KIT104 ICT Architecture and Operating Systems
Tutorial Week 9

1. UNIX Shell Scripting – The `if` Conditional

The if conditional can be used for complex decision making. The general syntax is as follows.

Form 1

```
if command successful
then
     execute commands
fi
```

Form 2

```
if command successful
then
     execute commands
else
     execute commands
fi
```

Form 3

```
if   command successful
then
     execute commands
elif command successful
then ...
else ...
fi
```

(You can have as many `elif … then …` as necessary)

The above forms are similar to the `if` statements in C Programming, except that an `fi` (which is `if` in reverse order) is needed to mark the end of an `if` conditional.

The following are some examples (do not type them onto your shell). The `e.lst` and `f.lst` are record files which contain some customer details.

```
if grep "john" e.lst
then
     echo “pattern found”
else
     echo “pattern not found”
fi

if grep "john" e.lst; then
     echo “pattern found in e.lst”
elif grep "john" f.lst; then
     echo “pattern found in f.lst”
else echo “pattern not found”
fi
```

Try to understand the above statements, and then perform the following actions.

A

Create a script named `emp3.sh`, with the following content:

The \c keeps the cursor on the same line after the end of the echo, but to enable it, you need the -e flag: echo -e "bla bla \c"

```
#! /bin/sh

echo  -e  "Enter the pattern to be searched: \c"
read  pname
echo  -e  "Searching for $pname\n"

if grep "$pname" e.lst; then
        echo "Pattern found in e.lst"
elif grep "$pname" f.lst; then
        echo "Pattern found in f.lst"
else echo "Pattern not found"
fi
```

Save the script. Assign execute permission to the script.

B

Create a text file named `e.lst`, with the following content:

```
2000, John Warren, NSW
2001, Adam Davis, NSW
3000, John Smith, ACT
```

Create a text file named `f.lst`, with the following content:

```
2000, Jack Williams, NSW
2001, Adam Davis, NSW
3000, Jack Swan, ACT
```

C

Ensure that your `emp3.sh`, `e.lst`, and `f.lst` are all stored under the current directory. Then run the `emp3.sh` script by specifying the pattern as `John`, `Jack`, and `Andrew`, respectively. Observe the outputs and ensure that they are correctly generated.

2. UNIX Shell Scripting – `test` and `[ ]`

The `test` command uses certain operators to evaluate a condition and returns either a true or false *exit status* - which is then used by `if` for making decisions. It works in three ways:

- Compare two numbers
- Compares two strings or a single one for a null value
- Checks a file's attributes

The command does not display any output but simply returns a value that sets the parameter `$?`.

A

Perform the following actions on your shell. For each command line, the first `$` is the shell prompt. Recall from a previous tutorial that the `$?` has the value of 0 if the command succeeds, and a non-zero (normally 1 or 2) value if it fails.

```
$ x=5; y=7; z=8

$ test $x -eq $y ; echo $?

$ test $x -lt $y ; echo $?

$ test $z -gt $y ; echo $?

$ test $z -eq $y ; echo $?
```

返回0或者非零数字如1，严重前面等式是否成立

There are 6 operators which you can use:

`-eq`	(equal to)
`-ne`	(not equal to)
`-gt`	(greater than)
`-ge`	(greater than or equal to)
`-lt`	(less than)
`-le`	(less than or equal to)

B Note you need to put whitespace on either side of an operator.

C Do you understand the following statements:

参数数量不等于3

```
if  test $# -ne 3;  then
 echo "you did not enter three arguments"
else
 echo "you entered the right number"
fi
```

Remember from a previous tutorial that, `$#` refers to the number of arguments in a command line.

D There is a shortcut for the `test` command, which is simply `[]`.

For example, `test $# -ne 3` is equivalent to

`[ $# -ne 3 ]`   (note the whitespaces after the "`[`" and before the "`]`")

Therefore, the statements from step C can become the following

```
if  [ $# -ne 3 ];  then
 echo "you did not enter three arguments"
else
 echo "you entered the right number"
fi
```

E Repeat step A by changing the `test` command to `[]`:

```
$ [ $x -eq $y ]; echo $?
$ [ $x -lt $y ]; echo $?

$ [ $z -gt $y ]; echo $?

$ [ $z -eq $y ]; echo $?
```

F The above examples show how to compare numbers on a UNIX shell. The following example demonstrates how to compare strings.

G Create the following script and name it as `compile.sh`:

```
#! /bin/sh
if [ $# -eq 1 ] ; then
     if [ $1 = “j” ] ; then
          file=`ls -t *.java | head -1`
          echo -e “The last modified Java file is $file\n”
     elif [ $1 = “c” ] ; then
          file=`ls -t *.c | head -1`
          echo -e “The last modified C file is $file\n”
     else echo “Invalid file type”
     fi
else echo -e “Usage: $0 file_type\nValid file types are c and j”
fi
```

按时间修改排序寻找第一个文件

This script stores the last modified C or Java program filename in the variable `file`. It then displays the file name. You have to provide one argument to the script - the file type, which must be `c` (for C program) or `j` (for Java program). If a user does not enter a file type, or enters a file type which is neither `c` nor `j`, then the script usage information is displayed. The usage information shows the correct way to run a script.

