www.ihaveapc.com
Mixed Bag Of Tech Blogging And Humor
Windows | Linux | Apps | Humor | Games

Linux Shell Scripting : A Brief Primer

- By www.ihaveapc.com Team

Table Of Contents :

Shell – What is it good for ?

The most powerful thing Linux offers is the shell. The shell is a command line interpreter similar to the command prompt in Windows.

It helps in managing processes, run programs and tons of other important stuff. The command prompt is the interactive part of the shell. When you copy files, delete directories, check if a system is working or not through various commands, shell does that for you.

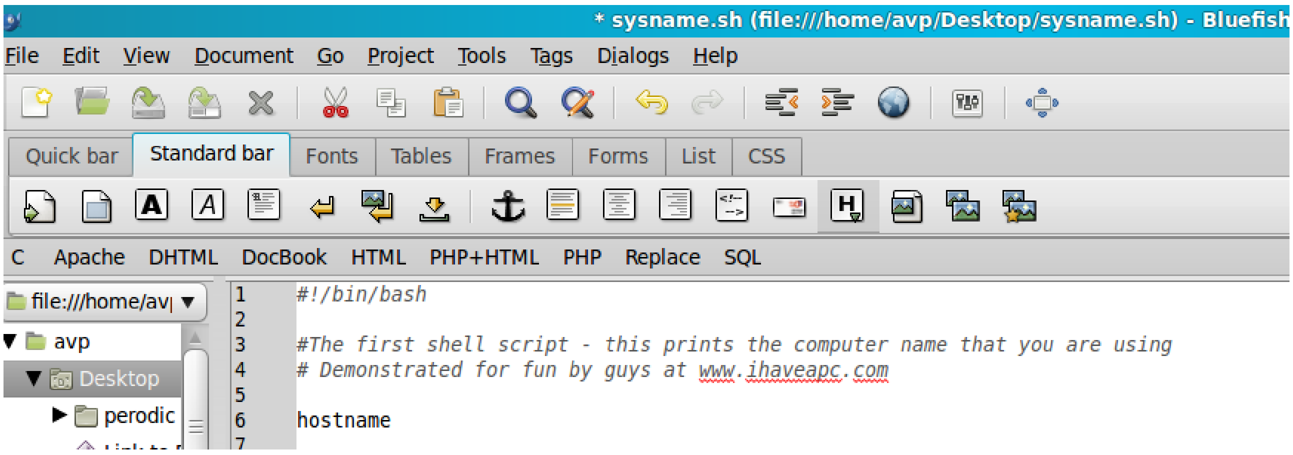
Where there is shell, there are scripts. Simply put, scripts are tiny programs that do what you want them to do thereby freeing you from doing repetitive tasks. Scripts can be anything like checking an IP address of your computer, finding out what the free disk space is in your Linux system or even generating various system based reports.

The First Linux Shell Script

Let's see how to get started with Linux shell scripting using Linux Mint.

What you will need is a Linux system :) and a cool text editor. Bluefish editor is a really cool one for this. Examples in this ebook used Linux Mint system and Bluefish editor to write the scripts.

The first shell script : ***Finding the computer name in Linux Mint***


* sysname.sh (file:///home/avp/Desktop/sysname.sh) - Bluefish
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/av
avp
Desktop
perodic

```
#!/bin/bash

#The first shell script - this prints the computer name that you are using
# Demonstrated for fun by guys at www.ihaveapc.com

hostname
```

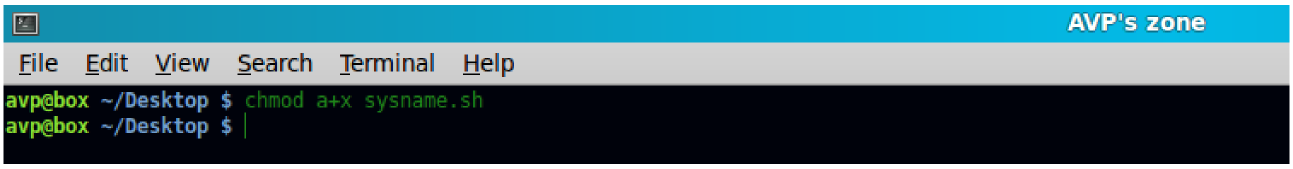
The first line in above example or any Linux shell script (#!/bin/bash in this case) tells the shell which shell to use to run the script. This is called a 'shebang'. So #!/bin/bash specifies bash shell to be used for executing this script.

The next two lines are all comments, anything worth mentioning that gives info about the script should be included by first typing #.

Finally, the command that does the job of finding the Linux Mint computer name is entered on last line – hostname.

