

Name: Mayuresh Choudhary
Enrollment Number: 2107004
Batch: A Batch

OPERATING SYSTEMS

Micro-Project

Aim: Executing various Shell commands Creating shell variables , Writing shell scripts using decision making and various control structures., Executing various shell utilities, Using file test and string test conditions in scripts., Making use of Positional Parameters. Configuring your own login shell. Using Functions in Shell scripts.

Shell Scripting

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell. Shell Scripting is a program to write a series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script that can be stored and executed anytime which, reduces programming efforts.

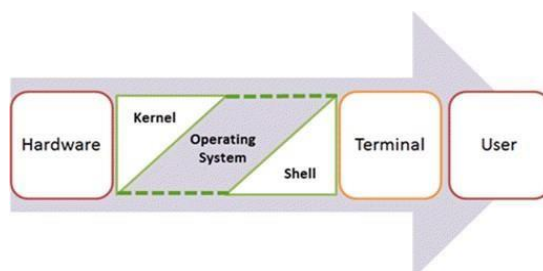
This Shell Scripting tutorial helps to learn a basic understanding of the Linux/Unix shell scripting program to advanced concepts of Shell Scripting. This Shell Script tutorial designed for beginners and professionals who want to learn What is Shell Scripting? How shell scripting works, types of shell, and more.

What is Shell?

Shell is a UNIX term for an interface between a user and an operating system service. Shell provides users with an interface and accepts human-readable commands into the system and executes those commands which can run automatically and give the program's output in a shell script.

An Operating is made of many components, but its two prime components are –

- Kernel
- Shell



A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues a **command prompt (usually \$)**, where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

How to Write Shell Script in Linux/Unix

Shell Scripts are written using text editors. On your Linux system, open a text editor program, open a new file to begin typing a shell script or shell programming, then give the shell permission to execute your shell script and put your script at the location from where the shell can find it.

Let us understand the steps in creating a Shell Script:

1. **Create a file using a vi editor(or any other editor).** Name script file with **extension .sh**
2. **Start** the script with **#!/bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

“#!” is an operator called shebang which directs the script to the interpreter location. So, if we use “#!/bin/sh” the script gets directed to the bourne-shell.

Example:

Demo.sh file:

```
#!/bin/sh
```

A terminal window with a dark background and light-colored text. The prompt is 'onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop\$'. The user enters 'bash new.sh'. The output is 'Hello World'. The prompt returns to 'onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop\$'.

Shell Variables

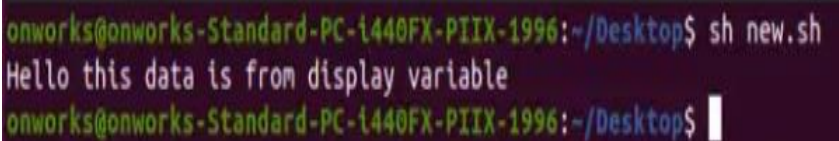
Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can be used by the shell only.

For example, the following creates a shell variable and then prints it:

```
variable="Hello"  
echo $variable
```

Example:

```
#!/bin/sh  
echo "what is your name?"  
read name  
echo "How do you do, $name?"  
read remark  
echo "I am $remark too!"
```



```
onworks@onworks-Standard-PC-L440FX-PIIX-1996:~/Desktop$ sh new.sh  
Hello this data is from display variable  
onworks@onworks-Standard-PC-L440FX-PIIX-1996:~/Desktop$
```

Decision making

Conditional Statements: There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

Their description with syntax is as follows:

if statement

This block will process if specified condition is true.

Syntax:

```
if [ expression ]  
then  
statement
```

if-else statement

If specified condition is not true in if part then else part will be execute.

Syntax

```
if [ expression ]  
then  
statement1  
else  
statement2  
fi
```

if..elif..else..fi statement (Else If ladder)

To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax

```
if [ expression1 ]  
then  
    statement1  
    statement2  
    .  
    .  
elif [ expression2 ]  
then  
    statement3  
    statement4  
    .  
    .  
else  
    statement5  
fi
```

if..then..else..if..then..fi..fi..(Nested if)

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

```
if [ expression1 ]
then
statement1
statement2
.
else
  if [ expression2 ]
  then
    statement3
  .
fi
fi
```

switch statement

case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern

When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed.

If there is no match, the exit status of the case is zero.

