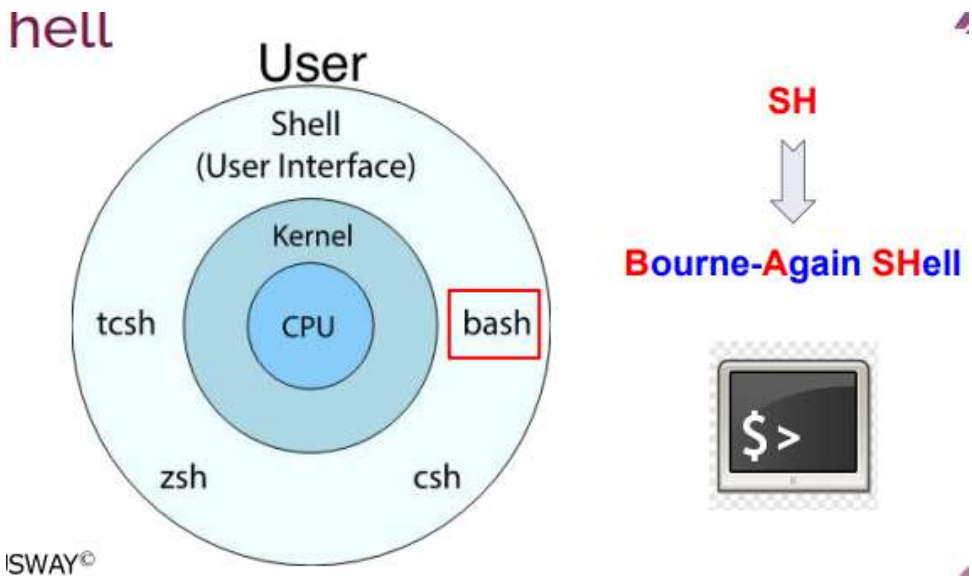
İlk aşamada yazmaktan ziyade başkaları tarafından yazılmış scriptleri okumamız tavsiye olunur. Kod öğrenmede bu önemli bir husus. Bunları Notion da not olarak tutup ileride kullanabiliriz de.




Shell Kernel vasıtasıyla bilgisayar donanımıyla iletişim kurmamızı sağlayan ara katmandır. Interface ile yapabildiğimizden çok daha fazlasını bash ile yapabiliriz.

Bizim DevOps çu olarak birden fazla resoruce u birbiriyle ilişkili şekilde oluşturmamız istenecek. Bunları da yapmanın yöntemlerinden birisi shell script tir. AWS deki her butonun shell de bir karşılığı var.

Texti ekrana yazdırmak, AWS kaynağı oluşturmak, o kaynaktan veri çekip başka bir dosyay yazdırmak, log almaki gecenin bir yarısı scripti devreye alıp belli bir logu birilerine mail atmak gibi bir çok alanda işlem yapılabilir. Başkasının klavye ve mouse ile yaptığı tüm işlemleri shell scripting işlemi ile yapabiliriz.

**What is Shell Scripting?**

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell which could be one of the following:

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

Shell scriptin interpreter leri var. Derleyici demektir. Bu interpreterler ile shell dosyaları derlenir. Bu shell ler ile shell scripting yazıp uygulayabiliyoruz.

DevOps ta bir ya da iki dilde script yazma ya da mevcut scripti düzenleyebilme yeteneği istenir.


Shell scripting bir dosyadır. Peki bunun PEP8 gibi kuralları var mıdır? Vardır:

Genelde shell scriptleri .sh uzantısıyla oluştururuz.

Hangi interpreter ile derleneceğini belirtmek için #! /shebang) yazdıktan sonra boşluk bırakmadan interpreter yazarız.
Örneğin; #!/bin/bash

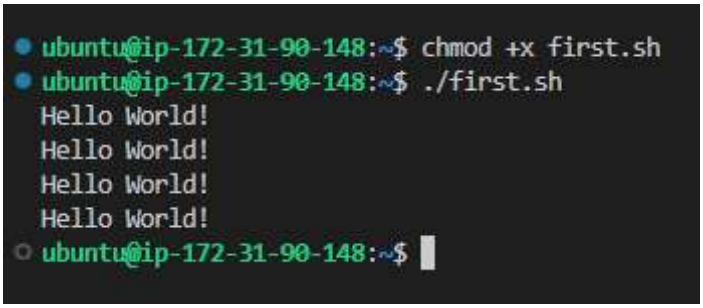
Normalde bir programlama dilinde çalışırken, o pc içinde o dilin interpreteri olması gerekir.