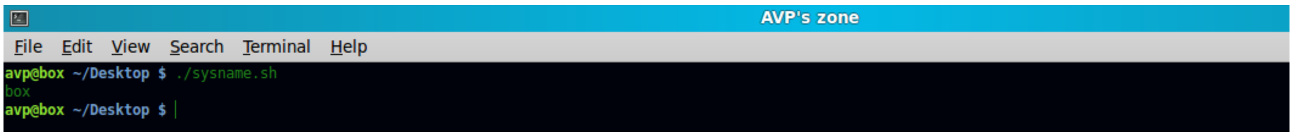
That's it ! You have made your first Linux Mint shell script. Now to make it work, we first save it with a relevant name like sysname.sh

To make it executable so that it can run, type in chmod a+x sysname.sh



```
avp@box ~/Desktop $ chmod a+x sysname.sh
avp@box ~/Desktop $ 
```

One thing to note is that by default if you are in directory A and your script is in directory B, it won't execute because it won't be found. To fix this, either define the path of your script in the $PATH using export command or simply execute the script from the directorty it is present in. The above script is saved on my desktop, so I will execute it from my Desktop folder with the command ./sysname.sh



```
avp@box ~/Desktop $ ./sysname.sh
box
avp@box ~/Desktop $ 
```

Voila, it works ! What the script did was what we wanted it to do, list the name of the current computer. In short, to do something using a Linux script, first write the script, save it somewhere and then make it executable. After that, run it and be overjoyed :)

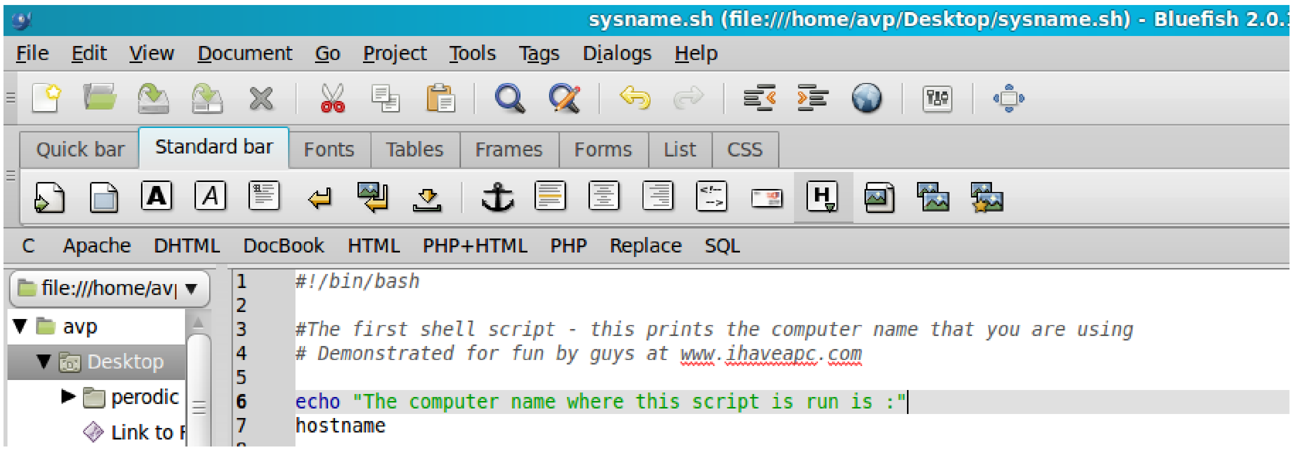
[Note that all examples from now won't mention the step of chmod a+x, so if you make a script and then get Permission Denied error, do ensure that the script being run is made executable first.]

Just to make the above script a bit more clear, let's output some text. This is done with the help of echo.

Example :
echo “ Some text here “. The result would be : Some text here

There are certain nuances when using echo command in shell scripts like when using echo without quotes, we can't use a semicolon and so on. Make sure to read the man page of echo (man echo in Terminal to get an overall idea).

The above script is modified to include echo as follows :

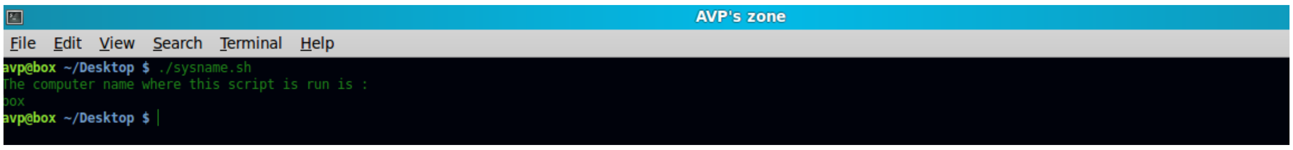

sysname.sh (file:///home/avp/Desktop/sysname.sh) - Bluefish 2.0.
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/avp
avp
Desktop
perodic
Link to

```
#!/bin/bash

#The first shell script - this prints the computer name that you are using
# Demonstrated for fun by guys at www.ihaveapc.com

echo "The computer name where this script is run is :"
hostname
```