Syntax:

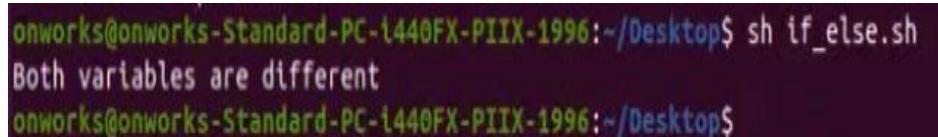
```
case in
Pattern 1) Statement 1;;
Pattern n) Statement n;;
esac
```

Example Programs

Example 1: Implementing if statement

```
#Initializing two variables
a=10
b=20

#Check whether they are equal
if [ $a == $b ]
then
  echo "a is equal to b"
fi
#Check whether they are not
equal
if [ $a != $b ]
then
  echo "a is not equal to b"
fi
```



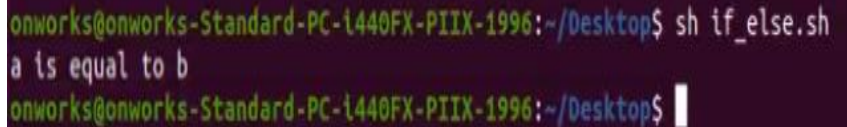
```
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$ sh if_else.sh
Both variables are different
onworks@onworks-Standard-PC-l440FX-PIIX-1996:~/Desktop$
```

Example 2: Implementing if.else statement

```
#Initializing two variables
a=20
b=20

if [ $a == $b ]
then
    #If they are equal then print
    this
    echo "a is equal to b"
else

    #else print this
    echo "a is not equal to b"
fi
```



```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ sh if_else.sh
a is equal to b
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$
```


Example 3: Implementing switch statement

```
CARS="bmw"

#Pass the variable in string
case "$CARS" in
    #case 1
    "mercedes") echo
    "Headquarters - Affalterbach,
    Germany" ;;

    #case 2
    "audi") echo "Headquarters -
    Ingolstadt, Germany" ;;

    #case 3
    "bmw") echo "Headquarters
    - Chennai, Tamil Nadu, India"
    ;;
esac
```



```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ sh switch.sh
Headquaters - Chennai, Tamil Nadu, India
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$
```

Note: Shell scripting is a case-sensitive language, which means proper syntax has to be followed while writing the scripts.

Looping Statements in Shell Scripting:

There are total 3 looping statements which can be used in bash programming

1. while statement
2. for statement
3. until statement

while statement:

Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated

Syntax

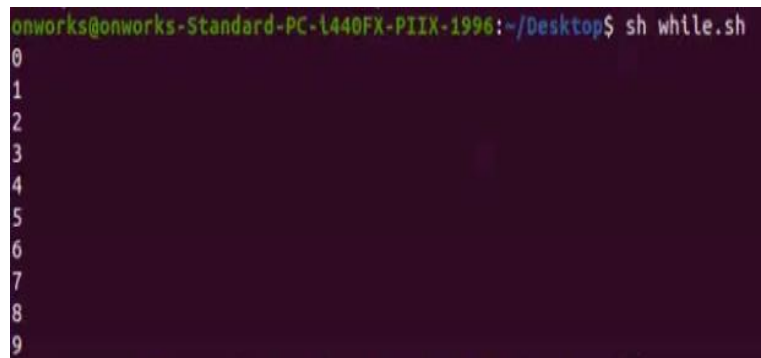
```
while command
do
    Statement to be executed
done
```

Program:

```
#!/bin/sh
a=0

while [ $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

Output:



```
onworks@onworks-Standard-PC-L440FX-PIIX-1996:~/Desktop$ sh while.sh
0
1
2
3
4
5
6
7
8
9
```

for statement:

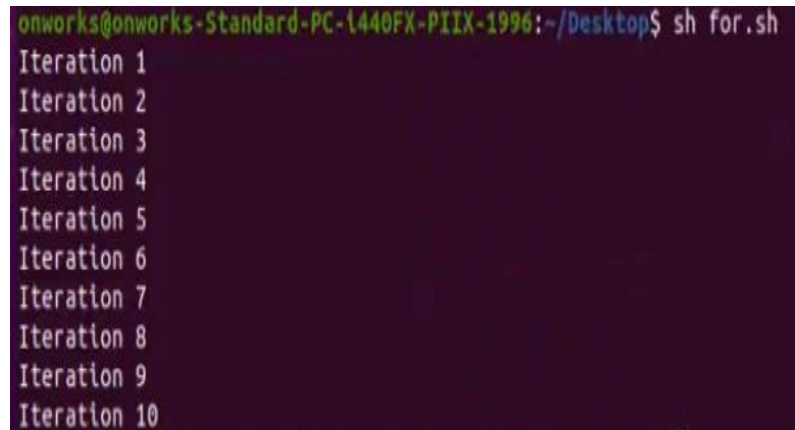
The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax

