Shell Scripting

Content :

- Introduction
- Variable
- Arguments
- Conditional Statements
- loops
- Functions.

11-October (2024)

عبدالرزاق

→ Shell Scripting for DevOps → used for automation

Interactive Terminal → directly interact with Kernel

for repetitive tasks.

Roadmap :

→ Introduction to Shell and Environment Setup → Terminal setup → vim, nano etc

→ Basic Scripting Skills
- → variables, conditions, functions
- → Operators, loops

→ Intermediate Scripting Techniques
- → Errors handling
- → Email → when failure occur.
- → Backup restore of databases
- → Crone → schedule.

→ Real world Application and Integration

→ Shell Mastery and continous learning

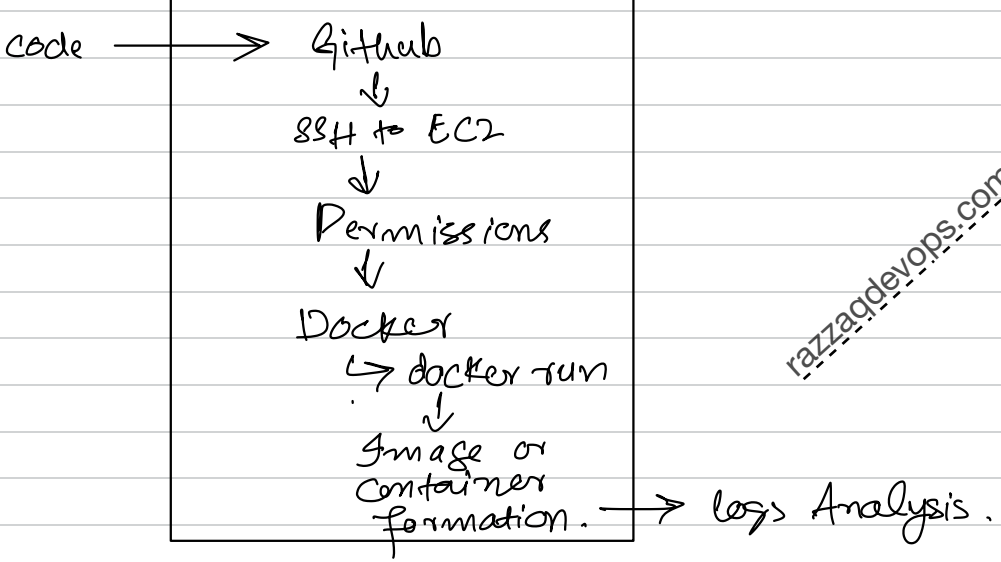
→ Advanced Scripting and Debugging
- → Reading logs
  - ↳ Awk, Sed, grep
- → Automate using python language.

→ Real World Application and Integration
- → Docker integration
- → setupping K8s cluster
  - ↳ updating K8s nodes
    - ↳ Autoscale.

→ Shell Mastery
- ↳ continous learning

→ Project → for Industry.

Shell Script



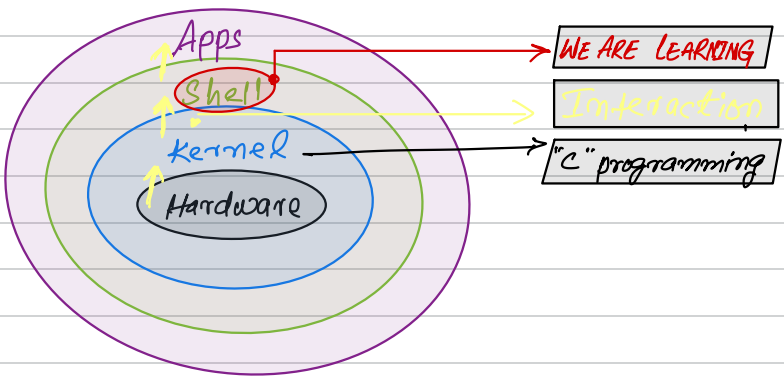
Written By Abdur Razzaq

(Shell Scripting)

INTRODUCTION TO SHELL AND ENVIRONMENT SETUP

Linux Os

Linux Architecture →



→ We operate on linux using shell commands.
↳ Example : We are printing something, shell command is `Echo`
↳ There are a lot of commands to communicate with linux operating system.