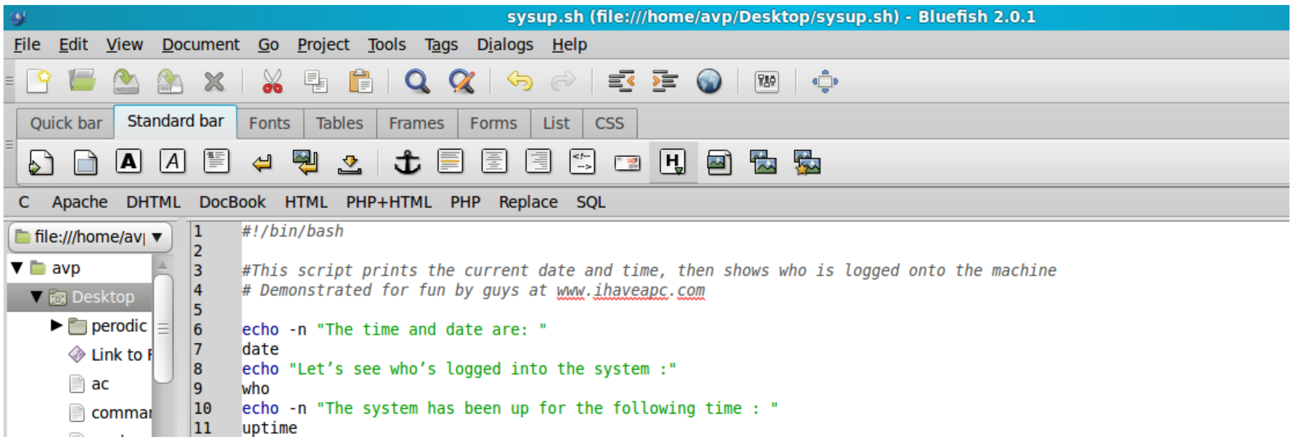
The output for above will be :



```
avp@box ~/Desktop $ ./sysname.sh
The computer name where this script is run is :
box
avp@box ~/Desktop $ |
```

It just outputs the text we want to display for some clarity, that's all.

Script that prints current date and time, displays who is currently logged in and shows for how long the system has been running :

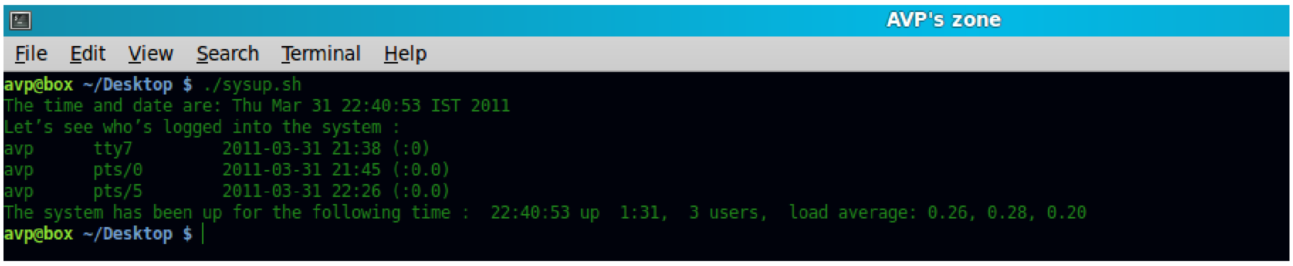

sysup.sh (file:///home/avp/Desktop/sysup.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/avp
avp
Desktop
perodic
Link to f
ac
commar

```
#!/bin/bash

#This script prints the current date and time, then shows who is logged onto the machine
# Demonstrated for fun by guys at www.ihaveapc.com

echo -n "The time and date are: "
date
echo "Let’s see who’s logged into the system :"
who
echo -n "The system has been up for the following time : "
uptime
```

Terminal commands date, who and uptime are used with some meaningful text to tell you what is really being displayed. Note that we used echo -n to format the output. On running the script we get :



```
avp@box ~/Desktop $ ./sysup.sh
The time and date are: Thu Mar 31 22:40:53 IST 2011
Let’s see who’s logged into the system :
avp      tty7         2011-03-31 21:38 (:0)
avp      pts/0        2011-03-31 21:45 (:0.0)
avp      pts/5        2011-03-31 22:26 (:0.0)
The system has been up for the following time :  22:40:53 up  1:31,  3 users,  load average: 0.26, 0.28, 0.20
avp@box ~/Desktop $ |
```

