Chapter 12 Notes

I. Working with the if-then Statement

A. The most basic type of structured command is the if-then statement. The if-then statement has the following format:

```
if command
then
        commands
fi
```

1. Ex:

a)
```
$ cat test1.sh
#!/bin/bash
# testing the if statement
if pwd
then
        echo "It worked"
fi
$
```

b) Output:

(1)
```
$ ./test1.sh
/home/Christine
It worked
$
```

B. If the exit status of the command (see Chapter 11) is zero (the command completed successfully), the commands listed under the then section are executed

1. If the exit status of the command is anything else, the then commands aren't executed, and the bash shell moves on to the next com-mand in the script

C. The fi statement delineates the if-then statement's end

D. You are not limited to just one command in the then section. You can list commands just as in the rest of the shell script. The bash shell treats the commands as a block, executing all of them when the command in the if statement line returns a zero exit status or skipping all of them when the command returns a non-zero exit status

II. Exploring the if-then-else Statement

A. The if-then-else statement provides another group of commands in the statement:

1. if command
   then
   commands
   else
   commands
   fi

B. When the command in the if statement line returns with a zero exit status code, the commands listed in the then section are executed, just as in a normal if-then statement. When the command in the if statement line returns a non-zero exit status code, the bash shell executes the commands in the else section

1. $ cat test4.sh
   #!/bin/bash# testing the else section
   #testuser=NoSuchUser
   #
   if grep $testuser /etc/passwd
   then
   echo "The bash files for user $testuser are:"
   ls -a /home/$testuser/.b*
   echo
   else
   echo "The user $testuser does not exist on this system."
   echo
   fi
   $
2. Output:
   a) $ ./test4.sh
   The user NoSuchUser does not exist on this system.
   $

III. Nesting ifs

A. Sometimes, you must check for several situations in your script code. For these situations, you can nest the if-then statements

B. Instead of having to write separate if-then statements, you can use an alternative version of the else section, called elif. The elif continues an else section with another if-then statement:

1. if command1
   then
   commands
   elif command2
   then
   more commands
   fi

C. Keep in mind that, with an elif statement, any else statements immediately following it are for that elif code block. They are not part of a preceding if-then statement code block

IV. Trying the test Command

A. If-then statements can only evaluate based on the exit status code

B. The test command provides a way to test different conditions in an if-then statement

1. If the condition listed in the test command evaluates to TRUE, the test command exits with a zero exit status code
2. if test condition
   then
   commands
   fi

C. The bash shell provides an alternative way of testing a condition without declaring the test command in an if-then statement:

1. if [condition]
   then
   commands
   fi
2. Be careful; you must have a space after the first bracket and a space before the last bracket, or you'll get an error message

D. The test command and test conditions can evaluate three classes of conditions:

1. Numeric comparisons
    a) The test Numeric Comparisons
        (1) **n1 -eq n2**: Checks if n1 is equal to n2
        (2) **n1 -ge n2**: Checks if n1 is greater than or equal to n2
        (3) **n1 -gt n2**: Checks if n1 is greater than n2
        (4) **n1 -le n2**: Checks if n1 is less than or equal to n2
        (5) **n1 -lt n2**: Checks if n1 is less than n2
        (6) **n1 -ne n2**: Checks if n1 is not equal to n2
    b) The only numbers the bash shell can handle are integers
2. String comparisons
    a) The test String Comparisons
        (1) **str1 = str2**: Checks if str1 is the same as string str2
        (2) **str1 != str2**: Checks if str1 is not the same as str2
        (3) **str1 < str2**: Checks if str1 is less than str2
        (4) **str1 > str2**: Checks if str1 is greater than str2
        (5) **-n str1**: Checks if str1 has a length greater than zero
        (6) **-z str1**: Checks if str1 has a length of zero
    b) The test comparison takes all punctuation and capitalization into account when comparing strings for equality
    c) Two problems often plague shell programmers when trying to use the greater-than or less-than features of test conditions:
        (1) The greater-than and less-than symbols must be escaped, or the shell uses them as redirection symbols, with the string values as filenames
        (2) The greater-than and less-than order is not the same as that used with the sort command
            (a) Capitalized letters are treated as less than lowercase letters in test comparisons. However, the sort command does the opposite

