

CHAPTER 8, LAB 1: WRITING AND EXECUTING A SHELL SCRIPT (20 MINUTES)

LEARNING OBJECTIVES AND OUTCOMES

In this lab you will learn to write and execute a shell script that includes comments. You will use `chmod` to make the file that holds the script executable and include a line that starts with `#!` in the script to make sure `bash` executes it. This lab also provides an introduction to positional parameters.

READING

Read “Writing and Executing a Simple Shell Script” on pages 284–289 of Sobell.

PROCEDURE

1. Use `vim` or `cat` (see page 20 of this lab manual for instructions) to create a file named `short` with the following line in it:

```
echo 'hi there'
```

2. Use `cat` to verify the contents of `short` and then try to execute it. Use `ls -l` to display the permissions for `short`. Read the tip on Sobell, page 286.

```
$ cat short
echo 'hi there'

$ ./short
-bash: ./short: Permission denied

$ ls -l short
-rw-r--r--. 1 max pubs 16 10-19 13:31 short
```

3. Use `chmod` (Sobell, pages 99 and 285) to make the file executable, display the permissions for `short`, and try executing the file again.

```
$ chmod u+x short
$ ls -l short
-rwxr--r--. 1 max pubs 16 10-19 13:31 short
```

or

```
$ chmod 755 short
$ ls -l short
-rwxr-xr-x. 1 max pubs 16 10-19 13:31 short

$ ./short
hi there
```

4. Add a line that starts with `#!` (Sobell, page 287) to the beginning of `short` to make sure it is executed by `bash`.

```
$ cat short
#!/bin/bash
echo 'hi there'
```

5. Add a comment line (Sobell, page 288) to `short` that explains what the script does.

```
$ cat short
#!/bin/bash
# This script sends the string 'hi there' to standard output
echo 'hi there'
```

6. Within a shell script, the shell expands `$1` (a variable called a *positional parameter*; Sobell, page 462) to the first argument on the command line the script was called with. Write and execute a script named `first` that displays (sends to standard output) the first argument on the command line it was called with. Include the `#!` line and a comment. Remember to make the file executable.

```
$ cat first
#!/bin/bash
#
# This script sends its first argument to standard output
#
echo $1

$ ./first Hello
Hello
```

7. Write a shell script that copies the file named by its first argument to a file with the same name with the filename extension of `.bak`. Thus, if you call the script with the argument `first` (and a file named `first` exists in the working directory), after the script runs you would have two files: `first` and `first.bak`. Demonstrate that the script works properly.

```
$ cat cptobak
#!/bin/bash
#
# This script copies the file named by its first argument
# to the same name with a filename extension of .bak.
#
cp $1 $1.bak

$ ls
cptobak  first  short

$ ./cptobak first

$ ls
cptobak  first  first.bak  short
```

8. Read the caution titled “Always quote positional parameters” on page 462 of Sobell. Use `touch` to create a file whose name has a `SPACE` in it. What happens when you give that filename as an argument to `cptobak` from the previous step?

```
$ touch aa\ bb
$ ls -l aa*
-rw-rw-r-- 1 mark mark 0 Jan 15 16:35 aa bb
$ ./cptobak aa*
cp: target `bb.bak' is not a directory
```

The shell passes one argument (that includes a `SPACE`) to `cptobak`: `aa bb`. But when the shell that is running `cptobak` parses `$1`, it expands it to two arguments because it contains an unquoted `SPACE`; the shell passes to `cp` four arguments:

```
cp aa bb aa bb.bak
```

When you pass more than two arguments to `cp`, the last argument must be a directory: `cp` copies the files named by the first arguments into the directory named by the last argument. If the last argument is not a directory, `cp` displays an error message. For more information see the second format of `cp` on Sobell, page 754.

Modify the `cptobak` script from the previous step by quoting the positional parameters in the `cp` line. Now what happens when you use the script to make a copy of a file with a `SPACE` in its name?

```
$ cat cptobak
...
cp "$1" "$1.bak"

$ ./cptobak aa*
$ ls aa*
aa bb  aa bb.bak
```

DELIVERABLES

This lab gives you practice writing and executing shell scripts.

CHAPTER 8, LAB 2: SHELL PARAMETERS AND VARIABLES (15 MINUTES)

LEARNING OBJECTIVES AND OUTCOMES

In this lab you will learn about user-created variables and keyword variables.

READING

Read “Parameters and Variables” on page 300 of Sobell up to “Pathname expansion in assignments” on page 303.

PROCEDURE

Although variables are mostly used in scripts and read by programs, you can experiment with them on the command line.

1. Assign your name to the variable named **myname** and use **echo** to display the value of **myname** when it is unquoted, quoted using double quotation marks, and quoted using single quotation marks. (Refer to “Parameter substitution” on page 302 of Sobell and “Quoting the \$” on page 302 of Sobell.)

```
$ myname=Max
$ echo $myname
Max
$ echo "$myname"
Max
$ echo '$myname'
$myname
```

2. Use the **readonly** (Sobell, page 305) builtin to make the **myname** variable you created in the previous step a readonly variable and then assign a new value to it. What happens?

```
$ readonly myname
$ myname=Sam
-bash: myname: readonly variable
```

3. What is the value of your **HOME** (Sobell, page 307) keyword variable?

```
$ echo $HOME
/home/max
```

Demonstrate that the tilde (~; Sobell, page 307) holds the same value as **HOME**. List the contents of your home directory using a tilde.

```
$ echo ~
/home/max
```

```
$ ls ~
cptobak  first  first.bak  short
```

4. The **PATH** (Sobell, page 308) keyword variable specifies the directories in the order bash should search them when it searches for a script or program you run from the command line. What is the value of your **PATH** variable?

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/sbin
```

Append the absolute pathname of the **bin** directory that is a subdirectory of your home directory to the **PATH** variable. What does this change allow you to do more easily?

```
$ PATH=$PATH:/home/max/bin

$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/sbin:/home/max/bin
```

Adding the **~/bin** directory to **PATH** allows you to run scripts in that directory by just typing a simple filename (you do not need to type a pathname that includes a slash).

5. The **PS1** (Sobell, page 309) keyword variable holds the value of your primary shell prompt. Change the value of this variable so that your prompt is simply a **\$** followed by a **SPACE** when you are working as yourself and a **#** followed by a **SPACE** when you are working with **root** privileges.

```
$ PS1'\$ '
```

6. The **date** (Sobell, page 62) utility displays the date and time. Write and execute a shell script that displays the date and time, the name of your home directory, and the value of your **PATH** variable.

```
$ cat mine
#!/bin/bash
# This script displays time and date,
# the name of your home directory, and
# the value of your PATH variable.
date
echo $HOME
echo $PATH

$ ./mine
Fri Oct 19 16:24:43 PDT 2012
/home/max
/usr/local/bin:/bin:/usr/bin:/sbin:/home/max/bin
```

DELIVERABLES

This lab gives you practice working with user-created variables and the **HOME**, **PATH**, and **PS1** keyword variables as well as practice using the **date** utility.