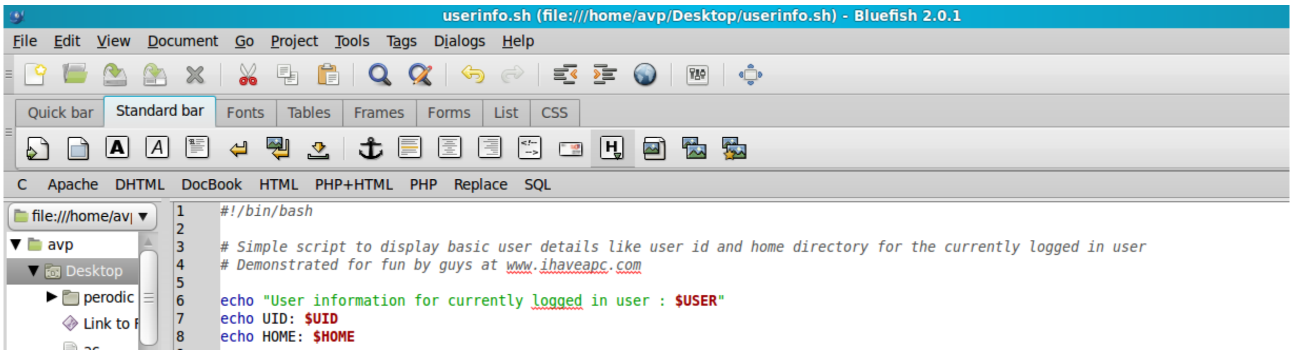
Pretty handy info to get and that too by only writing the above script once and running it again and again !

Let's check out other stuff related to shell scripting. Do note that as this is a very brief ebook, many concepts won't be covered in depth but what this ebook will do is get you up and running on understanding as well as using Linux shell scripts.

Variables

Writing shell scripts is a sort of programming itself with Linux shell following it's own rules and syntax. Variables, loops and user interaction are the heart of any shell script.

Variables help you store temporary information which can be retrieved later in the script. Consider this simple script that demonstrates just that :


userinfo.sh (file:///home/avp/Desktop/userinfo.sh) - Bluefish 2.0.1

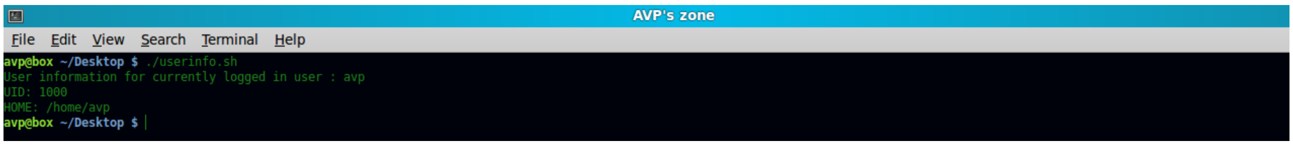
```
#!/bin/bash

# Simple script to display basic user details like user id and home directory for the currently logged in user
# Demonstrated for fun by guys at www.ihaveapc.com

echo "User information for currently logged in user : $USER"
echo UID: $UID
echo HOME: $HOME
```

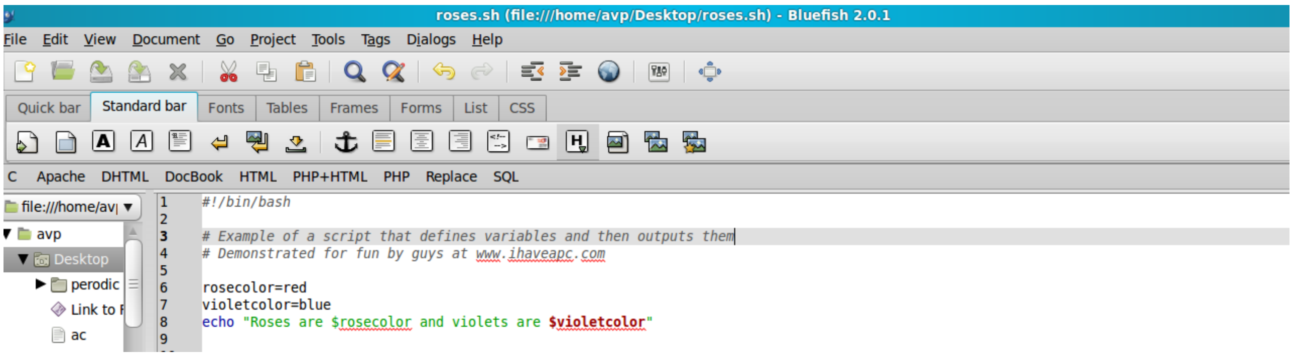
What the above script does is read the user details of the currently logged in user, these are basically system variables already defined and are in all capitals ($USER, $UID and $HOME) which are echoed out. Note that $ is used to output the contents of a variable.