Sonrasında istediğimiz komutları yazıp kaydedip ıktıktan sonra shell scriptin koşulabilmesi için ona chmod komutuyla +x diyerek execute yetkisi vermemiz gerekiyor.

./filename.sh yazdığımızda dosya koşulur.

İlk scriptimizi yazalım. First.sh ismiyle doysamızı oluşturup içine komutları yazıyoruz:


Sonra komut satırında chmod +x ile execute yetkisi veriyoruz ve komutu koşuyoruz:

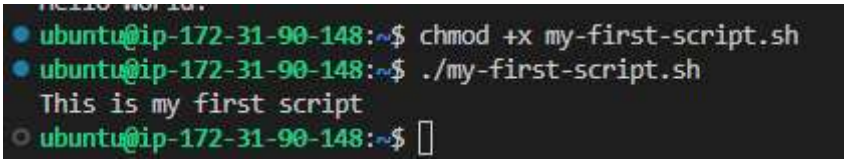



Exercise:

1. Create a script named: "**my-first-script.sh**"

   It should print: "**This is my first script.**"
2. Make the script executable.
3. Execute the script.

Sonucu:




Script dosyalarımızın komut olarak ./ eklemeden çalışması için bulundukları klasörü PATH eklememiz gerekiyordu. Ekleyelim ve deneyelim:

Variable tanımlamak script yazarken işimizi çok kolaylaştıracaktır.

- A variable is pointer to the actual data. The shell enables us to create, assign, and delete variables.
- The name of a variable can contain only letters (a to z or A to Z), numbers ( 0 to 9) or the underscore character (_) and beginning with a letter or underscore character.
- The reason you cannot use other characters such as !, *, or - is that these characters have **a special meaning for the shell.**

```
$VARIABLE=value
$echo $VARIABLE
value
$
$my_var=my_value
$echo $my_var
my_value
$
$my-var=my-value
my-var=my-value: command not found
$
$myvar?=my-value
myvar?=my-value: command not found
```

Variable tanımlarken ismi büyük harfle yapacaz, baş karakter olmamak şartıyla sayıları kullanabiliyorduk ve özel karakter olarak sadece _ kullanabiliyorduk.

Environment variable ı script içinde kullanabiliyorduk. Görelim:
Scriptimi güncelliyorum:

```
$ my-first-script.sh
1   #!/bin/bash
2   NAME="Emre"
3   echo "Hello $NAME"
```

NAME variable ı Emre olarak value atıyorum ve kodu çalıştırıyorum:

```
ubuntu@ip-172-31-90-148:~$ export NAME="Emre"
ubuntu@ip-172-31-90-148:~$ my-first-script.sh
Hello Emre
ubuntu@ip-172-31-90-148:~$
```

Şimdi kullanıcıya değer veriderecek bir script yazalım. İsmini girsin ve girince Hello NAME yazdırsın:

```
$ my-first-script.sh
1   #!/bin/bash
2   echo "Please enter your name: "
3
4   read NAME
5
6   echo "Hello $NAME"
```

```
ubuntu@ip-172-31-90-148:~$ my-first-script.sh
Please enter your name:
Emre
Hello Emre
```

read komutuna -p eklersek echo yu scriptten kaldırabiliriz:

```
#!/bin/bash
read -p "Please enter your name: " NAME

echo "Hello $NAME"
```

-sp eklersek yazılan veriyi gizler (örneğin password istediğimiz için görünmemesini isteriz.)
doğru sırası
-sp olacak. -ps olmaz.

