

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	<p>Create a script named <code>emp3.sh</code>, with the following content:</p> <pre>#!/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</pre> <p>Save the script. Assign execute permission to the script.</p>	<p>The <code>\c</code> keeps the cursor on the same line after the end of the echo, but to enable it, you need the <code>-e</code> flag: <code>echo -e "bla bla \c"</code></p>
B	<p>Create a text file named <code>e.lst</code>, with the following content:</p> <pre>2000, John Warren, NSW 2001, Adam Davis, NSW 3000, John Smith, ACT</pre> <p>Create a text file named <code>f.lst</code>, with the following content:</p> <pre>2000, Jack Williams, NSW 2001, Adam Davis, NSW 3000, Jack Swan, ACT</pre>	
C	<p>Ensure that your <code>emp3.sh</code>, <code>e.lst</code>, and <code>f.lst</code> are all stored under the current directory. Then run the <code>emp3.sh</code> script by specifying the pattern as John, Jack, and Andrew, respectively. Observe the outputs and ensure that they are correctly generated.</p>	

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

	<pre>\$ x=5; y=7; z=8</pre> <pre>\$ test \$x -eq \$y ; echo \$?</pre> <pre>\$ test \$x -lt \$y ; echo \$?</pre> <pre>\$ test \$z -gt \$y ; echo \$?</pre> <pre>\$ test \$z -eq \$y ; echo \$?</pre> <p>There are 6 operators which you can use:</p> <ul style="list-style-type: none"> -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) 	返回0或者非零数字如1，严重前面等式是否成立
B	Note you need to put whitespace on either side of an operator.	
C	<p>Do you understand the following statements:</p> <pre>if test \$# -ne 3; then echo "you did not enter three arguments" else echo "you entered the right number" fi</pre> <p>Remember from a previous tutorial that, <code>\$#</code> refers to the number of arguments in a command line.</p>	参数数量不等于3
D	<p>There is a shortcut for the test command, which is simply <code>[]</code>.</p> <p>For example, <code>test \$# -ne 3</code> is equivalent to</p> <pre>[\$# -ne 3]</pre> <p>(note the whitespaces after the “[” and before the “”)</p> <p>Therefore, the statements from step C can become the following</p> <pre>if [\$# -ne 3]; then echo "you did not enter three arguments" else echo "you entered the right number" fi</pre>	
E	<p>Repeat step A by changing the test command to <code>[]</code>:</p> <pre>\$ [\$x -eq \$y]; echo \$?</pre>	

	<pre>\$ [\$x -lt \$y]; echo \$?</pre> <pre>\$ [\$z -gt \$y]; echo \$?</pre> <pre>\$ [\$z -eq \$y]; echo \$?</pre>
F	The above examples show how to compare numbers on a UNIX shell. The following example demonstrates how to compare strings.
G	<p>Create the following script and name it as <code>compile.sh</code>:</p> <pre>#!/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</pre> <p>This script stores the last modified C or Java program filename in the variable <code>file</code>. It then displays the file name. You have to provide one argument to the script - the file type, which must be <code>c</code> (for C program) or <code>j</code> (for Java program). If a user does not enter a file type, or enters a file type which is neither <code>c</code> nor <code>j</code>, then the script usage information is displayed. The usage information shows the correct way to run a script.</p> <p>Remember from a previous tutorial that, a shell script can read in command-line arguments. The first argument is referred to as <code>\$1</code>, the second argument is referred to as <code>\$2</code>, and so on. <code>\$0</code> refers to the script file name. <code>\$#</code> refers to the number of arguments in the command line.</p> <p>The <code>ls</code> with option <code>-t</code> displays file names by modification time. The last modified file is displayed first. The <code>head -1</code> (digit one) gets the file name listed at the top (or in the beginning).</p> <p>按时间修改排序寻找第一个文件</p>
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 <code>e.lst</code> and name it as <code>t1.c</code> . Make a copy of <code>f.lst</code> and name it as <code>p1.java</code> . Make a copy of <code>t1.c</code> and name it as <code>t2.c</code> . Make a copy of <code>p1.java</code> and name it as <code>p2.java</code> .
I	<p>Assign execute permission to <code>compile.sh</code>. Run the scrip as follows. Ensure that you understand the outputs.</p> <pre>\$./compile.sh</pre>

	<pre>\$./compile.sh k \$./compile.sh c \$./compile.sh j</pre>
J	<p>You can also use <code>test</code> or <code>[]</code> to check file attributes, which is commonly used in many shell scripts.</p> <pre>[-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</pre>
K	<p>Create a scrip named <code>emp4.sh</code>, with the following content:</p> <pre>if [-f \$1] ; then echo "File exists" else echo "File does not exist" fi</pre> <p>Save it. Assign execute permission to it.</p>
L	<p>Run the script as follows:</p> <pre>\$ emp4.sh t1.c \$ emp4.sh t2.c \$ emp4.sh t3.c</pre>
M	<p>Modify the content of <code>emp4.sh</code>, so that it becomes the following:</p> <pre>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</pre> <p>Here the “!” reverses a condition, so</p> <pre>[! -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</pre>

	[! -x \$1] is true if the file \$1 is NOT executable
N	Run the script again as follows: \$ emp4.sh t1.c \$ emp4.sh t2.c \$ emp4.sh t3.c
O	Remove the read permission of t1.c and the write permission of t2.c, and then run the above command lines again to observe the outputs.