Here is what the output of the script is :



```
avp@box ~/Desktop $ ./userinfo.sh
User information for currently logged in user : avp
UID: 1000
HOME: /home/avp
avp@box ~/Desktop $
```

One more example to get the idea of a user-defined variable compared to the system variable clear :


roses.sh (file:///home/avp/Desktop/roses.sh) - Bluefish 2.0.1

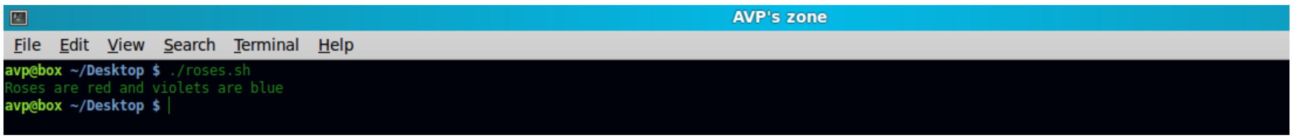
```
#!/bin/bash

# Example of a script that defines variables and then outputs them
# Demonstrated for fun by guys at www.ihaveapc.com

rosecolor=red
violetcolor=blue
echo "Roses are $rosecolor and violets are $violetcolor"
```

Here we have defined two variables named rosecolor and violetcolor each with values red and blue. Note that to properly output them in a sentence we use $ sign like $rosecolor and $violetcolor.

The output of above script is :



```
avp@box ~/Desktop $ ./roses.sh
Roses are red and violets are blue
avp@box ~/Desktop $
```

Just FYI, the other way to output the same variable values is ${var_name}.

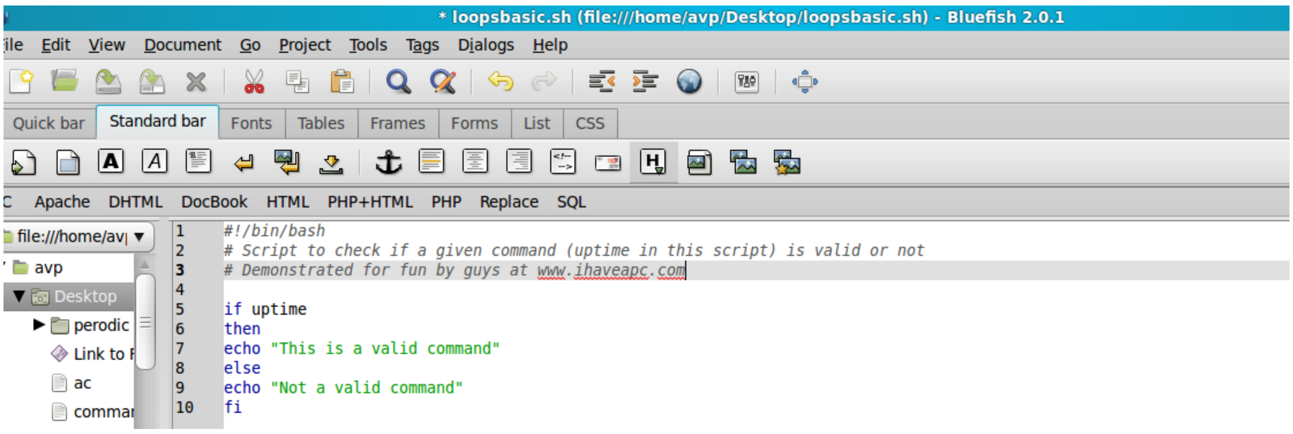
Loops

Loops do what they mean. Check for some condition and execute some stuff as defined. The simplest and pretty useful loop used in shell scripts is the if-then-else loop. There are various other loops too like while, for, until but for simplicity the most common loop is described here.

Here is how it works :

If some command
then
    do some commands as mentioned here
else
do these commands here
fi

An example of script where loop is used will make it more clear.

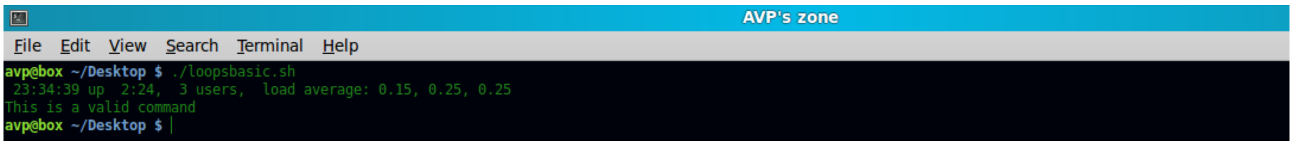

* loopsbasic.sh (file:///home/avp/Desktop/loopsbasic.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/avp
avp
Desktop
perodic
Link to
ac
commai

```
#!/bin/bash
# Script to check if a given command (uptime in this script) is valid or not
# Demonstrated for fun by guys at www.ihaveapc.com

if uptime
then
echo "This is a valid command"
else
echo "Not a valid command"
fi
```

The above script checks if a command (uptime in this case) is valid or not, if it is valid it will display it's contents else will output an error.

