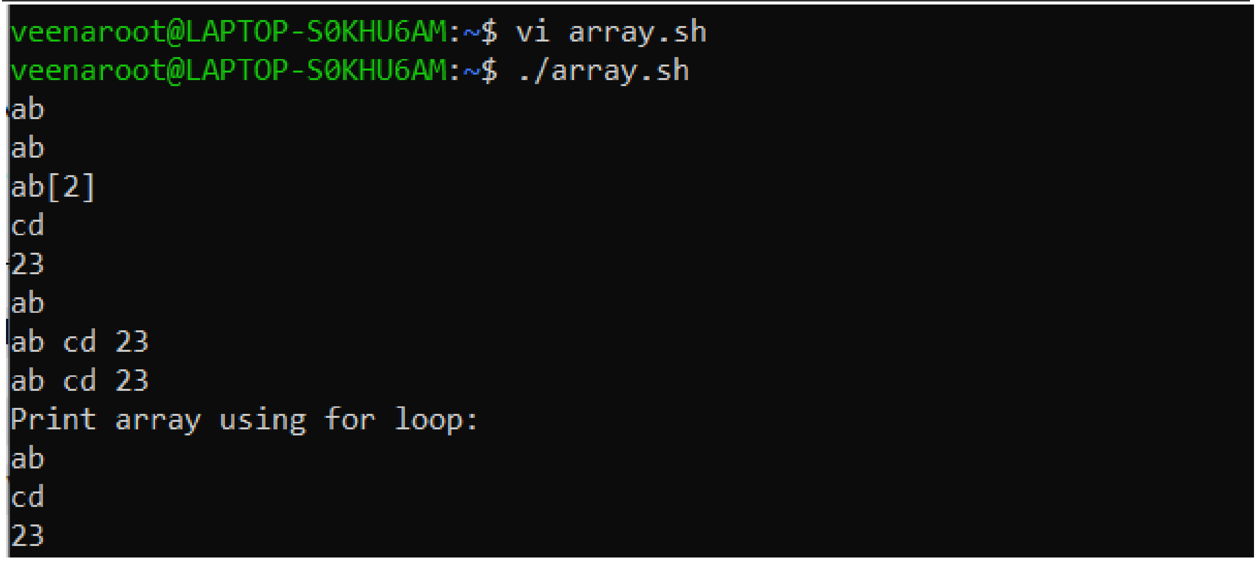
Day - 8 SRE Training

Topic - Shell Scripting

Arrays

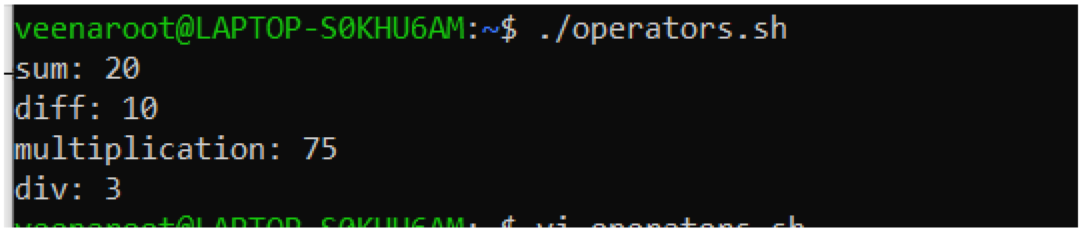
```
a[0]="ab" #declaring array variables
a[1]="cd"
a[2]=23
echo $a   #displays a[0]
echo "$a"
echo "$a[2]"
echo "${a[1]}"  #displays a[1]
echo "${a[2]}"
echo "${a}"
echo "${a[@]}"  #displays entire array
echo "${a[*]}"  #displays entire array

echo "Print array using for loop: "
for i in "${a[@]}"
do
        echo "$i"
done
```



```
veenaroot@LAPTOP-S0KHU6AM:~$ vi array.sh
veenaroot@LAPTOP-S0KHU6AM:~$ ./array.sh
ab
ab
ab[2]
cd
23
ab
ab cd 23
ab cd 23
Print array using for loop:
ab
cd
23
```

