Laboratory Experiment V

Shell Scripting I

by Alex Shaw III

CIS 247, Section ZG81
Professor Braithwaite, R.

May 17, 2005

TABLE OF CONTENTS

INTRODUCTION

1
LABORATORY SHEETS

LABORATORY EXPERIMENT V

Shell Scripts

Introduction

We humans find constant repetition boring. And boredom leads to poor performance. Computers, by contrast, don't know the difference. Tell your computer to do the same job four thousand times, and you'll hear nary a whimper.

The Linux **shell script** is a means by which you can get the computer to perform repetitive tasks. The **shell script** is also a valuable tool for dealing with complicated command lines. If you find yourself using the **find** command with the **-exec** option often, for example, using a **shell script** to accomplish this task will remove the necessity of retyping and require that you enter that complicated command line only once.

In This Module You Will Learn:

- the purpose behind shell scripts.
- how to construct simple shell scripts.
- how the use of **variables** can greatly increase the usefulness of your scripts.

As we did with the Try It Live! exercises in the *Security* chapter of this guide, we will omit step-by-step instructions for some of the exercises. If you need some help refer to the Try It Live! Answer Key at the back of this guide. You will find whatever step-by-step instructions we have omitted in the exercises.

1. Create a subdirectory under your directory **practice**. Call this new directory **scripts**. You will use **scripts** to store the shell scripts you create in these exercises.
2. Change to the directory **scripts**.
3. In this exercise, you will create the first shell script presented in the *Linux Made Easy* CBT. This script will pipe the output of the **ls -l** command into the utility **more**. Name the file **list**.
   a. Type **cat > list**
   b. Press **<Enter>**
   c. Type **ls -l | more**
   d. Press **<Ctrl-D>**
4. Check your work. Use **cat** to read the file **list**. Your screen should look like Figure 9-1.

```
ls -l | more
```

Figure 9-1 Text of the Script *list*

If you typed something incorrectly, you will have to recreate the file. As we pointed out in the *Linux Files* chapter of this guide, if your system permits, use **cat > list** to recreate the file. If your system returns an error message similar to `list: file exists` when you try this, you will have to delete the file with **rm** before you recreate it.

5. Make **list** executable by user, group, and others.
   a. Type **chmod a+x list**
   b. Press **<Enter>**

   The default mode for a newly created file is usually

   -rw-r--r--

   For a shell script to run, the permissions must include read *and* execute; execute permission alone is not sufficient.
6. List your directory **scripts** to be certain that the file mode for **list** includes both read and execute permission. Your screen should look similar to Figure 9-2.

```
-rwxr-xr-x 1 ronsm 12  Fri Nov 17 9:12 list
$ |
```

Figure 9-2 Directory Listing Showing Mode for *list*

If user, group, and others do not have read permission for **list**, you must add that permission before someone in that user category (u, g, o) tries to run the script.

7. Run your new shell script, **list**.
   a. Type **list**
   b. Press **<Enter>**

Did your script run? If it did not run and the permissions are set correctly, there are three other methods to try.

1. You may have received an error message, something like "Command not found." In simple terms, this happens because the shell doesn't know that it is supposed to look for executable files in the working directory. The simplest solution, then, is to tell the shell, "Look here!" To do that, precede the name of the shell script with **./**.

   This means that the command line in step a of Exercise 7 will now look like this:

   ./list

   There is no space between the forward slash and the filename.
2. Run the script with the "dot command." This means that the command line in step a of Exercise 7 will now look like this:

   . list

   There *is* a space between the dot and **list**.
3. Run the script with the **sh** command. This means that the command line in step a of Exercise 7 will now look like this:

   sh list

continue on next page…

The script **list** is quite simple in both its construction and what it does. Yet it requires nine fewer keystrokes than the command line that it replaces — **ls -l | more**. If you use that command line often, the nine keystrokes saved with each use of **list** become significant.

list works only in its own directory. We can make **list** more useful by adding the ability to specify which directory to list. To do this, we will use a variable.

8. Create a shell script that incorporates **ls –l | more**, and a prompt for the specific directory to list. Call this file **list2**.
   a. Type **cat > list2**
   b. Press **<Enter>**
   c. Type **echo -n "Which directory do you want to list? "**
   d. Press **<Enter>**
   e. Type **read dir**
   f. Press **<Enter>**
   g. Type **ls -l $dir | more**
   h. Press **<Ctrl-D>**
9. The script **list2** is a text file. Display the contents of the file on standard output.

   Your screen should look like Figure 9-3.

```
echo -n "Which directory do you want to list? "
read dir
ls -l $dir | more
```

Figure 9-3 Text of the Shell Script *list2*

The command **echo** tells the shell to display the text enclosed in double quotes. The "**-n**" at the beginning of the line keeps the cursor from moving to the next line; your response to the question "Which directory do you want to list?" will be on the same line as the question. The command **read** tells the system to take whatever you entered in response to that question and place that response in the variable **dir**. The third line is the familiar command, **ls -l**. The directory that you specified now appears as the argument to **ls -l**. When you run the script, if you respond to the question with "/etc", **$dir** becomes "**/etc**", and **ls -l $dir** becomes **ls -l /etc**.

Shell script programming syntax generally requires that a variable, **dir** in this case, be preceded by a "**$**" when the variable is accessed—that is, when the *value* (the content) of the variable is used. That is why **dir** becomes **$dir** in the last line of the script: The *value* of the variable **dir** is used in this line. This means that, when the script executes the line `ls -l $dir | more`, the system replaces **$dir** with whatever you entered in response to the question echoed in the first line of the script, "Which directory do you want to list?"