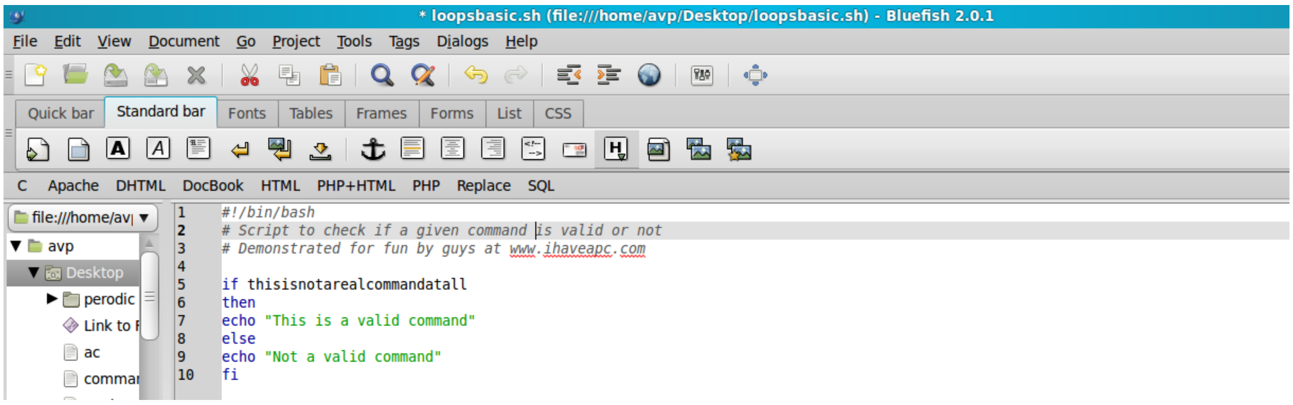
The output of this script is :



```
avp@box ~/Desktop $ ./loopsbasic.sh
 23:34:39 up  2:24,  3 users,  load average: 0.15, 0.25, 0.25
This is a valid command
avp@box ~/Desktop $
```

As uptime is indeed a valid command, the script displayed it's output along with the correct message.

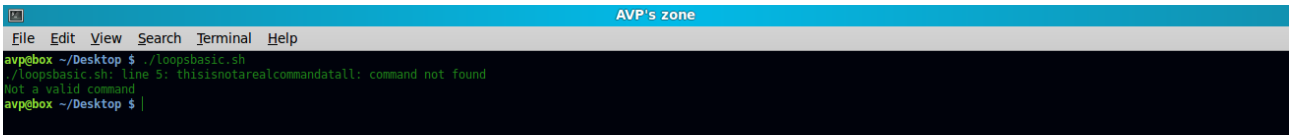
Let us change it to a fake command and see what message do we get :


* loopsbasic.sh (file:///home/avp/Desktop/loopsbasic.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/av
avp
Desktop
perodic
Link to F
ac
commar

```
#!/bin/bash
# Script to check if a given command is valid or not
# Demonstrated for fun by guys at www.ihaveapc.com

if thisisnotarealcommandatall
then
echo "This is a valid command"
else
echo "Not a valid command"
fi
```

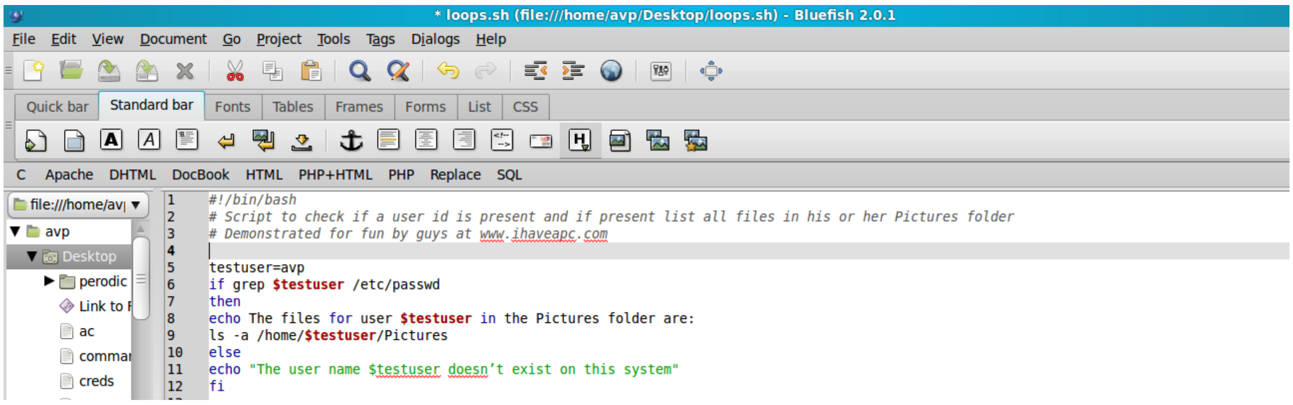
A fake command or any random text that is not a command is entered in above script, output for this script is :



```
avp@box ~/Desktop $ ./loopsbasic.sh
./loopsbasic.sh: line 5: thisisnotarealcommandatall: command not found
Not a valid command
avp@box ~/Desktop $ |
```

As you can see, since the if statement couldn't find the above as a real command, the else part was executed displaying some error message.