```
for var in word1 word2 ...wordn
do
    Statement to be executed
done
```

Program:

```
#!/bin/sh
for a in 1 2 3 4 5 6 7 8 9 10
do
    # if a = 5 then continue the loop and
    # don't move to line 8
    if [ $a == 5 ]
    then
        continue
    fi
    echo "Iteration no $a"
done
```



```
onworks@onworks-Standard-PC-L440FX-PIIX-1996:~/Desktop$ sh for.sh
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
```

until statement

The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

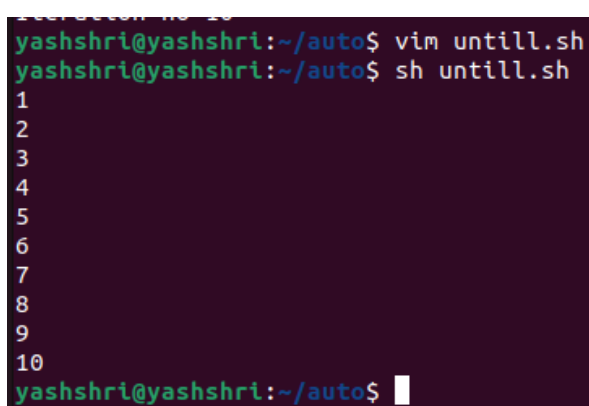
Syntax

```
until command
do
```

Program:

```
#!/bin/sh
i=1
until [ $i -gt 10 ]
do
    echo $i
    ((i++))
done
```

Output:



```
yashshri@yashshri:~/auto$ vim untill.sh
yashshri@yashshri:~/auto$ sh untill.sh
1
2
3
4
5
6
7
8
9
10
yashshri@yashshri:~/auto$
```


Shell Utilities

The shell utilities implement a number of shell commands you can use interactively or in shell scripts. Each utility program performs a specific task. The idea is that you can combine these commands in shell scripts to perform more complicated tasks.

Table 8-2 briefly describes each shell utility. You can try out some of these utilities by typing the program name at the shell prompt. Each program takes command-line options. You should first type **man *progrname***, where *progrname* is the name of the utility, to learn more about the command-line options. Then try the utility with appropriate options.

Cross Ref You encounter some of these shell utilities again in Chapter 24 in the section covering shell programming.

Program	Description
[Evaluates an expression and returns zero if the result is true (same as test)
basename	Removes the path prefix from a given pathname
chroot	Changes the root directory
date	Prints or sets the system date and time
dirname	Removes the last level or filename from a given pathname
echo	Prints a line of text
env	Prints environment variables or runs a command with modified environment variables
expr	Evaluates expressions
factor	Prints prime factors of a number
false	Returns an unsuccessful exit status
groups	Prints the names of the groups to which a user belongs
hostid	Prints the numeric identifier for the host
id	Prints the real and effective user and group IDs
logname	Prints the current login name
nice	Runs a program with specified scheduling priority

Program	Description
nohup	Allows a command to continue running after the user logs out
pathchk	Checks whether filenames are valid and portable
pinky	Prints information about users. (This is similar to the finger program; in fact, the name pinky refers to a lightweight finger.)
printenv	Prints environment variables
printf	Formats and prints data
pwd	Prints the working (current) directory
seq	Prints numeric sequences
sleep	Suspends execution for a specified time
stty	Prints or changes terminal settings
su	Enables the user to adopt the ID of another user, or the super user (root)
tee	Sends output to multiple files
test	Evaluates an expression and returns zero if the result is true
true	Returns a successful exit status
tty	Prints the terminal name
uname	Prints system information
users	Prints current user names
who	Prints a list of all users currently logged in
whoami	Prints the user name for the current effective user ID
yes	Prints a string repeatedly until the process is killed