$((expression)) is used for arithmetic calculations in shell scripting.
+, -, *, / are standard operators.
Integer division returns an **integer** (e.g., 17/5 would give 3, not 3.4).

```
x=15
y=5
echo "sum: $((x+y))"
echo "diff: $((x-y))"
echo "multiplication: $((x*y))"
echo "div: $((x/y))"
```



```
veenaroot@LAPTOP-S0KHU6AM:~$ ./operators.sh
sum: 20
diff: 10
multiplication: 75
div: 3
veenaroot@LAPTOP-S0KHU6AM:~$ vi operators.sh
```

-ge (Greater than or equal to) → Checks if the left operand is greater than or equal to the right. Example: [$a -ge $b]

-gt (Greater than) → Checks if the left operand is strictly greater than the right. Example: [$a -gt $b]

-le (Less than or equal to) → Checks if the left operand is less than or equal to the right. Example: [$a -le $b]

-lt (Less than) → Checks if the left operand is strictly less than the right. Example: [$a -lt $b]

-eq (Equal to) → Checks if two values are equal. Example: [$a -eq $b]

-ne (Not equal to) → Checks if two values are not equal. Example: [$a -ne $b]

-o or || (Logical OR) → Returns true if at least one condition is met. Example: [$a -gt 5 -o $b -lt 3]

-a or && (Logical AND) → Returns true only if both conditions are met. Example: [$a -gt 5 -a $b -lt 3]

= (String comparison for equality) → Checks if two strings are equal. Example: ["$str1" = "$str2"]

-z (String is empty) → Checks if a string is empty. Example: `[ -z "$str" ]`

```
balance=1400
min_balance=500
withdrawal=600
daily_limit=1000
account_type="savings"
description=""
if [ $balance -ge $min_balance ]; then
        echo "Minimum balance is maintained"
fi

if [ $min_balance -eq 500 ]; then
        echo "Minimum balance is maintained to 500"
fi

if [ $min_balance -ne 500 ]; then
        echo "Minimum balance is not 500"
fi

if [ $balance -gt $withdrawal ]; then
        balance=$((balance - withdrawal))
        echo "Withdrawal successful, balance = $balance"
fi

if [ $withdrawal -le $balance -a $withdrawal -le $daily_limit ]; then
          echo "Transaction approved"
  else
            echo "transaction not approved"
fi

if [ $withdrawal -le $balance -o $balance -ge 500 ]; then
           echo "Account minimum balance maintained and transaction is
approved"
fi

if [ "$account_type" = "savings" ]; then
           echo "These is a saving account"
fi

if [ -z "$description" ]; then
           echo "description is not provided"
fi
```

-s (File is not empty) → Checks if `new.txt` exists and is not empty.
Example: `[ -s "$file1" ]`

-e (File exists) → Checks if new.txt exists. Example: [-e "$file1"]

-r (Readable file) → Checks if new.txt has read permission. Example: [-r "$file1"]

-w (Writable file) → Checks if new.txt has write permission. Example: [-w "$file1"]

-x (Executable file) → Checks if new.txt has execute permission. Example: [-x "$file1"]

```
file1="new.txt"
if [ -s "$file1" ]; then
        echo "new exists and is not empty"
fi
if [ -e "$file1" ]; then
                echo "new exists"
fi
if [ -r "$file1" ]; then
            echo "$file1 has read permission"
fi

if [ -w "$file1" ]; then
            echo "$file1 has write permission"
fi

if [ -x "$file1" ]; then
            echo "$file1 has execute permission"
fi
```

`read name` → Reads a user's input (`name`) and stores it.

`read -t 5 -p "" pin` → Prompts for input (`pin`) with a **5-second timeout**. If no input is given within 5 seconds, it moves on.

`read -p ""` → Prompts for any no. of arguments in a **single line**.