Here is a cool script based on the loop discussed above :

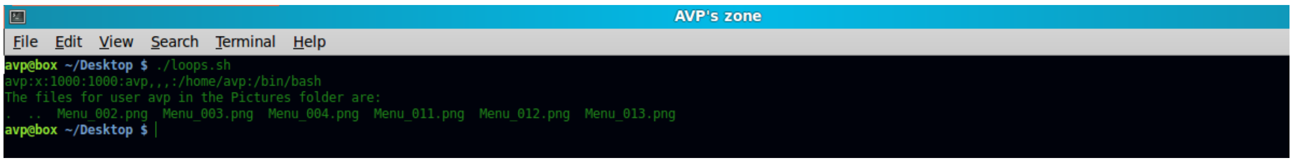

* loops.sh (file:///home/avp/Desktop/loops.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/av
avp
Desktop
perodic
Link to F
ac
commar
creds

```
#!/bin/bash
# Script to check if a user id is present and if present list all files in his or her Pictures folder
# Demonstrated for fun by guys at www.ihaveapc.com

testuser=avp
if grep $testuser /etc/passwd
then
echo The files for user $testuser in the Pictures folder are:
ls -a /home/$testuser/Pictures
else
echo "The user name $testuser doesn't exist on this system"
fi
```

The purpose of the above script is to check for a specific user name, if present it will list all the files of that specific user folder. This is done by using grep on /etc/passwd fileHere, the above script will list all contents of Pictures folder if the user named avp exists.

The output for this script is :



```
avp@box ~/Desktop $ ./loops.sh
avp:x:1000:1000:avp,,,:/home/avp:/bin/bash
The files for user avp in the Pictures folder are:
.  ..  Menu_002.png  Menu_003.png  Menu_004.png  Menu_011.png  Menu_012.png  Menu_013.png
avp@box ~/Desktop $ |
```

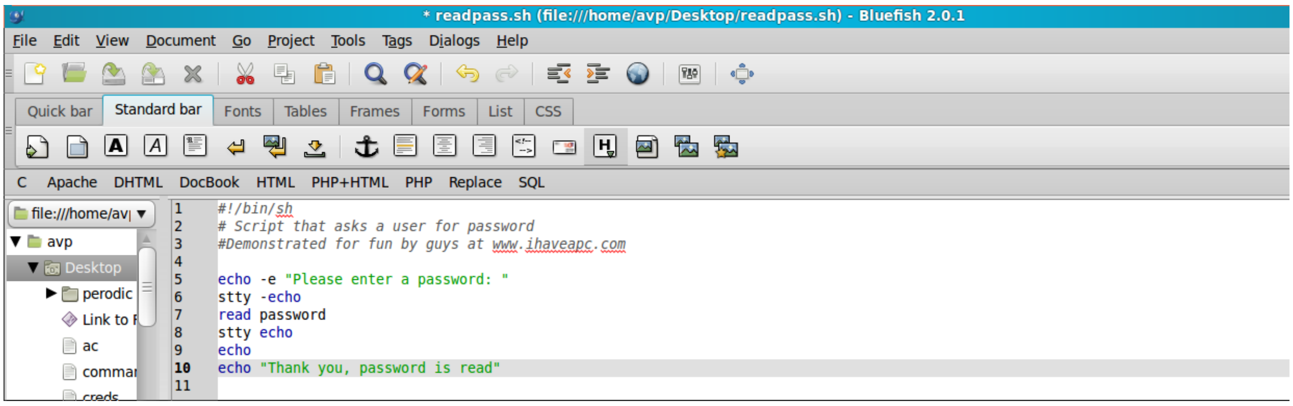
The user named avp did exist based on /etc/passwd file info so it displayed all the files present in the /home/avp/Pictures folder.

Awesome isn't it, few lines that can accomplish so much ?

User Interaction

Now let's take a look at how Linux shell scripts can interact with users.

The below script will ask a user who runs this script for password and confirms that the password is read without displaying it back on the Terminal :

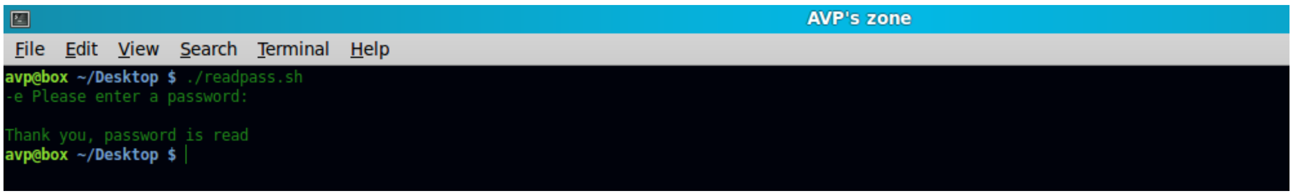

* readpass.sh (file:///home/avp/Desktop/readpass.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/avp
avp
Desktop
perodic
Link to
ac
comma
creds

```
#!/bin/sh
# Script that asks a user for password
#Demonstrated for fun by guys at www.ihaveapc.com

echo -e "Please enter a password: "
stty -echo
read password
stty echo
echo
echo "Thank you, password is read"