→ `/bin/sh` → path.
↓
Improvement in Shell (scripts).
↓
`/bin/bash` → mostly used !

→ To run shell script → First we need an environment → can be provided by multiple cloud platforms.

→ Create an EC2 instance → where we will run shell scripts.

→ Connect with it

→ Install Text Editor to write a Shell Script.

(vim, vi, nano etc)

↓ Insert Mode (Type 'i')

↓ To Exit (:wq , :qa)

:wq ↓ write & quit

:qa ↓ quit without saving.

hello.txt

Hello Abdur

→ To create a new file `vim hello.txt` → (write something & Exit)

→ To view what's in the file.

`cat hello.txt` → "Hello Abdur".

Shell Script

→ Shell script extension "sh"

→ Create a shell script name "hello" → `vim hello.sh`

```
#!/bin/bash
# This is Sample Script
echo "Hello Abdur"
echo "Hello DevOps"
echo "Abdur: create a directory"
echo "DevOps: Okay!"
mkdir sample.
```

→ Tells interpreter which type of shell it is —

`which bash`

↓ It will tell you the bash path.

→ Now if you do `ls -l` → To check permission.

↓ It will be read and write only (Not Executable)

754 → u / g / o

rwx rx

→ Now to make it executable,

run → `chmod 754 hello.sh`

↓ hello.sh is now executable.

↓ Run → `./hello.sh`

Basic Scripting Skills.

↳ variables, Arguments, Conditionals, Loops
Functions.

for while

VARIABLES

→ Single line comment

<< (Block)
Anything
written
here → are commented!
(Block)

Example

<< comment
Anything
written
here → are commented!
comment.

```
# This is Jetha Lal ki Duniya

<< comment
Anything
written
here will not be execute
comment

name="jetha"

echo "Name is $name"
~
~
~
-- INSERT --
```

This is Jetha Lal ki Duniya → Single Line Comment

<< comment
Anything
written
here will not be execute
comment → Multi-line Comment

name="jetha" → Variable

echo "Name is $name" → $ → dollar sign use for identification of variable.

↳ Script file → chmod 755 <file-name.sh>
↳ To run Script → ./<file-name.sh>

Name is jetha → output

```
# This is Jetha Lal ki Duniya

<< comment
Anything
written
here will not be execute
comment

name="babitaji"

echo "Name is $name, and date is $(date)"
```

→ To print date.

TAKE USER INPUT

SCRIPT

```
#! /bin/bash
echo "enter the name :"
read username
echo "you entered $username
```

→ ./script.sh
↳ enter the name :
↳ jetha
↳ you entered jetha.

ARGUMENTS

./file.sh ____ ____

Argument 0 = 1

Argument 1 = ↳

Argument 2 = ↱

→ We can use these arguments in a script or Code.

./file.sh file1 file2

↓

only ($1) first argument will run → Because of no Access.

↓ To print these argument ↓ let's make some changes in the script.

SCRIPT ..

```
echo "The character in $0 $1"
file
file1
:wq
```

OUTPUT ←

file file1

→ We can create multiple users using arguments.

Example:

./create_user.sh

```
#!/bin/bash
if [ -z "$1" ]; then
echo "provide a username"
  exit 1
fi

sudo useradd $1
```

condition to check, as the username is provided or not.

username provided ! Exit Successfully

Add user.

↳ run → ./create_user Razzaq

↳ Argument passed "username provided"

Conditional Statements (if else)

Book - Content

> In Bash, conditional expressions are used by the [[compound command and the [built-in commands to test file attributes and perform string and arithmetic comparisons.

```
if [condition] ;
    then
    ...
else ;
    ...
```

Example

Script To check if-else -

```
#!/bin/bash
read -p "Enter number" number
if [[ $number == number/2 ]];
then echo " number is Even"
else
    echo " number is odd ".
fi
```

accessing variable

variable

condition

```
chmod 755 number.sh
```

↓ run the Script.

```
./number.sh
```

↳ Enter number

4	→	number is Even.
5	→	number is odd.