Remember from a previous tutorial that, a shell script can read in command-line arguments. The first argument is referred to as `$1`, the second argument is referred to as `$2`, and so on. `$0` refers to the script file name. `$#` refers to the number of arguments in the command line.

The `ls` with option `-t` displays file names by modification time. The last modified file is displayed first. The `head -1` (digit one) gets the file name listed at the top (or in the beginning).

H Prepare some testing C and Java files before you attempt to run this script. Following exercises in the precious section, make a copy of `e.lst` and name it as `t1.c`. Make a copy of `f.lst` and name it as `p1.java`. Make a copy of `t1.c` and name it as `t2.c`. Make a copy of `p1.java` and name it as `p2.java`.

I Assign execute permission to `compile.sh`. Run the scrip as follows. Ensure that you understand the outputs.

```
$ ./compile.sh
$ ./compile.sh k

$ ./compile.sh c

$ ./compile.sh j
```

J You can also use `test` or `[]` to check file attributes, which is commonly used in many shell scripts.

`[ -f $1 ]` is true if the ordinary file `$1` exists
`[ -r $1 ]` is true if the file `$1` is readable
`[ -w $1 ]` is true if the file `$1` is writable
`[ -x $1 ]` is true if the file `$1` is executable
`[ -d $1 ]` is true if the directory file `$1` exists

K Create a scrip named `emp4.sh`, with the following content:

```
if [ -f $1 ] ; then
     echo “File exists”
else
     echo “File does not exist”
fi
```

Save it. Assign execute permission to it.

L Run the script as follows:

```
$ emp4.sh t1.c
$ emp4.sh t2.c
$ emp4.sh t3.c
```

M Modify the content of `emp4.sh`, so that it becomes the following:

```
if [ ! -f $1 ] ; then
     echo “File does not exist”
elif [ ! -r $1 ]; then
      echo “File is not readable”
elif [ ! -w $1 ]; then
      echo “File is not writable”
else
     echo “File is readable and writable”
fi
```

Here the “`!`” reverses a condition, so

`[ ! -f $1 ]` is true if the ordinary file `$1` does NOT exist.
`[ ! -r $1 ]` is true if the file `$1` is NOT readable
`[ ! -w $1 ]` is true if the file `$1` is NOT writable

<table>
<tr><td></td><td><code>[! -x $1]</code> is true if the file <code>$1</code> is NOT executable</td></tr>
<tr><td>N</td><td>Run the script again as follows:

```
$ emp4.sh t1.c
$ emp4.sh t2.c
$ emp4.sh t3.c
```

</td></tr>
<tr><td>O</td><td>Remove the read permission of <code>t1.c</code> and the write permission of <code>t2.c</code>, and then run the above command lines again to observe the outputs.</td></tr>
</table>

3. UNIX Shell Scripting – The `case` conditional

The case conditional matches an expression for more than one alternative, permitting multi-way branching. The general format is this:

```
case      expression          in
          pattern1) commands1 ;;
          pattern2) commands2 ;;
          pattern3) commands3 ;;
          ......
esac
```

The `esac` here is `case` in reverse order, which marks the end of a `case` conditional. Note the double semicolons at the end of each command.

<table>
<tr><td>A</td><td>Remove all the temporary C or Java files from your current directory.

Create a scrip which is named <code>emp5.sh</code>, with the following content:

```
#! /bin/sh
tput clear
echo -e “\n 1. Find files modified in last 24 hours”
echo -e “\n 2. The free disk space”
echo -e “\n 3. Space consumed by this user”
echo -e “\n 4. Exit\n\n”
echo -e “SELECTION: \c”

read choice
case $choice in
     1) find $HOME -mtime -1 -print ;;
     2) df ;;
     3) du -s $HOME ;;
     4) exit ;;
     *) echo “Invalid option” ;;
esac
```

Here the first 4 <code>echo</code> commands generate a menu. The 5th <code>echo</code> prompts the user to make a selection (which needs to be a number in the range of 1 - 4, inclusive).</td></tr>
</table>

Depending on the user’s selection (which is stored in the variable choice), a corresponding command line is run. If the user’s selection is a character other than 1, 2, 3, 4, then the message “Invalid option” is displayed.

B

Run emp5.sh, and enter a number in the range of 1 - 4 (inclusive) to observe its output. Run the script again by entering another number. Also try to enter a number which is not in the range (such as 5) to see the output.