`read -s -p "Enter password" p` → Prompts for a **silent input (`p`)**, meaning the password is **not displayed** while typing.

```
read -t 5 -p "quick 5 sec" pin
```

```
echo "Enter your name"
read name
echo "$name"

read -p " Enter account number and password:" accno password
echo $accno
echo $password
#echo "enter sensitive password"
read -s -p " Enter password" p
```

read -p "Enter selection [1-3]" selection → Prompts the user to enter a selection and stores it in the variable selection.

case $selection in → Starts a **case statement** to handle different input values.

1) → If the user enters 1, sets accounttype="checking" and prints "you have selected checking".

***)** → The **default case** (for invalid input) sets accountype="random" and prints "random selection".

```
read -p "Enter selection [1-3]" selection
case $selection in
          1) accounttype="checking"; echo " you have selected checking";;
          2) accountype="saving"; echo "you have selected saving";;
          3) accounttype="current"; echo " you have selected current";;
          *) accounttype="random"; echo "random selection";;
esac
```

grep "pattern" filename → Searches for the exact "pattern" in filename.

grep "a.b" filename" → Matches any three-character sequence where a is the first character, b is the last, and any character (.) is in between (e.g., acb, axb).

grep "abc.d" filename" → Matches abc followed by any character and then d (e.g., abc1d, abcXd).

grep "a.*b" filename" → Matches lines where a appears first and b appears later, with anything (.*) in between (e.g., alphabet, anb).

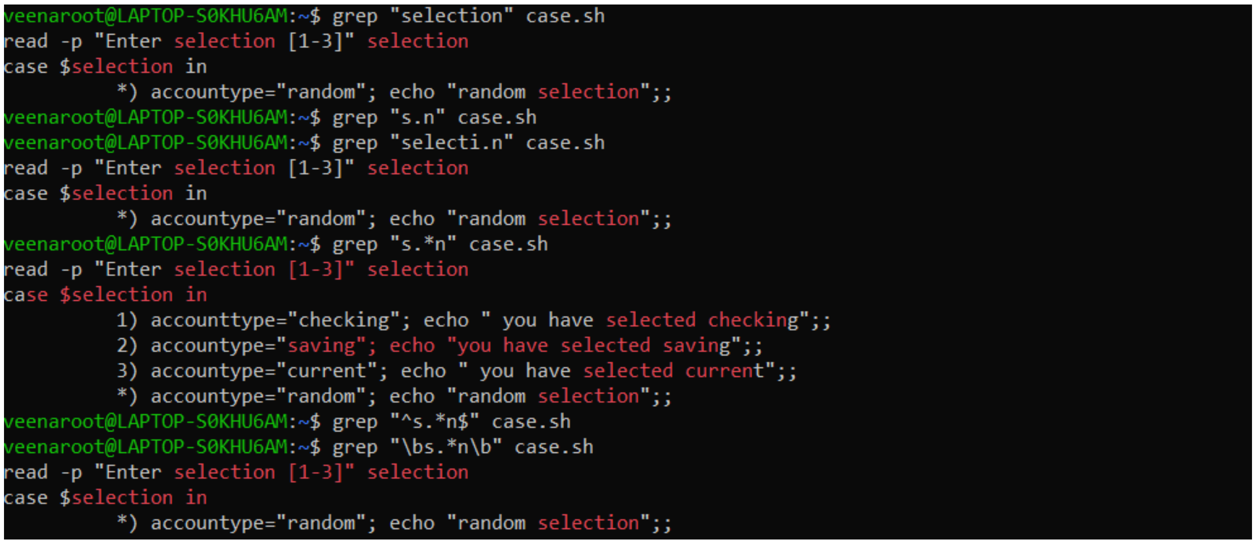
grep "^a.*b$" filename" → Matches lines that start with a and end with b (e.g., alphabet, but not anb word).

grep "\ba.*b\b" filename" → Matches words that start with a and end with b, ensuring word boundaries (\b).

