Answers to Even-numbered Exercises

9

2. What are two ways you can execute a shell script when you do not have execute access permission to the file containing the script? Can you execute a shell script if you do not have read access permission?

   You can give the name of the file containing the script as an argument to the shell (for example, **sh *scriptfile***, where ***scriptfile*** is the name of the file containing the script).

   Under bash you can give the following command:

   ```
   $ . scriptfile
   ```

   Because the shell must read the commands from the file containing a shell script before it can execute the commands, you must have read permission for the file in order to execute the shell script.

4. Assume that you have made the following assignment:

   ```
   $ person=jenny
   ```

   Give the output of each of the following commands:

   a. **echo $person**

   jenny

   b. **echo '$person'**

   $person

   c. echo "$person"

   jenny

6. Assume that the **/home/jenny/grants/biblios** and **/home/jenny/biblios** directories exist. For both (a) and (b), give Jenny's working directory after she executes the sequence of commands given. Explain.

   a.

```
$ pwd
/home/jenny/grants
$ CDPATH=$(pwd)
$ cd
$ cd biblios
```

   After executing the preceding commands, Jenny's working directory is **/home/jenny/grants/biblios**.

   When **CDPATH** is set, cd searches only the directories specified by **CDPATH**; cd does not search the working directory unless the working directory is specified in **CDPATH** (by using a period).

   b.

```
$ pwd
/home/jenny/grants
$ CDPATH=$(pwd)
$ cd $HOME/biblios
```

   After executing the preceding commands, Jenny's working directory is **/home/jenny/biblios** because, when you give cd an absolute pathname as an argument, cd does not use **CDPATH**.

8. Give the following command:

```
$ sleep 30 | cat /etc/inittab
```

   Is there any output from sleep? Where does cat get its input from? What has to happen before you get a prompt back?

   There is no output from sleep (try giving the command **sleep 30** by itself). The **/etc/inittab** file provides input for cat (when cat has an argument, it does not check its standard input). The sleep command has to run to completion before you get another prompt.

10. Write a shell script that outputs the name of the shell that is executing it.

    There are many ways to solve this problem. The following solutions are all basically the same. These scripts take advantage of the **PPID** shell variable which holds the **PID** of the shell that is the parent of the process using the variable and of the fact that echo changes multiple sequential SPACEs to a single SPACE. The cut utility interprets multiple sequential SPACEs as multiple delimiters, so, without echo, the script does not work properly.

```
$ cat a
pid=$PPID
line=$(ps | grep $pid)
echo $line | cut --delimiter=" " --fields=4
```

```
$ cat a2
pid=$PPID
echo $(ps | grep $pid) | cut --delimiter=" " --fields=4

$ cat a3
echo $(ps | grep $PPID) | cut --delimiter=" " --fields=4
```

The easy solution is to put the following line in the shell script:

```
echo $0
```

The **$0** is the first command line token, which is usually the name of the script that is running. In some cases, such as when you call the script with a relative or absolute pathname, this may not be exactly what you want.

12. The following is a modified version of the **read2** script from page 281. Explain why it behaves differently. For what type of input does it produce the same output as the original **read2** script?

```
$ cat read2
echo -n "Enter a command: "
read command
"$command"
echo "Thanks"
```

In the preceding **read2** script, the string **$command** is surrounded by double quotation marks, causing it to pass a single argument to the shell. The script in the text does not surround the string with quotation marks and therefore passes the shell as many arguments as the user enters. The result is that, given single word commands (commands without arguments), both versions of the script produce the same results.

When you respond to the prompt issued by the preceding command with **echo hi there**, the script generates an error because the script passes the string **echo hi there** as the command to execute; the SPACEs are not special characters that separate words because they are quoted. You get the same error message when you give the following command:

```
$ echo\ hi\ there
```