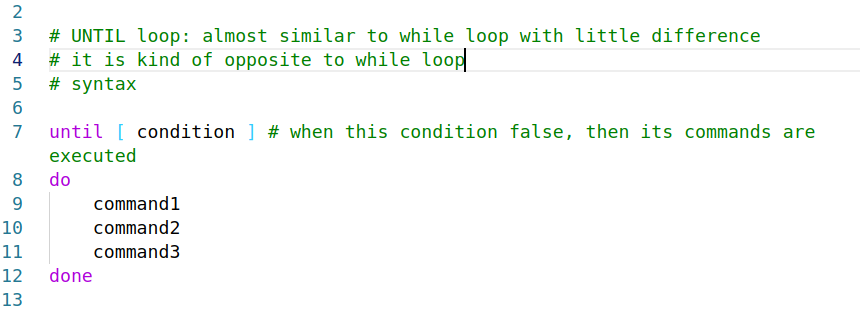
****

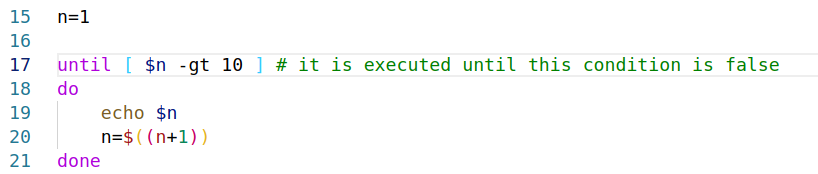
**17. Read a file content in bash**

****

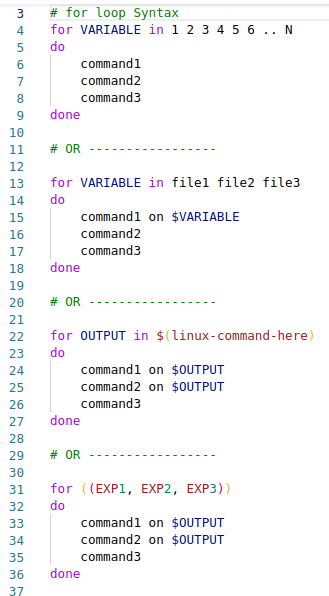
****

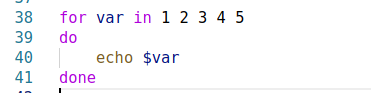
**18. UNTIL loop**

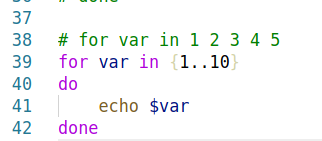
****

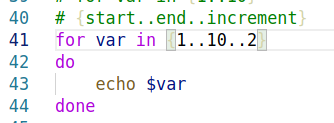
****

**19. FOR loop**

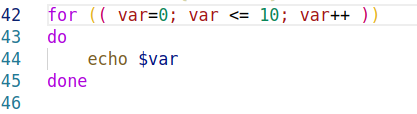
****

****

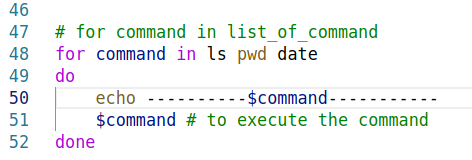
****

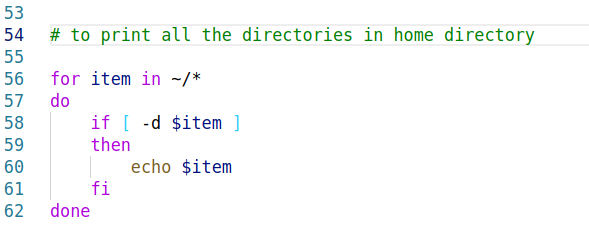
****

* echo ${BASH\_VERSION}

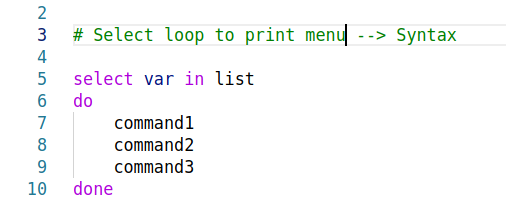
****

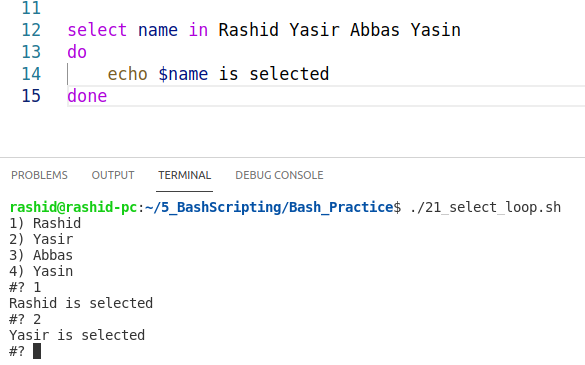
**20. FOR loop to execute commands**

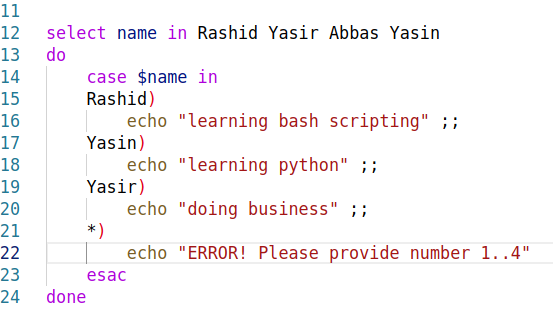
