**Shell Script Cookbook**

**Table of Contents**

1. Basic of UNIX

1.1 Basic Terminology:

1.2 Basic UNIX Commands

1.3. Unix Environments

1.4 Unix Pipes and filters

2. Shell Programming Cook Book

2.1. My first Script:

2.2. Input and output in Shell Script

2.3. Shell Quotes

2.4. Variable and Data type in Shell.

2.5. Command-Line Arguments

2.6 Array or list in Shell Programming

2.7 Shell Operations

2.8 Decision Control in Shell

2.9 Shell Loop Control

2.10 Shell Functions

2.11 Shell I/0 and File Programming

**1. Basic of UNIX**

* UNIX is a computer Operating System which is capable of handling activities from multiple users at the same time.
* Unix was originated around in 1969 at AT&T Bell Labs by Ken Thompson and Dennis Ritchie. This tutorial gives a very good understanding on Unix.

Several people can use a UNIX computer at the same time; hence UNIX is called a multiuser system.

A user can also run multiple programs at the same time; hence UNIX is called multitasking

**1.1 Basic Terminology:**

**Kernel:** The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, tash scheduling and file management.

**Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want.

**Commands and Utilities:** There are various command and utilities which you would use in your day to day activities. cp, mv, cat and grep etc. are few examples of commands and utilities.

**Files and Directories:** All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

**1.2 Basic UNIX Commands**

Login To Unix System:

login : amrood

amrood's password:

Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73

Print Calendar:

$ cal

June 2009

Su Mo Tu We Th Fr Sa

1 2 3 4 5 6

7 8 9 10 11 12 13

14 15 16 17 18 19 20

21 22 23 24 25 26 27

28 29 30

Add User

ipi@Espresso-Ubuntu64-VM-1:~/ZebOS8NG$ sudo useradd dipankar1

[sudo] password for ipi:

ipi@Espresso-Ubuntu64-VM-1:~/ZebOS8NG$ cd /home/dipankar

ipi@Espresso-Ubuntu64-VM-1:/home/dipankar$

Change Password:

$ passwd

Changing password for amrood

(current) Unix password:\*\*\*\*\*\*

New UNIX password:\*\*\*\*\*\*\*

Retype new UNIX password:\*\*\*\*\*\*\*

passwd: all authentication tokens updated successfully

|  |  |  |
| --- | --- | --- |
| Sl. | What | Command |
|  | Listing Directories and Files: | ls -l  $ls ch\*.doc  $ ls –a # Hidden file |
|  | Who Are You? | $ whoami |
|  | Who is Logged In? | $ users  amrood bablu qadir  $ who  amrood ttyp0 Oct 8 14:10 (limbo)  bablu ttyp2 Oct 4 09:08 (calliope)  qadir ttyp4 Oct 8 12:09 (dent)  $ |
|  | OS and Kernel Vesion | Uname –a |
|  | Reboot | reboot |
|  | Other Commands | vi filename  $ cat filename  wc filename  $ cp source\_file destination\_file  mv old\_file new\_file  $ rm filename  $ rm –rf filename1 filename2 filename3  $cd ~ # HomeDir  $cd ~username  $pwd  /user0/home/amrood  $ls dirname  $mkdir dirname |
|  | Permission Related | ls -l testfile  $chmod o+wx testfile  $ chmod 755 testfile  chown user filelist  $ chgrp special testfile |
|  | Process Related | Foreground Processes: ls ch\*.doc  Background Processes: $ls ch\*.doc &  Listing Running Processes: $ps  Ful version : $ps -f  Stopping Processes: $kill -9 6738  see the statistics of CPU utilization by different processes: $top |
|  | Networking utilities. | Check Reachability : $ping hostname or ip-address  FTP : $ftp amrood.com  ftp> dir  ftp> cd mpl  ftp> get wave\_shift  ftp> quit  The telnet Utility:  C:>telnet amrood.com  SSH unility: ssh [root@10.0.2.15](mailto:root@10.0.2.15)  The finger Utility: - displays information about users on a given host : $ finger |

**1.3. Unix Environments**

An important Unix concept is the environment, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

$TEST="Unix Programming"

$echo $TEST

Unix Programming

Environment variables are set without using $ sign but while accessing them we use $sign as prefix. These variables retain their values until we come out shell.