Aşağıdakiler default variable dır kullanılamaz:

**$0** - The name of the Bash script.
**$1 - $9** - The first 9 arguments to the Bash script.
**$#** - How many arguments were passed to the Bash script.
**$@** - All the arguments supplied to the Bash script.
**$?** - The exit status of the most recently run process.
**$$** - The process ID of the current script.
**$USER** - The username of the user running the script.
**$HOSTNAME** - The hostname of the machine the script is running on.
**$SECONDS** - The number of seconds since the script was started.
**$RANDOM** - Returns a different random number each time is it referred to.
**$LINENO** - Returns the current line number in the Bash script.


Dosyaya inputu komut satırında verirken;
$0 dosya ismini tutar, $1-9 sırasıyla argümanları tutar.
$# kaç tane argüman olduğunu tutar.
$@ bütün argümanları liste olarak tutar.

my-first-script.sh

```
#!/bin/bash

echo "script name is $0"

echo " first argument is $1"

echo " second argument is $2"

echo "$# arguments have been passed to scrpit"

echo " we have those arguments: $@"
```

```
ubuntu@ip-172-31-86-222:~$ my-first-script.sh first_argument second_argument
script name is /home/ubuntu/my-first-script.sh
 first argument is first_argument
 second argument is second_argument
2 arguments have been passed to scrpit
 we have those arguments: first_argument second_argument
```

$? Önceki process i kontrol etmek için kullanırız. Eğer bu değer 0 sa önceki process başarılıdır. Eğer none zero exit verdiyse önceki process başarısızdır.
$$ process ID sini tutar.

my-first-script.sh

```
1   #!/bin/bash
2
3   echo "script name is $0"
4
5   echo " first argument is $1"
6
7   echo " second argument is $2"
8
9   echo "$# arguments have been passed to scrpit"
10
11  echo " we have those arguments: $@"
12
13  echo "Previous command exited with $? number"
14
15  echo "This script's process id is : $$"
```

```
ubuntu@ip-172-31-86-222:~$ my-first-script.sh first_argument second_argument
script name is /home/ubuntu/my-first-script.sh
 first argument is first_argument
 second argument is second_argument
2 arguments have been passed to scrpit
 we have those arguments: first_argument second_argument
Previous command exited with 0 number
This script's process id is : 3717
```

```
my-first-script.sh
1   #!/bin/bash
2
3   echo "script name is $0"
4
5   echo " first argument is $1"
6
7   echo " second argument is $2"
8
9   echo "$# arguments have been passed to scrpit"
10
11  zecho " we have those arguments: $@"
12
13  echo "Previous command exited with number $?"
14
15  echo "This script's process id is : $$"
```

11 . Satırda komutu yanlış girerek scripti yeniden koşalım:

```
ubuntu@ip-172-31-90-148:~$ my-first-script.sh
0 arguments have been passed to script
/home/ubuntu/my-first-script.sh: line 4: zecho: command not found
previous command exited with number 127
This script's process ID is 2600
```

$USER scripti koşturan kullanıcıyı tutuyor.
$HOSTNAME scriptin koşulduğu makinenin hostname ini tutuyor.
$LINENO ile bir komut yazdığımızda o komutun script içindeki satır numarasını yazdırır.

10. Argümanı yazacaksak ${10} şeklinde yazmamız gerkir:
11. 


Argümanı nasıl kullanacağımıza bir örnek:

my-first-script.sh

```
1  #!/bin/bash
2  USERNAME=$1
3
4  sudo useradd $USERNAME
5
6  echo "user $USERNAME have been added"
7
```

Üstteki script ile my-first-script.sh ı çalıştırarak user ekledik. Bunu aynı anda çoklu kullanıcı eklediğimizde kullanacağız.

```
ubuntu@ip-172-31-86-222:~$ my-first-script.sh mehmet
user mehmet have been added
ubuntu@ip-172-31-86-222:~$
```

# Simple Arithmetic

**expr** command **print** the value of expression to **standard output.**

**expr item1 operator item2**

**let** is a builtin function of Bash that helps us to do simple arithmetic. It is similar to **expr** except instead of printing the answer **it saves the result to a variable.**

**let <arithmetic expression>**

We can also evaluate arithmetic expression with double parentheses.