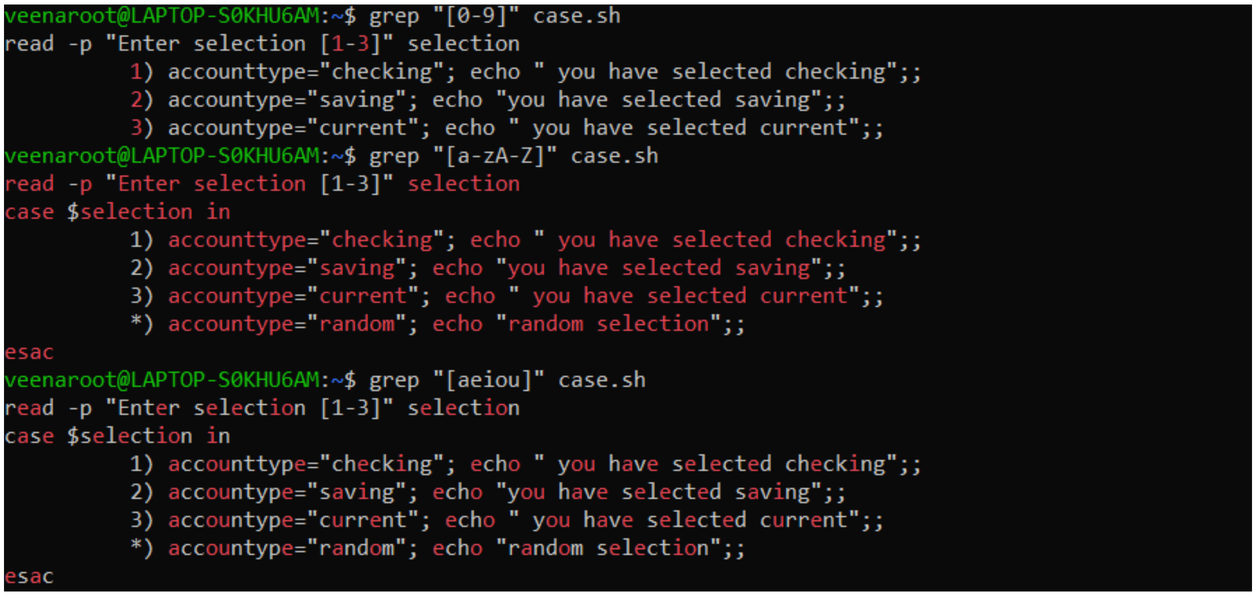
grep "[0-9]" filename" → Matches any single digit (0-9).

grep "[a-zA-Z]" filename" → Matches any single uppercase or lowercase letter.

grep "[aeiou]" filename" → Matches any single vowel (a, e, i, o, or u).



```
veenaroot@LAPTOP-S0KHU6AM:~$ grep "selection" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          *) accountype="random"; echo "random selection";;
veenaroot@LAPTOP-S0KHU6AM:~$ grep "s.n" case.sh
veenaroot@LAPTOP-S0KHU6AM:~$ grep "selecti.n" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          *) accountype="random"; echo "random selection";;
veenaroot@LAPTOP-S0KHU6AM:~$ grep "s.*n" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          1) accounttype="checking"; echo " you have selected checking";;
          2) accountype="saving"; echo "you have selected saving";;
          3) accountype="current"; echo " you have selected current";;
          *) accountype="random"; echo "random selection";;
veenaroot@LAPTOP-S0KHU6AM:~$ grep "^s.*n$" case.sh
veenaroot@LAPTOP-S0KHU6AM:~$ grep "\bs.*n\b" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          *) accountype="random"; echo "random selection";;
```