If you made an error in creating the script **list2**, you should correct the error before running the script. Refer to the comment following Exercise 4.

10. Make the script **list2** an executable file for all.
11. Run the shell script **list2**. Respond to the directory question with **/etc**.
    a. Type **list2**
    b. Press **<Enter>**
    c. Type **/etc**
    d. Press **<Enter>**

    On most systems, the directory **/etc** contains many entries. The listing will fill the screen and stop until you press one of the two keys necessary to continue the listing. (If you aren't certain what two keys we are talking about, see the Try It Live! Answer Key at the very end of this guide.)

Reminder: If the script does not run after you have set the permissions properly, use one of the alternative methods described in the paragraphs following Exercise 7.

12. In this exercise, you will create the shell script **ren** that we presented in the *Linux Made Easy* CBT.
    a. Type **cat > ren**
    b. Press **<Enter>**
    c. Type **echo -n "Name of file you want to rename. "**
    d. Press **<Enter>**
    e. Type **read oldname**
    f. Press **<Enter>**
    g. Type **echo -n "New name of file. "**
    h. Press **<Enter>**
    i. Type **read newname**
    j. Press **<Enter>**
    k. Type **mv $oldname $newname**
    l. Press **<Enter>**
    m. Type **echo "File $oldname changed to $newname. "**
    n. Press **<Ctrl-D>**
13. Check your work by displaying the contents of the file **ren** on standard output. Your screen should look like Figure 9-4.

```
echo -n "Name of file you want to rename. "
read oldname
echo -n "New name of file. "
read newname
mv $oldname $newname
echo "File $oldname changed to $newname. "
```

Figure 9-4 Text of Script *ren*

continue on next page…

14. Make the script **ren** executable by all.
15. The only files in this directory **scripts** are the scripts themselves. For this exercise, create a file to use with the script **ren**. Name the file **thisfile**. Use the following text for the file: "This is the text for a practice file." (Or make up your own text.) Use **cat** and redirection to create the file.
16. List your working directory, **scripts**, to be certain that **thisfile** exists. Your screen should look like either Figure 9-5 or Figure 9-6.

```
list    list2    ren     thisfile
```

Figure 9-5 List of Files in the Directory *scripts*

```
-rwxr-xr-x  1  ronsm  15   Sep 19 8:12 list
-rwxr-xr-x  1  ronsm  78   Sep 19 8:15 list2
-rwxr-xr-x  1  ronsm  154  Sep 19 8:21 ren
-rwxr--r--  1  ronsm  22   Sep 19 8:25 thisfile
```

Figure 9-6 List of Files in the Directory *scripts*

17. Run the script **ren**. Change the name of that new file from **thisfile** to **thatfile**.
18. List the directory **scripts** to be certain that the filename has been changed.

For the final exercise in this module, you will create a script that adds a tiny bit of automation to **grep**. To put the script to use, you will use the supply files you created as Robert's intern. Those files are in the directory **catalog**. Figure 9-7 displays the directory structure you have created so far.

```
/home/userid
   /practice
        /backup
        /catalog
        /dir1
        /dir2
        /scripts
```

Figure 9-7 Directory Structure You have Created

19. Change to the directory **catalog**. Use the double dot notation (**..**) to make this change.
20. Create a script that uses **grep**. Call the script **srch**.
    a. Type **cat > srch**
    b. Press **<Enter>**
    c. Type **echo -n "What do you want to find? "**
    d. Press **<Enter>**
    e. Type **read thing**
    f. Press **<Enter>**
    g. Type **grep $thing ***
    h. Press **<Ctrl-D>**
21. Check your work by displaying the contents of the script **srch** on standard output. Your screen should look like Figure 9-8.

```
echo -n "What do you want to find? "
read thing
grep $thing *
```

Figure 9-8 Text of Script *srch*

22. Make the file **srch** executable for all.
23. Run the script **srch**. In response to the question "What do you want to find?", answer "pens", "file", or "specialty".

    If you searched for "pens," your screen should look like Figure 9-9. If you searched for "file," your screen should look like Figure 9-10. If you searched for "specialty," your screen should look like Figure 9-11. (Note that these figures show only partial screen displays.)

```
What do you want to find? pens
sorted:correction pens
.
.
.
supply1:writing -- pens.
supply3:correction pens
$
```

Figure 9-9 System Response with Script *srch* for "pens"

```
What do you want to find? file
sorted:folders -- file
.
.
.
supply9:folders -- file
supply9:folders -- hanging file
supply9:folders -- specialty file
$
```

Figure 9-10 System Response with Script *srch* for "file"

```
What do you want to find? specialty
sorted:folders -- specialty file
.
.
.
supply6:staplers -- specialty
supply7:tape -- specialty
supply9:folders -- specialty file
$
```

Figure 9-11 System Response with Script *srch* for "specialty"

2
DATA SHEETS

LABORATORY EXPERIMENT V

Results

3
SOURCE CODE

LABORATORY EXPERIMENT V

```
ls -l more
```

Code Listing 3-1 *list* file

```
echo -n "Which directory do you want to list? "
read dir
ls -l $dir | more
```

Code Listing 3-2 *list2* file

```
echo -n "Name of file you want to rename. "
read oldname
echo -n "New name of file. "
read newname
mv $oldname $newnmae
echo "File $oldname changed to $newname."
```

Code Listing 3-3 *ren* file

```
echo -n "What do you want to find?"
read thing
grep $thing
```

Code Listing 3-4 *srch* file