|  |  |
| --- | --- |
| Setting path | $PATH=/bin:/usr/bin |
| PS1 and PS2 Variables | $PS1='=>'  =>  =>  =>  =>PS1="[\u@\h \w]\$"  [root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$  [root@ip-72-167-112-17 /var/www/tutorialspoint/unix]$ |
| he default secondary prompt is > (the greater than sign), but can be changed by re-defining the PS2 | $ echo "this is a  > test"  this is a  test  $  $ PS2="secondary prompt->"  $ echo "this is a  secondary prompt->test"  this is a  test  $ |
|  |  |
|  |  |

**1.4 Unix Pipes and filters**

Pipe connect two commands together so that the output from one program becomes the input of the next program.

The grep program searches a file or files for lines that have a certain pattern. The syntax is:

$ls -l | grep "Aug"

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$$ls -l | grep -i "carol.\*aug"

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

|  |  |
| --- | --- |
| **-v** | Print all lines that do not match pattern. |
| **-n** | Print the matched line and its line number. |
| **-l** | Print only the names of files with matching lines (letter "l") |
| **-c** | Print only the count of matching lines. |
| **-i** | Match either upper- or lowercase. |

The sort command arranges lines of text alphabetically or numerically.

A long output would normally zip by you on the screen, You can use more see some part.

The following pipe consists of the commands ls, grep, and sort:

$ls -l | grep "Aug" | sort +4n

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

$$ls -l | grep "Aug" | sort +4n | more

2. Shell Programming CookBook

**2.1 My first Script:**

#!/bin/bash

# Author : Dipankar Duuta

# Script follows here:

pwd

ls

--------------------------------------------------------------------

$chmod +x test.sh

$./test.sh

**2.2 Input and output in Shell Script**

#!/bin/sh

# Author : Dipankar

# Script follows here:

echo "What is your name?"

read PERSON

echo "Hello, $PERSON"

---------------------------------------------------------------------

$./test.sh

What is your name?

Dipankar

Hello, Dipankar

$

Output can be formatted by Shell substitutions as below:

a=10

echo -e "Value of a is $a \n" # Value of a is 10

**# Command Substitution: syntax : `command`**

DATE=`date`

echo "Date is $DATE"

USERS=`who | wc -l`

echo "Logged in user are $USERS"

UP=`date ; uptime`

echo "Uptime is $UP"

The Metacharacters:

\* ? [ ] ' " \ $ ; & ( ) | ^ < > new-line space tab

echo Hello; Word

-----

Hello

./test.sh: line 2: Word: command not found

Correct is:

echo Hello\; Word

echo "I have \$1200"

**2.3 Shell Quotes**

|  |  |
| --- | --- |
| **Quoting** | **Description** |
| **Single quote** | All special characters between these quotes lose their special meaning. |
| **Double quote** | Most special characters between these quotes lose their special meaning with these exceptions:   * $ * ` * \$ * \' * \" * \\ |
| **Backslash** | Any character immediately following the backslash loses its special meaning. |
| **Back Quote** | Anything in between back quotes would be treated as a command and would be executed. |

2.4 Variable and Data type in Shell.

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

**Variable Names:** contain only letters ( a to z or A to Z), numbers ( 0 to 9) or the underscore character ( \_)

#!/bin/sh

NAME="Zara Ali" # Define

NAME1="Zara Ali" # Define

echo $NAME # Access Variable

readonly NAME # Make Readonly

# NAME="Qadiri" # This will through error

unset NAME1 #Unsetting Variable

echo $NAME1

**2.5. Command-Line Arguments:**

The command-line arguments $1, $2, $3,...$9 are positional parameters

#!/bin/sh

echo "File Name: $0"

echo "First Parameter : $1"

echo "First Parameter : $2"

echo "Quoted Values: $@"

echo "Quoted Values: $\*"

echo "Total Number of Parameters : $#"

----------------------------------

$./test.sh Zara Ali

File Name : ./test.sh

First Parameter : Zara

Second Parameter : Ali

Quoted Values: Zara Ali

Quoted Values: Zara Ali

Total Number of Parameters : 2

Special Parameters $\* and $@: "$\*" special parameter takes the entire list as one argument with spaces between and the "$@" special parameter takes the entire list and separates it into separate arguments