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.

A	<p>Remove all the temporary C or Java files from your current directory.</p> <p>Create a scrip which is named emp5.sh, with the following content:</p> <pre> #!/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 </pre> <p>Here the first 4 echo commands generate a menu. The 5th echo prompts the user to make a selection (which needs to be a number in the range of 1 - 4, inclusive).</p>
---	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	Depending on the user's selection (which is stored in the variable <code>choice</code>), 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 <code>emp5.sh</code> , 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	<p>The <code>case</code> conditional is able to match multiple patterns or using wild cards as described in previous tutorials. Add the following statements to the end of <code>emp5.sh</code>:</p> <pre>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</pre> <p>Here <code>Y y</code> matches Y or y, <code>N n</code> matches N or n.</p> <p>Save the change and run <code>emp5.sh</code> again.</p>
D	<p>You can also use wild cards in <code>case</code> conditional:</p> <pre>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</pre>
E	<p>Here is another example with using wild cards in <code>case</code> conditional. Explain the purpose of this script (5 marks). Note: gcc is a C program compiler. Javac is a Java program compiler.</p> <pre>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</pre>

4. The **sleep** and **expr** commands

A	<p>The sleep command introduces some delay in a shell script. Run the following command line.</p> <pre>\$ sleep 3; echo "3 seconds have elapsed"</pre>
B	<p>The expr performs basic arithmetic operations:</p> <pre>\$ expr 3 + 5 \$ x=3; y=5 \$ expr \$x - \$y \$ expr 3 * 5 \$ expr \$y / \$x</pre> <p>expr 是数学公式的意思</p> <p>The operators (+, -, *, /) must be enclosed on either side by whitespace. The expr only handles integers.</p> <p>In the 4th command line above, why a “\” required before “*”? </p> <p>The most common use of expr is for incrementing the value of a variable:</p> <pre>\$ x=5 \$ x=`expr \$x + 1` (note the back-quotes) \$ echo \$x \$ x=5; y=2; z=`expr \$x + \$y` (note the back-quotes)</pre>

5. UNIX Shell Scripting – Looping

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

A	<p>Under your kit104 directory, make a new directory named Looping. Change into this new directory.</p>
B	<p>Create a shell script which is named as colour.sh, with the following content:</p> <pre>echo "Guess my favourite colour: " read guess while ["\$guess" != "red"] do echo "No, not that one. Try again." read guess done echo "Well done"</pre> <p>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</p>

	<p>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.</p> <p>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.</p>
C	<p>Copy the script <code>emp5.sh</code> into the current <code>Looping</code> folder. The script is from a previous section of this tutorial. Rename <code>emp5.sh</code> as <code>emp6.sh</code>. Modify the exiting content of <code>emp6.sh</code> so that its new content are as follows:</p> <pre>#!/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</pre> <p>Note the use of a variable named <code>answer</code> which is used to control the loop. Run the script. Can you see the purpose of the loop for this script?</p>
D	<p>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 <code>continue</code> command and the <code>break</code> command.</p> <p><code>break</code>: break out of the current loop <code>continue</code>: start the current loop again</p> <p>Add the following statements to the end of the loop body in <code>emp6.sh</code> (ie, between the <code>esac</code> line and the <code>done</code> line):</p> <pre>echo -e "Wish to continue? (y/n): \c" read answer2 case \$answer2 in [yY]) continue ;; *) break ;;</pre>

	<p>esac</p> <p>Save the change and run <code>emp6.sh</code> again.</p>
E	<p>Sometimes it is necessary to set up an infinite loop. The following is such an example:</p> <pre>while true do echo "message repeated every 2 seconds" sleep 2 done</pre> <p>Use the above statements to make a new script named <code>emp7.sh</code>. Assign execute permission to it. Run the script. You have to press <code>CTRL-C</code> to stop it.</p> <p>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.</p>
F	<p>What does the following program do? (5 marks)</p> <pre>while [! -r invoice.lst] do sleep 30 done</pre> <p>echo "That file can be read now!"</p>

(The End)

lp = line printer 不能玩，实际输送到打印机