KIT104 ICT Architecture and Operating Systems Tutorial Week 10

1. UNIX Shell Scripting – Looping – First Sample Script

The following sample script (and the sample script in section 2) reinforce the looping concept in UNIX.

A — Under your `kit104` directory, change into the directory named `Looping` which you created last week. If you have removed it, create it now (and change into it).

B — Create a script which must be named as `cpback.sh`, with the following content. *You do not need to type the comment lines into your script*. They are there to help you understand the script.

```
#! /bin/sh

# This program copies a file into a safe directory without overwriting. For example, it
# copies a file named foo to foo.1 if foo exists in the destination directory, or to
# foo.2 if foo.1 exists in the destination directory.

# To run the script, two arguments are required in the command line, a source file,
# followed by a destination directory. Without two arguments, the scrip displays the
# usage information, and exits

if [ $# -ne 2 ] ; then
   echo "Usage: $0 source destination"; exit

# The script also requires that the second argument must be an existing directory file,
# if not, displays an error message and exits

elif [ ! -d $2 ] ; then
   echo "Directory does not exist" ; exit
else

   # assign the first argument (the source filename) as value to variable file
   file=$1

   # If the source file does not exist in the destination directory, then simply copy it
   # into the directory
   if [ ! -f $2/$file ] ; then
      cp $1 $2

   # But if the source file does exist in the destination directory, then we have to do
   # something to avoid overwriting
   else
      copies=1      # The variable copies takes 1 as initial value
      while true
```

	```
	      do
	        if [ ! -f $2/$file.$copies ] ; then
	           cp $1 $2/$1.$copies
	           echo "File $1 copied to $1.$copies"
	           break
	           else
	             copies=`expr $copies + 1`
	        fi
	      done
	   fi
	fi
	```

C	Save the content of the script. Assign execute permission to the script.
	
	To run the scrip, you need a support source file and a support destination directory.
	
	Create a support text file named `foo`. The easiest way to do this is to redirect the output of a command line into the file:
	
	`$ man man > foo`
	
	Create a support directory named `safedir`.
	
	Run the script as follows:
	`$ ./cpback foo safedir`
	
	Run the command line two more times:
	`$ ./cpback foo safedir`
	`$ ./cpback foo safedir`
	
	Use `ls` to list all the file names stored under `safedir`. Has any file overwriting occurred?
	only one foo file

D	**Explain the functionality of the `while` loop in the last section of the above script (5 marks).**

2. UNIX Shell Scripting – Looping – Second Sample Script

A

Create a script which must be named as `dentry.sh`, with the following content. *You do not need to type the comment lines into your script*. They are there to help you understand the script.

Please note that this is a script with nested loop structure, that is, an inner loop is contained within an outer loop.

```
#! /bin/sh

# This script repeatedly accepts a two digit code (such as 01, 02, etc) and its
# corresponding description (such as engineer) from keyboard, performs some
# validation check, and then adds the entry into a text file named newlist

# First create a new blank text file named newlist, if it does not currently exist
if [ ! -f newlist ]; then
 echo > newlist
fi

# Start the outer loop by prompting the user to enter a code
while echo -e "Designation code: \c"
do
  read code
  case "$code" in
    # Ensure the two digit code is in correct format, and does not currently exist
    [0-9][0-9]) if grep "$code" newlist >/dev/null
                then echo "Code exists"; continue
                fi ;;
             *) echo "Invalid code"  ; continue ;;
  esac

  # Start the inner loop by prompting the user to enter a description
  while echo -e "Description: \c";
  do
    read desc

    case "$desc" in
    # This is how to ensure that the user has entered a valid description, which
    # contains letters and spaces only
    *[!\ a-zA-Z]*)  echo "Can contain only letters and spaces"; continue ;;
              # If the user simply presses the Enter key, without typing anything
               "")  echo "Description not entered" ; continue ;;
              # The user has entered a valid description, add it together with
              # the previously entered valid code into the file
                *)  echo "$code|$desc" >> newlist ; break ;;
    esac

  done
```

	```
	    echo -e "\nEnter another entry? (y/n): \c"
	    read answer
	    case "$answer" in
	         [yY]*) continue ;;
	             *) break ;;
	    esac
	done
	```

B	Save the content of the script. Assign execute permission to the script.
	
	Run the script by entering the following codes and descriptions, respectively:
	
	`01   student`
	`02   engineer`
	`03   chef`

C	Check the content of the `newlist` file to verify the above entries.
D	**Note the `continue` command and the `break` command located near the end of the script. Explain which loop (inner loop or outer loop) do they operate on (5 marks)?**

3. UNIX Shell Scripting – The `for` Loop - Looping with a List

The `for` loop does not test a condition but uses a list instead. Like in Java or C, a `for` loop is used where the number of repetitions is generally known.

The general format of `for` loop is as follows:

```
for variable in list
do
     commands
done
```

A

Ensure that your present working directory is `Looping`, which is under your `kit104` directory.

Make a new directory named `forloop` (under `Looping`), and change into the new directory.

B

Create 3 text files which must be named as `chap20`, `chap21`, and `chap22`, respectively, by simply redirecting outputs of some `echo` commands:

```
$ echo "content of chap20" > chap20
$ echo "content of chap21" > chap21
$ echo "content of chap22" > chap22
```

Verify that the 3 files have been created with the right contents.

C

Create a script which must be named as `ftest.sh`, with the following content:

```
for file in chap20 chap21 chap22
do
     cp $file $file”.bak”
     echo $file copied to $file”.bak”
done
```

In this script, `file` is a variable, which takes `chap20` as its value first. Then the loop body is entered, which copies `chap20` into `chap20.bak`, and displays a message to confirm the operation. Then the loop is repeated, with the variable `file` taking `chap21` as value this time, making a backup copy for `chap21` and displaying a message confirming the operation. Then the loop is repeated again, with the variable `file` taking `chap22` as value, making a backup copy for `chap22` and displaying a message confirming the operation. Now that all the files in the list have been manipulated, the loop becomes broken.

This example demonstrates the concept of looping with a `for` loop.

Save the content. Assign execute permission to the script.

Run the script. Observe the outputs. Verify that all the backup files have been created.

D	What if you want to make a backup copy of each file stored under the current directory, and there are currently 300 files available (under the current directory). Do you have to specify each filename in the list explicitly? You do not want to do this!
	
	The right approach is the following:
	
	`for file in `ls``
	`do`
	`     cp $file $file".bak"`
	`     echo $file copied to $file".bak"`
	`done`
	
	Note the command substitution used in the first line with the `ls` command. No matter how many files are stored under the current directory, the output of `ls` will get all of them!
	
	Modify the first statement in your `ftest.sh` so that it uses `` `ls` `` rather than an explicit list. Remove all the existing `.bak` files under the current directory. Run the script again. You will see that it also makes a backup copy of the script itself.
	
	Remove all the `.bak` files again.

E	Think about what the following script does:
	
	`for file in *.html *.htm`
	`do`
	`     gzip $file`
	`done`

(The End)