**$((arithmetic expression))**

expr işlemi herhangi bir değişkene atamadan ekrana yazdırır.

```
expr item1 operator item2
```

```
#!/bin/bash
first_number=8
second_number=2

echo "SUM=" `expr $first_number + $second_number`
echo "SUB=" `expr $first_number - $second_number`
echo "MUL=" `expr $first_number \* $second_number`
echo "DIV=" `expr $first_number / $second_number`
```

```
$ chmod +x cal.sh
$ ./cal.sh
SUM=10
SUB=6
MUL=16
DIV=4
```

let işlemin sonucunu variable olarak kaydeder.

```
let [expression]
```

```
#!/bin/bash

number1=8
number2=2

let total=number1+number2
let diff=number1-number2
let mult=number1*number2
let div=number1/number2

echo "Total = $total"
echo "Difference = $diff"
echo "Multiplication = $mult"
echo "Division = $div"
```

```
$ ./run.sh
Total = 10
Difference = 6
Multiplication = 16
Division = 4
```

$((arithmetic expression)):

```
$((Expression))

((Expression))
```

```
#!/bin/bash

number1=8
number2=2

echo "Total = $((number1+number2))"

((total=number1+number2))
echo "Total = $total"
~
```

```
[ec2-user@ip-172-31-91-206 ~]$ ./run.sh
Total = 10
Total = 10
[ec2-user@ip-172-31-91-206 ~]$
```



1 artırma veya azaltma için kısa yol:

## “num++” “++num” “num--” “--num”

```
#!/bin/bash

number=10
let new_number=number++
echo "Number = $number"
echo "New number = $new_number"

number=10
let new_number=--number
echo "Number = $number"
echo "New number = $new_number"

~
```

```
[ec2-user@ip-172-31-91-206 ~]$ ./run.sh
Number = 11
New number = 10
Number = 9
New number = 9
[ec2-user@ip-172-31-91-206 ~]$
```



# Exercise 1

1. Ask user to enter two numbers to variables **num1** and **num2.**
2. Calculate the total of 2 numbers.
3. Print the **total** number and increase it by 1.
4. Print the new value of the **total** number.
5. Subtract **num1** from the **total** number and print result.
6. Change the **num1** and **num2** variables to be passed from the **Command line arguments** instead of receiving them from the user

```
$ my-first-script.sh
1   #!/bin/bash
2
3   read -p "Enter first number: " NUM1
4   read -p "Enter first number: " NUM2
5
6   let SUM=NUM1+NUM2
7   echo TOTAL = $SUM
8   ((SUM++))
9   echo "NEW_SUM= $SUM"
10
11  let NEW_TOTAL=SUM-NUM1
12  echo "NEW_TOTAL=$NEW_TOTAL"
```

```
ubuntu@ip-172-31-90-148:~$ my-first-script.sh
Enter first number: 8
Enter first number: 6
TOTAL = 14
NEW_SUM= 15
NEW_TOTAL=7
```

```
$ my-first-script.sh
1   #!/bin/bash
2
3   NUM1=$1
4   NUM2=$2
5
6   let SUM=NUM1+NUM2
7   echo TOTAL = $SUM
8   ((SUM++))
9   echo "NEW_SUM= $SUM"
10
11  let NEW_TOTAL=SUM-NUM1
12  echo "NEW_TOTAL=$NEW_TOTAL"
```

PATH e ekleme yapmak için farklı bir yöntem:

```
ubuntu@ip-172-31-90-148:~/shell-scripting$ export PATH=$PATH:$PWD
ubuntu@ip-172-31-90-148:~/shell-scripting$ basic.sh
Hello World
```

Linux ta multiline comment olmuyor. Bunun için şu yöntemi kullanabiliriz:

```
- A heredoc consists of the **<<** `(redirection
operator)`, followed by a delimiter token. After the
delimiter token, lines of string can be defined to
form the content. Finally, the delimiter token is
placed at the end to serve as the termination. The
delimiter token can be any value as long as it is
unique enough that it won't appear within the content.

- Let's see how to use HereDoc.

```bash
cat << EOF
Welcome to the Linux Lessons.
This lesson is about the shell scripting
EOF
```
```

EOF arasında yazdığımız (bu delimiter değişebilir) şeyler orada içerik olarak kalır. komut olarak çalışmaz yani. başında cat olduğu için aşağıda yazdırır. ancak cat yazdırılmazsa çıktı olarak da göremeyiz.