(Multiple Conditions Scenario)

input from user ←

if num is greater than Zero → so " Positive "

if number is less than Zero → so " Negative "

```
#!/bin/bash

read -p "Enter a number: " num

if [[ $num -gt 0 ]] ; then
    echo "The number is positive"
elif [[ $num -lt 0 ]] ; then
    echo "The number is negative"
else
    echo "The number is 0"
fi
```

Nor Positive Nor Negative ↓ Number is Zero.

(SYNTAX)

```
if
  ...
fi
```

```
[[ condition ]]
```

SWITCH STATEMENTS

↳ As in other programming
↳ We use case statements to simplify complex conditional → When there are multiple diff choices.
↳ So rather using a few "if" and "if-else" statements, we can use a single "case" statement.

SYNTAX:

```
case $some_variable in
    pattern 1 )
        commands
        ;;
    pattern_1 | pattern_3)
        commands
        ;;
    *)
        default commands
        ;;
esac
```

① All case statements start with "case" keyword.
↳ On the same line as the "case" keyword → you need to specify a "variable" or an "expression" followed by "in" keyword.

② You can specify multiple patterns divided by a pipe "|".

③ After the pattern, you specify the commands that you like to be executed in case that the pattern matches the variable OR expression that is specified!

④ All clauses have to be terminated by adding ";;" at the end.

⑤ you can have a default statement by adding a "*" as the pattern.

⑥ To close the "case" statement, use "esac" keyword.
↳ (case type Backward)

Example

user input.

```
#!/bin/bash

read -p "Enter the name of your car brand: " car

case $car in

  Tesla)
    echo -n "${car}'s car factory is in the USA."
    ;;

  BMW | Mercedes | Audi | Porsche)
    echo -n "${car}'s car factory is in Germany."
    ;;

  Toyota | Mazda | Mitsubishi | Subaru)
    echo -n "${car}'s car factory is in Japan."
    ;;

  *)
    echo -n "${car} is an unknown car brand"
    ;;

esac
```

(variable → car)
(case1)
(case2)
(case3)
(default)

BASH LOOPS :

* Loops are used for
  - ↳ Automating repetitive tasks
  - ↳ Iterating through lists or sequences
  - ↳ Executing commands multiple times based on Conditions.

FOR LOOPS :

* Used for iterating over a series of values or performing repetitive tasks.

Structure OF A For Loop

```
for var in ${List}
do
        your_commands
done
```

Example

```
#!/bin/bash

users = "Abdur Razzaq Khan"
for user in ${users}
do
    echo "${users}"
done
```

Example 2

```
#!/bin/bash
for num in {1...10}
do
    echo ${num}
done.
```

WHILE Loop

↳ Executes a set of commands as long as a condition is true.

Structure of While loop

```
While [condition]
do
        your commands
done
```

(Example 1)

less than

```
#!/bin/bash
counter = 1
while [[ $counter -le 10 ]]
do
        echo $counter
        (( counter++ ))
done.
```

Example 2

```
#!/bin/bash
read -p "What is your name" name
While [[ -z ${name} ]]
do
    echo " name cannot be blank
           Please Enter your name "
read -p " Enter your name again" name
done
echo " Hi there ${name} "
```

→ user will prompt to enter name

↳ if left blank

↓ it will continously ask to enter again

↓ until name is given ↳ Exit Loop.

↓ When loop exits it will prompt " Hi there ! "

Shell Script For DevOps

Example #1

1. Automating Deployment on Kubernetes

Use Case: Automating the deployment process of an application on a Kubernetes cluster.

Scenario: Deploying a web application to a Kubernetes cluster, including creating the necessary resources like deployments, services, and config maps.

