1. Explain the following unexpected result:

**$ whereis date**

date: /bin/date ...

**$ echo $PATH**

.:/usr/local/bin:/usr/bin:/bin

**$ cat > date**

echo "This is my own version of date."

**$ ./date**

Tue May 21 11:45:49 PDT 2013

A: whereis located the file, date.

echo $PATH tells us what is the content of the $PATH environment parament.

The third command creates a new file named date. If you want to terminate inputting, use control-d.

./date means to execute the file, date but it is not executable. The reason is “permission denied.” The file, date needs permission (chmod) to execute. Then, system maybe searches for the next directory from the $PATH environment variable and gives us the output of current date and time.

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

A: Yes, we can execute a shell script if I do not have read permission for the file containing the script. I can put a special sequence of characters on the first line of a shell script to tell the operating system which shell (or other program) should execute the file and which options I want to include. Because the operating system checks the initial characters of a program before attempting to execute it using exec, these characters save the system from making an unsuccessful attempt. For example, I can use #! as the first two characters of a script and add absolute pathname of the program to execute the script.

$ cat bash\_script

#! /bin/bash

echo “This is a test.”

The second way is I can give the name of the file containing the script as an argument to the shell (for example, bash bash\_script). Under bash I can give either of the following commands:

$ . bash\_script

$ source bash\_script

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

3. What is the purpose of the **PATH** variable?

The default value of PATH is determined when bash is compiled. The PATH variable specifies the directories in the order the shell should search them.

a. Set the **PATH** variable and place it in the environment so it causes the shell to search the following directories in order:

* **/usr/local/bin**
* **/usr/bin**
* **/bin**
* **/usr/kerberos/bin**
* The **bin** directory in your home directory
* The working directory

A: export PATH=/usr/local/bin:/usr/bin/:/bin:/usr/Kerberos/bin:

b. If there is an executable file named **doit** in **/usr/bin** and another file with the same name in your **~/bin** directory, which one will be executed?

A: It is determined by the order in the path statement. Hence, the file in /usr/bin will be executed.

c. If your **PATH** variable is not set to search the working directory, how can you execute a program located there?

A: You should use a command with pathname. For example, ~/bin/dolt.

d. Which command can you use to add the directory **/usr/games** to the end of the list of directories in **PATH**?

A: export PATH=$PATH:/usr/games

4. Assume you have made the following assignment:

$ person=zach

Give the output of each of the following commands.

a. echo $person

A: zach

b. echo '$person'

A: person

c. echo "$person"

A: Zach

5. The following shell script adds entries to a file named **journal-file** in your home directory. This script helps you keep track of phone conversations and meetings.

**$ cat journal**

# journal: add journal entries to the file

# $HOME/journal-file

file=$HOME/journal-file

date >> $file

echo -n "Enter name of person or group: "

read name

echo "$name" >> $file

echo >> $file

cat >> $file

echo "----------------------------------------------------" >>

$file

echo >> $file

a. What do you have to do to the script to be able to execute it?

A: Use the command, chmod, such as chmod +x journal.

b. Why does the script use the **read** builtin the first time it accepts input from the terminal and the cat utility the second time?

A: It reads the name from stdin and cats it to journal-file rather than stdout.

6. Assume the **/home/zach/grants/biblios** and **/home/zach/biblios** directories exist. Specify Zach’s working directory after he executes each sequence of commands. Explain what happens in each case.

a. **$ pwd**

/home/zach/grants

**$ CDPATH=$(pwd)**

**$ cd**

**$ cd biblios**

A: Now, Zach’s working directory is /home/zach/grants/biblios. The CDPATH variable allows you to use a simple filename as an argument to the cd builtin to change the working directory to a directory other than a child of the working directory. After CDPATH is set to /home/zach/grants, cd keeps on working to biblios and the working directory is /home/zach/grants/biblios.

b. **$ pwd**

/home/zach/grants

**$ CDPATH=$(pwd)**

**$ cd $HOME/biblios**

A: Now, Zach’s working directory is /home/zach/biblios. When you give cd an absolute pathname as an argument, cd does not use CDPATH.

7. Name two ways you can identify the PID number of the login shell.

A: ps, echo $$.

8. Enter the following command:

**$ sleep 30 | cat /etc/services**

Is there any output from sleep? Where does cat get its input from? What has to happen before the shell will display a prompt?

A: There is no output from sleep. The /etc/services file offers input for cat. The sleep command has to run to completion before the shell will display a prompt.

9. Write a sequence of commands or a script that demonstrates variable expansion occurs before pathname expansion.

A: $ **ls letter\***

letter1 letter2 letter3

$ **var=letter\***

$ **set | grep var**

var='letter\*'

$ **echo '$var'**

$var

$ **echo "$var"**

letter\*

$ **echo $var**

letter1 letter2 letter3

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

A: $ cat bash\_script

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

11. Explain the behavior of the following shell script:

$ **cat quote\_demo**

twoliner="This is line 1.

This is line 2."

echo "$twoliner"

echo $twoliner

a. How many arguments does each echo command see in this script? Explain.

A: Running echo "$twoliner", there is two arguments:

This is line 1.

This is line 2.

Running echo $twoliner, there is one argument:

This is line 1. This is line 2.

b. Redefine the **IFS** shell variable so the output of the second echo is the same as the first.

A: $ twoline=“This is line1.

>This is line2.”

$ unset IFS

$ echo –e $twoliner

This is line 1.

This is line 2.

12. Add the exit status of the previous command to your prompt so it behaves similarly to the following:

$ [0] **ls xxx**

ls: xxx: No such file or directory

$ [1]

A: The following command sets up the prompt described in the question:

PS1=’ [$?] ’