Shell scripting: bash syntax, environment variables, variables, control

constructs such as if, for and while, aliases and functions, accessing command

line arguments passed to shell scripts. Startup scripts, login and logout

scripts, familiarity with system and system 5 init scripts is expected.

1. Creating a shell script

Create a new directory bash\_scripts where you will run the shell scripting

exercises.

mkdir bash\_scripts

cd bash\_scripts

Note: It is recommended to create a new script file for every exercise; make them

executable; give them names with extension .sh – it is just a convention rather

than a must. The first line of a bash script starts with

#!/bin/bash

All non-executable comments in a script are prepended with #,

# For example, we list all the files in the current director ls -la ls -l /etc

# Comment. Here, we list files in /etc directory.

Script scr1.sh is like a calculator:

#!/bin/bash echo "I will work out X\*Y"

echo "Enter X"

read X

echo "Enter Y"

read Y

echo "X\*Y = $X\*$Y = $[X\*Y]"

Make the script executable and run

chmod 755 scr1.sh

./scr1.sh

2. if and case statements

1) #!/bin/bash

X=10

Y=5

if [ "$X" -gt "$Y" ]; then

echo "$X is greater than $Y"

elif [ "$X" -lt "$Y"]; then

echo "$X is less than $Y"

else

echo "$X is equal to $Y"

fi

2) #!/bin/bash

case $1 in

--test|-t)

echo "you used the --test option"

exit 0

;;

--help|-h)

echo "Usage:"

echo " myprog.sh

[--test|--help|--version]"

exit 0

;;

--version|-v)

echo "myprog.sh version 0.0.1"

exit 0

;;

\*)

echo "No such option $1"

echo "Usage:"

echo " myprog.sh

[--test|--help|--version]"

exit 1

;;

esac

echo "You typed \"$1\" on the command-line"

Note:always watch for correct syntax of case statement.

3) case string

in

regex1)

commands1

;;

regex2)

commands2

;;

........

esac

Where regex is a regular expression to match the string. To catch all remaining

strings, use \*) at the end.

3. Looping with while and until statements

1) Script scr2.sh:

#!/bin/bash

N=1

while [ "$N" -le "10" ]

do

echo "Number $N"

N=$[N+1]

done

2)Script scr3.sh

#!/bin/bash

N=1

until [ "$N" -gt "10" ]

do

echo "Number $N"; N=$[N+1]

done

Note: common mistakes in shell scripting are usually due to incorrect syntax.

For example, there should be no spaces before and after operator “=” .

N=1 # correct

N =1 # error

N= 1 # error

N=$[N+1] # correct

N =$[N+1] # error

N= $[N+1] # error

4. Looping with for statement

1)Script scr4.sh

#!/bin/bash

for i in red white blue

do

echo "$i is a color"

done

2)Script backup-lots.sh

#!/bin/bash

for i in 0 1 2 3 4 5 6 7 8 9 ;do

cp $1 $1.BAK-$i

done

Now create a file important\_data with some numbers in it and then run

./backup-lots.sh important\_data

which will copy the file 10 times with 10 different extensions. As you can

see, the variable $1 has a special meaning – it is the first argument on the

command-line.

Note: watch for correct syntax:

for i in 0 1 2 3 4 5 6 7 8 9 do .... done

The continue statement is useful for terminating the current iteration of the

loop.

3) #!/bin/bash

for i in 0 1 2 3 4 5 6 7 8 9 ; do

NEW\_FILE=$1.BAK-$i

if [ -e $NEW\_FILE ] ; then

echo "backup-lots.sh: \*\*warning\*\* $NEW\_FILE"

echo " already exists - skipping"

continue

fi

cp $1 $NEW\_FILE

done

5. Functions

Function definitions provide a way to group statement blocks into one.

1)

#!/bin/bash

function usage ()

{

echo "Usage:"

echo " myprog.sh [--test|--help|--version]"

}

case $1 in

--test|-t)

echo "you used the --test option"

exit 0

;;

--help|-h)

usage

;;

--version|-v)

echo "myprog.sh version 0.0.2"

exit 0

;;

-\*)

echo "Error: no such option $1"

usage

exit 1

;;

esac

echo "You typed \"$1\" on the command-line"

Note: watch for syntax:

2)

function usage ()

{

command1

command2; command3

......

}

The word function in a function is optional.

That is, the following will work as well:

3)

usage ()

{

command1

command2; command3

.....

}

6. Using quotes

Single forward quotes ' protect the enclosed text from the shell.

1)

echo 'error $?'

echo 'shell name $0'

Double quotes " allow all shell interpretations to take place inside them.

2)

echo "error $?" #gives the error code of the last command

echo "shell name $0" #gives the current shell name

Command substitution

3)

X=`expr 100 + 50 '\*' 3`

echo $X

Assigning command output to a variable:

4)

FSIZE=`wc -l /etc/profile`

same as

5)

FSIZE=$(wc -l /etc/profile)

7. Introduction to awk

The basic function of awk is to search files for lines or other text units containing

one or more patterns. When a line matches one of the patterns, special actions are

performed on that line. Display user names from /etc/passwd (field 1):

1)

awk -F: '{ print $1 }' /etc/passwd

Where F is the field separator in the passwd file. The fields are separated by : Default

field separator is a blank space. Awk scans the input file and splits each input line

into fields.

2)

cat /etc/passwd | awk -F: '{ print $1 }'

Display user names home directories and login shell (fields 1 and 7), and store them

in a separate file, users.txt

3)

awk -F: '{ print $1, $6, $7 }' /etc/passwd > users.txt

or

cat /etc/passwd | awk -F: '{ print $1, $6, $7 }' > users.txt `

Default field separator is empty space. To print users (field 1) from just created file

users.txt: awk '{ print $1 }' users.txt

8.Introduction to sed

String editor, sed, is used for editing lines in a file or a stream; output is going to the

standard output and can be re-directed to a new file.

Syntax:

sed [options] 'command1' [files]

sed [options] -e 'command1' [-e command2 ...] [files]

sed [options] -f script [files]

Delete lines from 3 through 5 in file list.txt:

sed '3,5d' list.txt

Delete lines that contain “O” at the beginning of the line:

sed '/^O/d' list.txt

Translate capital C,R,O into small c,r,o:

sed 'y/CRO/cro/' list.txt

Delete empty lines:

sed '/^$/d' list.txt

Replace string Oop with Wee for the first occurence on a line

sed 's/Oop/Wee/' lsst.txt

Remove ss string (replace with empty entry)for the first occurence on a line:

sed 's/ss//' list.txt

Remove ss string for all occurrences on a line:

sed 's/ss//g' list.txt

Substitute a single space for any number of spaces wherever they occur on the

line:

sed 's/ \*/ /g' list.txt

Substitute underscore for any number of spaces wherever they occur on the

line:

sed 's/ \*/\_/g' list.tx