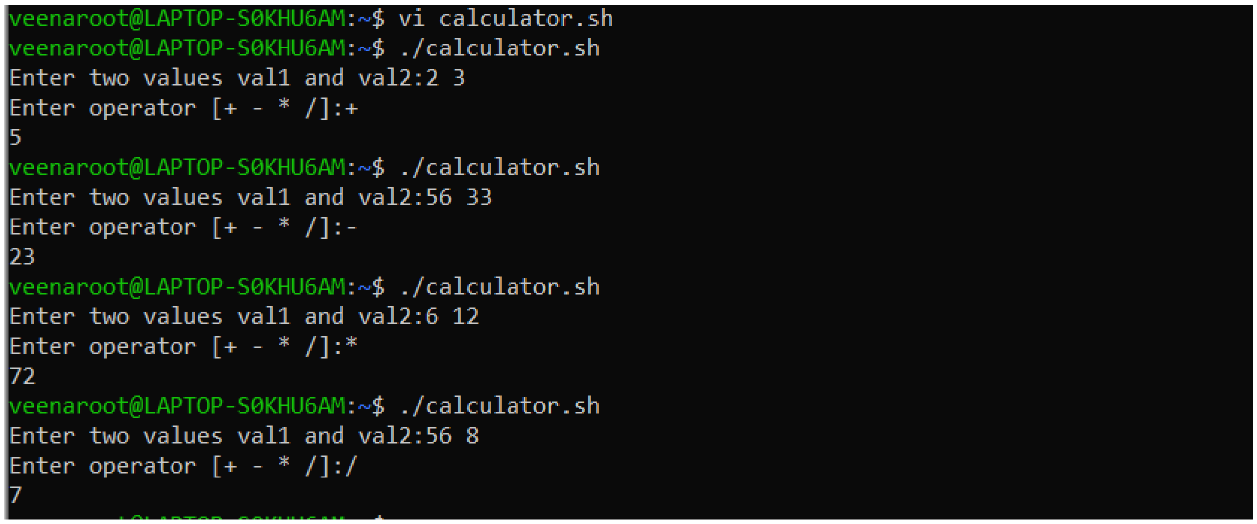
```
veenaroot@LAPTOP-S0KHU6AM:~$ grep "[0-9]" case.sh
read -p "Enter selection [1-3]" selection
          1) accounttype="checking"; echo " you have selected checking";;
          2) accountype="saving"; echo "you have selected saving";;
          3) accountype="current"; echo " you have selected current";;
veenaroot@LAPTOP-S0KHU6AM:~$ grep "[a-zA-Z]" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          1) accounttype="checking"; echo " you have selected checking";;
          2) accountype="saving"; echo "you have selected saving";;
          3) accountype="current"; echo " you have selected current";;
          *) accountype="random"; echo "random selection";;
esac
veenaroot@LAPTOP-S0KHU6AM:~$ grep "[aeiou]" case.sh
read -p "Enter selection [1-3]" selection
case $selection in
          1) accounttype="checking"; echo " you have selected checking";;
          2) accountype="saving"; echo "you have selected saving";;
          3) accountype="current"; echo " you have selected current";;
          *) accountype="random"; echo "random selection";;
esac
```

Simple Calculator

This script takes two values (`val1` and `val2`) and an operator (`+`, `-`, `*`, or `/`) as input, then performs the corresponding arithmetic operation using `$(( ))`. The `case` statement matches the operator and executes the respective calculation.

```
read -p "Enter two values val1 and val2:" val1 val2

read -p "Enter operator [+ - * /]:" op

case $op in
        +) echo $((val1 + val2));;
        -) echo $((val1 - val2));;
        \*) echo $((val1 * val2));;
        /) echo $((val1 / val2));;
        *) echo "Enter a operand which is + - * /";;
esac
```

The `\*` is used instead of `*` because `*` is a special character in shell scripting (used for wildcard expansion), so we escape it with a backslash (`\`) to ensure it is treated as a literal multiplication operator.