3. File comparisons

a) The test File Comparisons

(1) -**d file**: Checks if file exists and is a directory

(2) **-e file**: Checks if file exists

(3) **-f file**: Checks if file exists and is a file

(4) **-r file**: Checks if file exists and is readable

(5) **-s file**: Checks if file exists and is not empty

(6) **-w file**: Checks if file exists and is writable

(7) **-x file**: Checks if file exists and is executable

(8) **-O file**: Checks if file exists and is owned by the current user

(9) **-G file**: Checks if file exists and the default group is the same as the current user

(10) **file1 -nt file2**: Checks if file1 is newer than file2

(11) **file1 -ot file2**: Checks if file1 is older than file2

b) The -e comparison works for both files and directories. To be sure that the object specified is a file and not a directory, you must use the -f comparison

c) The -G comparison checks the default groups only and not all the groups to which the user belongs

V. Considering Compound Testing

A. The if-then statement allows you to use Boolean logic to combine tests. You can use these two Boolean operators:

1. [condition1] && [condition2]

a) AND: Both conditions must be met for the then section to execute

2. [condition1] || [condition2]

a) OR: If either condition evaluates to a TRUE condition, the then section is executed

VI. Working with Advanced if-then Features

A. Two additions to the bash shell provide advanced features that you can use in if-then statements:

1. Double parentheses for mathematical expressions

a) ((expression))

b) The Double Parentheses Command Symbols

(1) **val++**: Post-increment

(2) **val--**: Post-decrement

(3) **++val**: Pre-increment

(4) **--val**: Pre-decrement

(5) **!**: Logical negation

(6) **~**: Bitwise negation

(7) ******: Exponentiation

(8) **<<**: Left bitwise shift

(9) **>>**: Right bitwise shift

(10) **&**: Bitwise Boolean AND

(11) **|**: Bitwise Boolean OR

(12) **&&**: Logical AND

(13) **||**: Logical OR

c) You can use the double parentheses command in an if statement, as well as in a normal command in the script for assigning values

d) You don't need to escape the greater-than symbol in the expression within the dou-ble parentheses

2. Double square brackets for advanced string handling functions

a) [[expression]]

b) The double bracketed expression uses the standard string comparison used in the test evaluations. However, it provides an additional feature that the test evaluations don't — pattern matching

c) In pattern matching, you can define a regular expression (discussed in detail in Chapter 20) that's matched against the string value

d) $ cat test24.sh

```
#!/bin/bash
# using pattern matching
#
if [[ $USER == r* ]]
then
        echo "Hello $USER"
else
        echo "Sorry, I do not know you"
fi
$
$ ./test24.sh
Hello rich
$
```

(1) Notice in the preceding script that double equal signs (==) are used. These double equal signs designate the string to the right (r*) as a pattern, and pattern matching rules are applied

(2) The double bracket command matches the $USER environment variable to see whether it starts with the letter r. If so, the comparison succeeds, and the shell executes the then section commands

VII. Considering the case Command

A. Instead of having to write all the elif statements to continue checking the same variable value, you can use the case command. The case command checks multiple values of a single variable in a list-oriented format:

1. case variable in
   pattern1 | pattern2) commands1;;
   pattern3) commands2;;
   *) default commands;;
   esac

B. The case command compares the variable specifi ed against the different patterns. If the variable matches the pattern, the shell executes the commands specifi ed for the pattern

C. You can list more than one pattern on a line, using the bar operator to separate each pat-tern
D. The asterisk symbol is the catch-all for values that don’t match any of the listed patterns.
E. Ex:
    1. $ cat test26.sh

```
#!/bin/bash
# using the case command
#
case $USER in
rich | barbara)
        echo "Welcome, $USER"
        echo "Please enjoy your visit";;
testing)
        echo "Special testing account";;
jessica)
        echo "Do not forget to log off when you're done";;
*)
        echo "Sorry, you are not allowed here";;
esac
$
```

    2. Output:
        a) $ ./test26.sh

```
Welcome, rich
Please enjoy your visit
$
```