#!/bin/sh

for TOKEN in $\*

do

echo $TOKEN

done

----------------------

$./test.sh Zara Ali 10 Years Old

Zara

Ali

10

Years

Old

**2.5 Array or list in Shell Programming:**

# Defining an Array

NAME[0]="Zara"

NAME[1]="Qadir"

NAME[2]="Mahnaz"

NAME[3]="Ayan"

NAME[4]="Daisy"

# Accessing value..

echo "First Index: ${NAME[0]}"

echo "Second Index: ${NAME[1]}"

# Two way to access full things

echo "First Method: ${NAME[\*]}"

echo "Second Method: ${NAME[@]}"

# Note that echo $NAME[1] and echo echo ${NAME[1]} Are not same.

2.7 Shell Operations

- The Bourne shell didn't originally have any mechanism to perform simple arithmetic but it uses external programs: expr

ipi@Espresso-Ubuntu64-VM-1:/home/dipankar$ expr 2 + 2 #output 4

ipi@Espresso-Ubuntu64-VM-1:/home/dipankar$ expr 'helo' + 'world'# output expr: non-integer argument

ipi@Espresso-Ubuntu64-VM-1:/home/dipankar$ expr 2+2 # print 2+2 so space is important.

More Example:

#!/bin/sh

a=10

b=20

val=`expr $a + $b`

echo "a + b : $val"

val=`expr $a - $b`

echo "a - b : $val"

val=`expr $a \\* $b`

echo "a \* b : $val"

val=`expr $b / $a`

echo "b / a : $val"

val=`expr $b % $a`

echo "b % a : $val"

if [ $a == $b ]

then

echo "a is equal to b"

fi

if [ $a != $b ]

then

echo "a is not equal to b"

fi

# More conditional expression here:

if [ $a -lt 100 -a $b -gt 15 ]

then

echo "$a -lt 100 -a $b -gt 15 : returns true"

else

echo "$a -lt 100 -a $b -gt 15 : returns false"

fi

if [ $a -lt 100 -o $b -gt 100 ]

then

echo "$a -lt 100 -o $b -gt 100 : returns true"

else

echo "$a -lt 100 -o $b -gt 100 : returns false"

fi

if [ $a -lt 5 -o $b -gt 100 ]

then

echo "$a -lt 100 -o $b -gt 100 : returns true"

else

echo "$a -lt 100 -o $b -gt 100 : returns false"

fi

-------------------------

a + b : 30

a - b : -10

a \* b : 200

b / a : 2

b % a : 0

a is not equal to b

Note:

1. There must be spaces between operators and expressions for example 2+2 is not correct, where as it should be written as 2 + 2.
2. All conditional expressions would be put inside square braces with one spaces around them, for example [ $a == $b ] is correct whereas [$a==$b] is incorrect.
3. Complete expression should be enclosed between ``, called inverted commas.
4. You should use \ on the \* symbol for multiplication.
5. if...then...fi statement is a decision making statement which has been explained in next chapter.
6. if...then...else...fi statement is a decision making statement which has been explained in next chapter.
7. + : Addition -`expr $a + $b` will give 30
8. - : Subtraction - `expr $a - $b` will give -10
9. \* : Multiplication -`expr $a \\* $b` will give 200
10. / : Division - `expr $b / $a` will give 2
11. % : Modulus – remainder `expr $b % $a` will give 0
12. = : Assignment - a=$b would assign value of b into a
13. == : Equality - Compares two numbers, [ $a == $b ] would return false.
14. != Not Equality
15. -eq :equal [ $a -eq $b ] is not true.
16. -ne : Not Equeal [ $a -ne $b ] is true.
17. -gt : Greater than [ $a -gt $b ] is not true.
18. -lt : Less Than [ $a -lt $b ] is true.
19. -ge : Gretaer than and Equal[ $a -ge $b ] is not true.
20. -le Less or Equal [ $a -le $b ] is true.
21. ! This is logical negation. [ ! false ] is true.
22. –o This is logical OR. [ $a -lt 20 -o $b -gt 100 ] is true.
23. –a This is logical AND. [ $a -lt 20 -a $b -gt 100 ] is false.
24. For string we have following operations:
    1. = Checks if String equal; [ $a = $b ] is not true.
    2. != Checks if String not equal [ $a != $b ] is true.
    3. -z Checks String len [ -z $a ] is not true.
    4. -n is string len is NonZero [ -z $a ] is not false.
    5. str Is Empty String. [ $a ] is not false.