Shell Script Example:

Sh:

```
#!/bin/bash

# Define variables
NAMESPACE="myapp-namespace"
DEPLOYMENT_NAME="myapp-deployment"
IMAGE="myapp-image:latest"

# Create Kubernetes namespace
echo "Creating Kubernetes namespace..."
kubectl create namespace $NAMESPACE

# Apply ConfigMap
echo "Creating ConfigMap..."
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: myapp-config
  namespace: $NAMESPACE
data:
  APP_ENV: "production"
EOF

# Apply Deployment
echo "Creating Deployment..."
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $DEPLOYMENT_NAME
  namespace: $NAMESPACE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: $IMAGE
        ports:
        - containerPort: 80
        envFrom:
        - configMapRef:
            name: myapp-config
EOF

# Apply Service
echo "Creating Service..."
kubectl apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
  namespace: $NAMESPACE
spec:
  selector:
    app: myapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
EOF

echo "Deployment completed successfully!"
```

Output:

```
Creating Kubernetes namespace...
namespace/myapp-namespace created
Creating ConfigMap...
configmap/myapp-config created
Creating Deployment...
deployment.apps/myapp-deployment created
Creating Service...
service/myapp-service created
Deployment completed successfully!
```

Example 2

Interacting With APIs

Use Case: Automating the creation of GitHub issues via the GitHub API.

Scenario: A script that interacts with the GitHub API to create new issues in a repository based on input parameters.

Shell Script Example:

Sh:

```
#!/bin/bash

# Define GitHub credentials and repository details
GITHUB_USER="your-username"
GITHUB_REPO="your-repo"
GITHUB_TOKEN="your-token"
ISSUE_TITLE="New issue title"
ISSUE_BODY="Description of the new issue"

# Create the issue using the GitHub API
curl -u $GITHUB_USER:$GITHUB_TOKEN -X POST -H "Content-Type:
application/json" \
  -d '{
    "title": "'$ISSUE_TITLE'",
    "body": "'$ISSUE_BODY'"
  }' \
  https://api.github.com/repos/$GITHUB_USER/$GITHUB_REPO/issues
```

Output:

Json:

```
{
  "id": 123456789,
  "number": 1,
  "title": "New issue title",
  "state": "open",
  "body": "Description of the new issue",
  "user": {
    "login": "your-username",
    ...
  }
}
```

Example 3 :

→ Monitoring With AWS CloudWatch

Use Case: Monitoring CPU utilization of an EC2 instance and sending alerts if it exceeds a threshold.

Scenario: A script that uses AWS CLI to set up CloudWatch alarms for monitoring EC2 instances.

Sh:

```
#!/bin/bash

# Define variables
INSTANCE_ID="i-1234567890abcdef0"
ALARM_NAME="HighCPUUtilization"
ALARM_THRESHOLD=80

# Create CloudWatch alarm
aws cloudwatch put-metric-alarm --alarm-name $ALARM_NAME --metric-name
CPUUtilization --namespace AWS/EC2 --statistic Average --period 300 --
threshold $ALARM_THRESHOLD --comparison-operator GreaterThanThreshold --
dimensions Name=InstanceId,Value=$INSTANCE_ID --evaluation-periods 2 --
alarm-actions arn:aws:sns:us-east-1:123456789012:my-sns-topic --unit
Percent

echo "CloudWatch alarm created successfully!"
```

Output:

```
CloudWatch alarm created successfully!
```

FUNCTIONS

↳ functions are great way to reuse code.

↳ for example:

if we want to check the sum of two numbers every time when the user gives input.

↳ We donot write the same code every time for the sum.

↳ So basically, we will write function() which we can call each time user gives input.

★ function contains all the code required for sum ★

Syntax :-

```
function function_name() {
    your_commands
}
```

→ we can also omit the "function" keyword at start.

↳ It will work.

```
function_name() {
    your_commands
}
```

Example →

```
#function to add two numbers
add()
{
x=$1 → variable 1st
y=$2 → variable 2nd
echo -e "Number entered by u are: $x and $y"
echo "sum of $1 and $2 is `expr $x + $y` "
}
# main script
echo "enter first number"
read first → input 1st
echo "enter second number"
read sec → input 2nd
#calling function
add $first $sec → function call.
echo "end of the script"
```

Accessing Variable