****

****

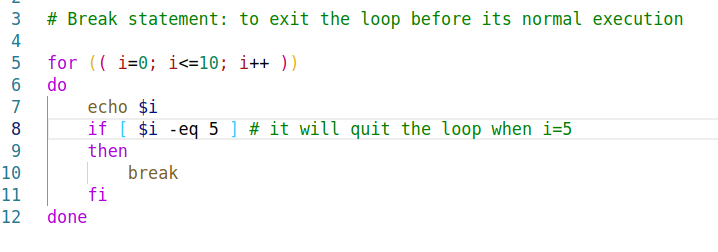
**21. Select Loop**

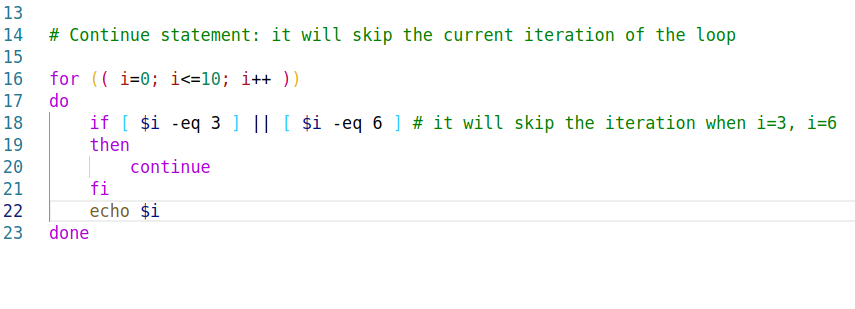
****

****

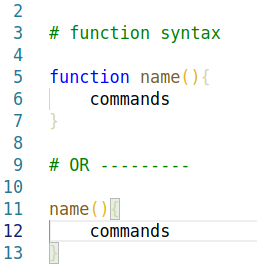
****

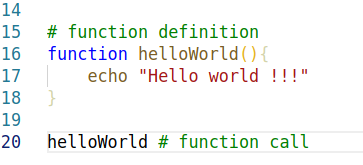
**22. Break and continue statement**

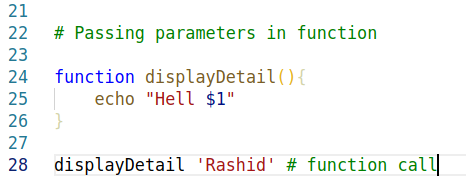
****

****

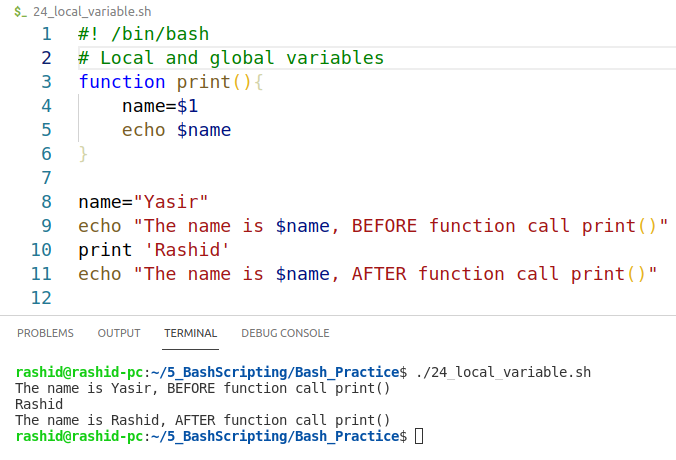
**23. Functions**

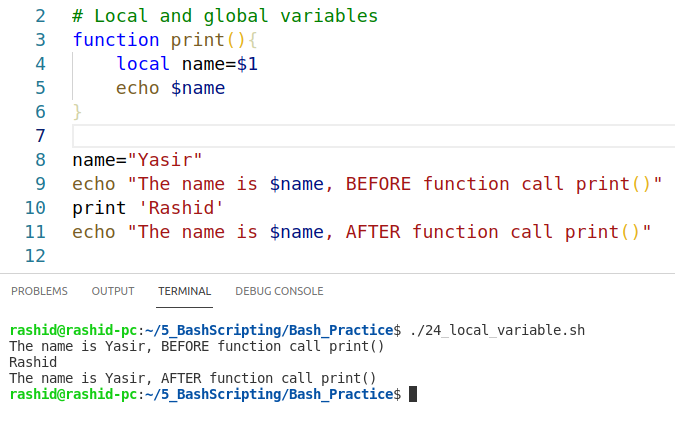
****

****

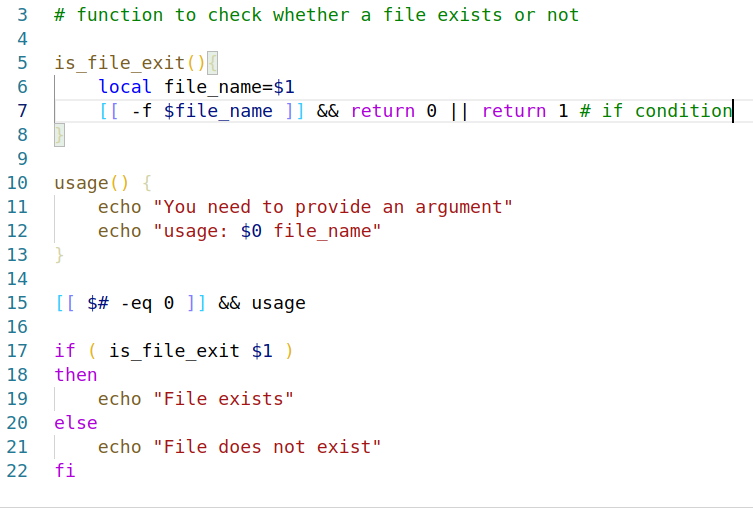
****

**24. Local Variables**

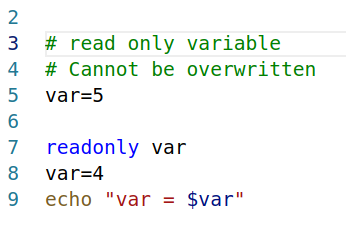
****

****

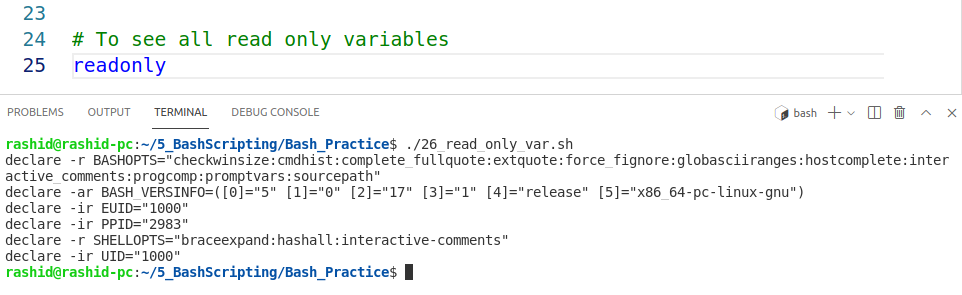
**25. Function Examples**

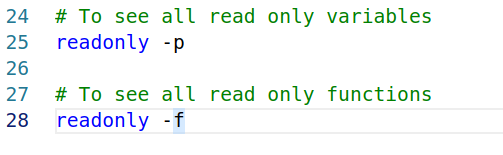
****

**26. Read only command**

****

****

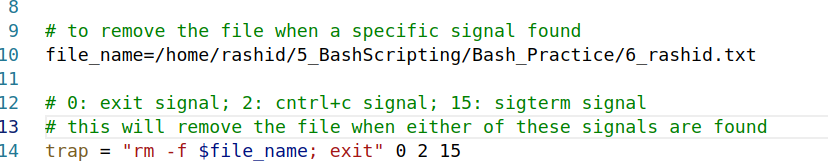
****

****

**27. Signals & Trap**

* To kill the process
  + kill -9 pid
* '$$' to print pid of the script itself
* to check about different signals
* man 7 signal
* simicolon is used to combine two commands



****

* to see trap that you have defined
* type trap
* to remove trap
  + trap - name\_of signal/number\_of\_signal e.g. trap – 0 2 15

**28. Debug a bash script**

* bash -x ./script\_name.sh OR
* #! /*bin/*bash -x (in the script) OR
* type set -x (in the script), it will start debugging from the point where you define this
* type set +x (in the script), it will deactivatedebugging from the point where you define this