2.8 Decision Control in Shell

- 3 Decision Control Discuss below.

a=10

b=20

# 1. If control Block.

if [ $a != $b ]

then

echo "a is equal to b"

fi

# 2. If-else-Block

if [ $a == $b ]

then

echo "a is equal to b"

else

echo "a is not equal to b"

fi

# 3. If-elif-elif-else block

if [ $a == $b ]

then

echo "a is equal to b"

elif [ $a -gt $b ]

then

echo "a is greater than b"

elif [ $a -lt $b ]

then

echo "a is less than b"

else

echo "None of the condition met"

fi

#4. Case in block

#!/bin/sh

FRUIT="apple"

case "$FRUIT" in

"apple") echo "Apple …”

;; #<<<<<<<<<<<<<<<<<<<<<<<<<< This is like break

"banana") echo "banana”

;;

"kiwi") echo "this kiwi”

;;

esac

**2.9 Shell Loop Control.**

It support 4 loops controls

1. The while loop
2. The for loop
3. The until loop
4. The select loop
   1. Case with Multile same action can be combine here
   2. Case with Null input
   3. Case with invalid input

#!/bin/sh

# 1. While Loop for print 0 to 9

a=0

while [ $a -lt 10 ]

do

echo $a

a=`expr $a + 1`

done

#2. For-in loop like python

for var in 0 1 2 3 4 5 6 7 8 9

do

echo $var

done

#3. Untill just like do-while.

a=0

until [ ! $a -lt 10 ]

do

echo $a

a=`expr $a + 1`

done

#4 Select loop provides an easy way to create a numbered menu from which users can select options.

select DRINK in tea cofee water juice appe all none

do

case $DRINK in # Case With multiple matches.

tea|cofee|water|all)

echo "Go to canteen"

;;

juice|appe)

echo "Available at home"

;;

none) # Case wih Null input

break

;;

\*) echo "ERROR: Invalid selection" # Case with Invalid input

;;

esac

done

----------------

$PS3="Please make a selection => " ; export PS3

$./test.sh

1) tea

2) cofee

3) water

4) juice

5) appe

6) all

7) none

Please make a selection => juice

Available at home

Please make a selection => none

$

Loops can be nested, given as below:

#!/bin/sh

a=0

while [ "$a" -lt 10 ] # this is loop1

do

b="$a"

while [ "$b" -ge 0 ] # this is loop2

do

echo -n "$b "

b=`expr $b - 1`

done

echo

a=`expr $a + 1`

done

----------------

0

1 0

2 1 0

3 2 1 0

4 3 2 1 0

5 4 3 2 1 0

6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

8 7 6 5 4 3 2 1 0

9 8 7 6 5 4 3 2 1 0

Shell also support Break and continue as bewlo:

1. break is used to break-out the loop

2. break n will break n nested loop

3. <continue / continue n > is work as this.

a=0

while [ $a -lt 10 ]

do

echo $a

if [ $a -eq 5 ]

then

break

fi

a=`expr $a + 1`

done

for var1 in 1 2 3

do

for var2 in 0 5

do

if [ $var1 -eq 2 -a $var2 -eq 0 ]

then

break 2

else

echo "$var1 $var2"

fi

done

done

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS

do

Q=`expr $NUM % 2`

if [ $Q -eq 0 ]

then

echo "Number is an even number!!"

continue

fi

echo "Found odd number"

done

2.10 Shell Functions

This is as simple creating a function in C/C++

* Function Without arguments
* Function Having Arguments
* Function having return value.

#!/bin/sh

# Define your function here

Hello () {

echo "Hello World"

}

# Invoke your function

Hello

# Define your function here

Hello1 () {

echo "Hello World $1 $2"

}

# Invoke your function

Hello1 Zara Ali

# Define your function here

Hello2 () {

echo "Hello World2 $1 $2"

return 10

}

# Invoke your function

Hello Zara Ali

# Capture value returnd by last command

ret=$?

echo "Return value is $ret"

# Nested functions.

number\_one () {

echo "This is the first function speaking..."

number\_two

}

number\_two () {

echo "This is now the second function speaking..."

}

# Calling function one.

number\_one

**2.11 Shell I/0 and File Programming.**