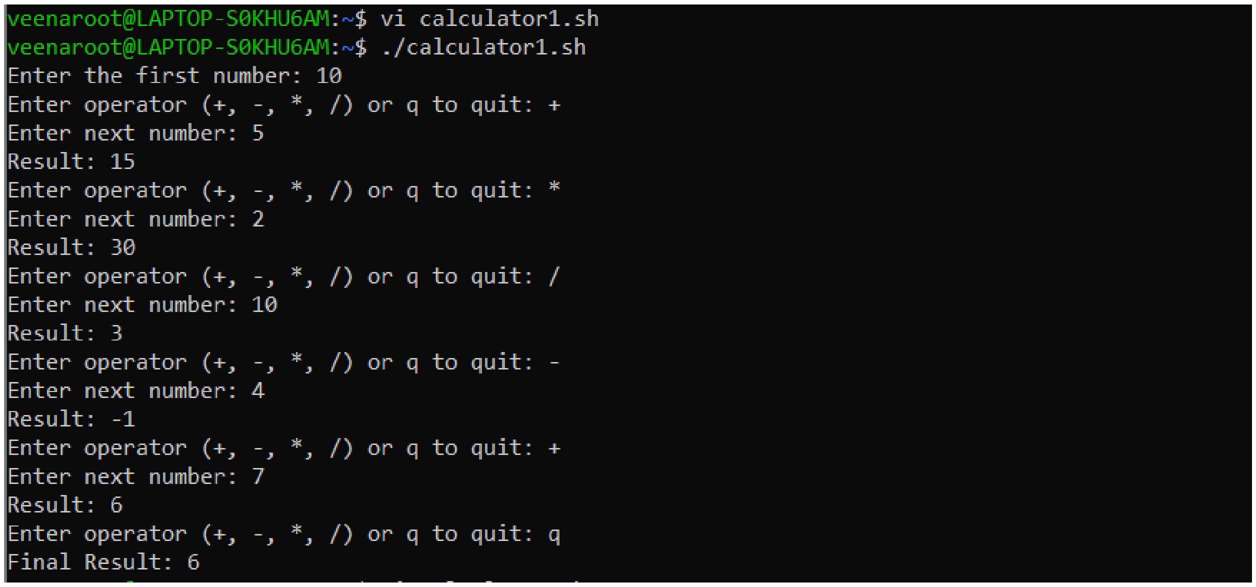


```
veenaroot@LAPTOP-S0KHU6AM:~$ vi calculator.sh
veenaroot@LAPTOP-S0KHU6AM:~$ ./calculator.sh
Enter two values val1 and val2:2 3
Enter operator [+ - * /]:+
5
veenaroot@LAPTOP-S0KHU6AM:~$ ./calculator.sh
Enter two values val1 and val2:56 33
Enter operator [+ - * /]:-
23
veenaroot@LAPTOP-S0KHU6AM:~$ ./calculator.sh
Enter two values val1 and val2:6 12
Enter operator [+ - * /]:*
72
veenaroot@LAPTOP-S0KHU6AM:~$ ./calculator.sh
Enter two values val1 and val2:56 8
Enter operator [+ - * /]:/
7
```

Calculator taking multiple operands:

It first takes an initial number (`result`) as input. Then, inside a `while` loop, it repeatedly asks for an operator (`+`, `-`, `*`, `/`) and another number. Based on the operator, it updates `result` using `$(( ))` for calculations. If the user enters `q`, the loop breaks, and the final result is displayed. The script also handles division by zero by displaying an error message and skipping the operation.

```
read -p "Enter the first number: " result

while true; do
    read -p "Enter operator (+, -, *, /) or q to quit: " op
    if [ "$op" = "q" ]; then
        break
    fi
    read -p "Enter next number: " num

    case $op in
        +) result=$((result + num)) ;;
        -) result=$((result - num)) ;;
        \*) result=$((result * num)) ;;
        /)
            if [ "$num" -eq 0 ]; then
                echo "Error: Division by zero!"
                continue
            else
                result=$((result / num))
            fi
            ;;
        *) echo "Invalid operator"; continue ;;
    esac

    echo "Result: $result"
done

echo "Final Result: $result"
```



```
veenaroot@LAPTOP-S0KHU6AM:~$ vi calculator1.sh
veenaroot@LAPTOP-S0KHU6AM:~$ ./calculator1.sh
Enter the first number: 10
Enter operator (+, -, *, /) or q to quit: +
Enter next number: 5
Result: 15
Enter operator (+, -, *, /) or q to quit: *
Enter next number: 2
Result: 30
Enter operator (+, -, *, /) or q to quit: /
Enter next number: 10
Result: 3
Enter operator (+, -, *, /) or q to quit: -
Enter next number: 4
Result: -1
Enter operator (+, -, *, /) or q to quit: +
Enter next number: 7
Result: 6
Enter operator (+, -, *, /) or q to quit: q
Final Result: 6
```