```

Output for this is :

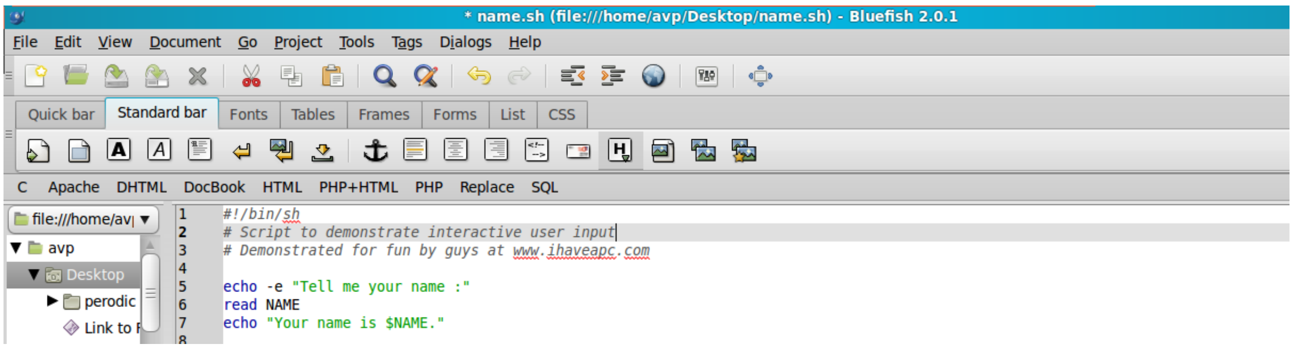


```
avp@box ~/Desktop $ ./readpass.sh
-e Please enter a password:

Thank you, password is read
avp@box ~/Desktop $
```

This is done by using stty -echo which will hide the typed characters. To turn the displayed characters on, simply use stty echo (right after the password was entered by the user in this case.)

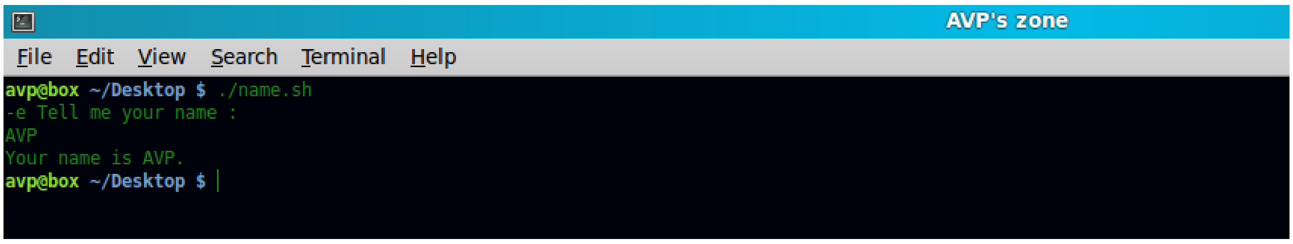
Here is one more, this script will read your name and output it back. This is a basic and classic example of how programs work by taking in inputs from user, process them and display the results.


* name.sh (file:///home/avp/Desktop/name.sh) - Bluefish 2.0.1
File Edit View Document Go Project Tools Tags Dialogs Help
Quick bar Standard bar Fonts Tables Frames Forms List CSS
C Apache DHTML DocBook HTML PHP+HTML PHP Replace SQL
file:///home/avp
avp
Desktop
perodic
Link to

```
#!/bin/sh
# Script to demonstrate interactive user input
# Demonstrated for fun by guys at www.ihaveapc.com

echo -e "Tell me your name :"
read NAME
echo "Your name is $NAME."
```

Output is :



```
avp@box ~/Desktop $ ./name.sh
-e Tell me your name :
AVP
Your name is AVP.
avp@box ~/Desktop $ |
```

What it did was read the user name and stored it in variable $NAME, and then echoed it back that's all.

As mentioned earlier, Linux shell scripting is a vast topic and so we are not able to include many other things here like functions, arrays and other loops but the examples in this ebook should be more than sufficient to get you started on your own in understanding the wonderful art of Linux shell scripting as well as automating repetitive tasks.

When in doubt of any Linux command, you can always use man pages to get help. A brief post explaining how to use man can be found here. Also if you want to understand the basics of Linux Terminal commands, we have an awesome free ebook here.

Hopefully you have enjoyed reading this ebook as much as we enjoyed making it.

Cheers.

The I Have A PC Team
www.ihaveapc.com