```

yashshri@yashshri:~/auto$ whoami
yashshri
yashshri@yashshri:~/auto$ who
yashshri tty2          2023-10-15 11:41 (tty2)
yashshri@yashshri:~/auto$ hostid
68bf4e0e
yashshri@yashshri:~/auto$ logname
yashshri
yashshri@yashshri:~/auto$ env
SHELL=/bin/bash
SESSION_MANAGER=local/yashshri:@/tmp/.ICE-unix/1581,unix/yashshri:/tmp/.ICE-unix/1581
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=en
Ubuntu Software  SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
GTK_MODULES=gail:atk-bridge
PWD=/home/yashshri/auto
LOGNAME=yashshri
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=wayland
SYSTEMD_EXEC_PID=1605
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.W9ZGC2
HOME=/home/yashshri
USERNAME=yashshri
IM_CONFIG_PHASE=1
LANG=en_IN
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;

```

File Test Operators

-ef	FILE1 and FILE2 have the same device and inode numbers	FILE1 -ef FILE2
-nt	FILE1 is newer (modification date) than FILE2	FILE1 -nt FILE2
-ot	FILE1 is older than FILE2	FILE1 -ot FILE2
-b	FILE exists and is block special	-b FILE
-c	FILE exists and is character special	-c FILE
-d	FILE exists and is a directory	-d FILE
-e	FILE exists	-e FILE
-f	FILE exists and is a regular file	-f FILE
-h	FILE exists and is a symbolic link (same as -L)	-h FILE
-L	FILE exists and is a symbolic link (same as -h)	-L FILE
-r	FILE exists and read permission is granted	-r FILE

-s	FILE exists and has a size greater than zero	-s FILE
-w	FILE exists and write permission is granted	-w FILE
-x	FILE exists and execute (or search) permission is granted	-x FILE

Positional Parameter

A parameter is an entity that stores values. It can be a name, a number or some special characters. A variable is a parameter denoted by a name. Some variables are set for you already, and most of these cannot have values assigned to them.

These variables contain useful information, which can be used by a shell script to know about the environment in which it is running.

Bash provides two kind of parameters.

- Positional Parameter
- Special Parameter

Positional Parameter Name	Returns true (0) if:
\$0	The name of the script
\$1	The first argument to the script
\$2	The second argument to the script
\$n	The n-th argument to the script
\$#	The number of arguments to the script
\$@	A list of all arguments to the script
\$*	A list of all arguments to the script
\${#N}	The length of the value of positional parameter N (Korn shell only)

\$@ and \$* have the same meaning.

This is true if they are not enclosed in double quotes " ".

Functions: -

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed. Using functions to perform repetitive tasks is an excellent way to create **code reuse**. This is an important part of modern object-oriented programming principles.

Shell functions are similar to subroutines, procedures, and functions in other programming languages.

Creating Functions

To declare a function, simply use the following syntax –

```
function_name ()  
{  
    list of commands  
}
```

The name of your function is **function_name**, and that's what you will use to call it from elsewhere in your scripts. The function name must be followed by parentheses, followed by a list of commands enclosed within braces.

Program:

```
#!/bin/sh  
  
welcome(){  
  
    echo "Hello, welcome to shell  
    scripting"  
  
}
```

Outout:

```
yashshri@yashshri:~/auto$ vim func1.sh  
yashshri@yashshri:~/auto$ vim func2.sh  
yashshri@yashshri:~/auto$ chmod +x func2.sh  
yashshri@yashshri:~/auto$ ./func2.sh  
Helloooo  
yashshri@yashshri:~/auto$
```

Pass Parameters to a Function

You can define a function that will accept parameters while calling the function. These parameters would be represented by **\$1**, **\$2** and so on.

```
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ #! /bin/sh  
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ welcome(){  
> echo "Hello Welcome $1 !"  
> }  
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$ welcome BoysOfIT  
Hello Welcome BoysOfIT !  
onworks@onworks-Standard-PC-i440FX-PIIX-1996:~/Desktop$
```

Conclusion:

Hence, we successfully studied Executing various Shell commands
Creating shell variables, writing shell scripts using decision making and various control structures., Executing various shell utilities, using file test and string test conditions in scripts., Making use of Positional Parameters. Configuring your own login shell. Using Functions in Shell scripts.