C

The case conditional is able to match multiple patterns or using wild cards as described in previous tutorials. Add the following statements to the end of emp5.sh:

```
echo -e “Wish to continue? (y/n): \c”
read answer
case $answer in
     Y|y) echo -e “I will do something later\n”;;
     N|n) exit ;;
       *) echo “Invalid option” ;;
esac
```

Here Y|y matches Y or y, N|n matches N or n.

Save the change and run emp5.sh again.

D

You can also use wild cards in case conditional:

```
echo -e “Wish to continue? (y/n): \c”
read answer
case $answer in
     [Yy][Ee]*) ;;     #  matches YES, yes, Yes, etc
     [Nn][Oo]) exit ;;     #  matches NO, No, no, nO
     *) echo “Invalid option” ;;
Esac
```

E

Here is another example with using wild cards in case conditional. Explain the purpose of this script (5 marks). Note: gcc is a C program compiler. Javac is a Java program compiler.

get the last modified file

```
file=`ls -t *.java *.c 2>/dev/null | head -1`

case $file in
     *.c) gcc $file ;;
     *.java) javac $file ;;
     *) echo “No Java or C program found”;;
esac
```

4. The `sleep` and `expr` commands

A

The `sleep` command introduces some delay in a shell script. Run the following command line.

```
$ sleep 3; echo “3 seconds have elapsed”
```

B

The `expr` performs basic arithmetic operations:

```
$ expr 3 + 5
$ x=3; y=5
$ expr $x - $y
$ expr 3 \* 5
$ expr $y / $x
```

expr 是数学公式的意思

The operators (`+, -, *, /`) must be enclosed on either side by whitespace. The `expr` only handles integers.

In the 4th command line above, why a “\” required before “*”?

The most common use of `expr` is for incrementing the value of a variable:

```
$ x=5
$ x=`expr $x + 1`      (note the back-quotes)
$ echo $x
$ x=5; y=2; z=`expr $x + $y`     (note the back-quotes)
```

5. UNIX Shell Scripting – Looping

Like in C or Java programming, loops allow you to perform a set of instructions repeatedly.

A

Under your `kit104` directory, make a new directory named `Looping`. Change into this new directory.

B

Create a shell script which is named as `colour.sh`, with the following content:

```
echo “Guess my favourite colour: ”
read guess
while [ “$guess” != “red” ]
do
     echo “No, not that one. Try again.”
     read guess
done
echo “Well done”
```

Here the `read` command stores the user input into variable `guess`. As long as the user input is not “red”, the loop body is entered, which displays a message and then prompts the user to enter a guess again. By the time the user input is “red” (has guessed correctly), the loop is broken and exited.

Assign execute permission to the script, and run it. To break the loop, you have to enter “red”. Alternatively you can press `CTRL-C` to stop.

C

Copy the script `emp5.sh` into the current `Looping` folder. The script is from a previous section of this tutorial. Rename `emp5.sh` as `emp6.sh`. Modify the exiting content of `emp6.sh` so that its new content are as follows:

```
#! /bin/sh
tput clear

answer=y
while [ $answer = y ]
do
     echo -e "\n 1. Find files modified in last 24 hours"
     echo -e "\n 2. The free disk space"
     echo -e "\n 3. Space consumed by this user"
     echo -e "\n 4. Exit\n\n"
     echo -e "SELECTION: \c"

     read choice
     case $choice in
     1) find $HOME -mtime -1 -print ;;
     2) df ;;
     3) du -s $HOME ;;
     4) exit ;;
     *) echo “Invalid option” ;;
     esac
done
```

Note the use of a variable named `answer` which is used to control the loop. Run the script. Can you see the purpose of the loop for this script?

D

The above script needs to be terminated by entering 4 as the user input (or pressing CTRL-C). Another way to control a loop is to use the `continue` command and the `break` command.

`break`: break out of the current loop
`continue`: start the current loop again

Add the following statements to the end of the loop body in `emp6.sh` (ie, between the `esac` line and the `done` line):

```
     echo –e “Wish to continue? (y/n): \c”
     read answer2
     case $answer2 in
     [yY]) continue ;;
        *) break ;;
     esac
```

Save the change and run `emp6.sh` again.

E

Sometimes it is necessary to set up an infinite loop. The following is such an example:

```
while true
do
     echo "message repeated every 2 seconds"
     sleep 2
done
```

Use the above statements to make a new script named `emp7.sh`. Assign execute permission to it. Run the script. You have to press `CTTRL-C` to stop it.

An infinite loop can be used for more useful purposes, eg, a system administrator can set up an infinite loop to monitor the available space in disks every few minutes.

F

What does the following program do? (5 marks)

- r FILE exists and read permission is granted
don't use if describe

```
while [ ! -r invoice.lst ]
do
     sleep 30
done

echo "That file can be read now!"
```

(The End)

lp = line printer 不能玩，实际输送到打印机