1. What does the shell ordinarily do while a command is executing? What should you do if you do not want to wait for a command to finish before running another command?
A: While the command is executing, the shell warts for the process to finish and it sleeps. When the program finishes execution, the shell returns to an active state (wakes up), issues a prompt, and waits for another command. When you run a command in the background, you don't have to wait for the command to finish before running another command.

2. Using sort as a filter, rewrite the following sequence of commands:
$ sort list > temp
$ lpr temp
$ rm temp
A: command: $ cat list | sort | lpr

3. What is a PID number? Why are these numbers useful when you run processes in the background? Which utility displays the PID numbers of the commands you are running?
A: PID number (process identification number) is a larger number assigned by the operating system. These numbers are useful because each of them identifies the command running in the background. PS (process status) utility displays the PID numbers of the commands you are running.

4. Assume the following files are in the working directory:

$ ls

intro	notesb	ref2	section1	section3	section4b
notesa	ref1	ref3	section2	section4a	sentrev

Give commands for each of the following, using wildcards to express filenames with as few characters as possible.
a. List all files that begin with **section**.
A: $ **ls section***
b. List the **section1**, **section2**, and **section3** files only.
A: $ **ls section[1-3]**
c. List the **intro** file only.
A: $ **ls i***
d. List the **section1**, **section3**, **ref1**, and **ref3** files.
A: $ **ls *[13]**

5. Refer to the info or man pages to determine which command will
(hint look at the wc command)
a. Display the number of lines in its standard input that contain the word **a** or **A**.
A: $ **command | grep –wci a**
b. Display only the names of the files in the working directory that contain the pattern **$(**.
A: **$ls *$\(***
c. List the files in the working directory in reverse alphabetical order.
A: **$ls | sort –r or $ ls -R**
d. Send a list of files in the working directory to the printer, sorted by size.
A: **$ ls -S | lpr**

6. Give a command to
a. Redirect standard output from a sort command to a file named **phone_list**. Assume the input file is named **numbers**.
A: $ **sort numbers > phone_list**
b. Translate all occurrences of the characters **[** and **{** to the character **(**, and all occurrences of the characters **]** and **}** to the character **),** in the file **permdemos.c**. (*Hint:* Refer to the tr man page.)
A: $ **cat permdemos.c | tr '[{}]' '(())'**
c. Create a file named **book** that contains the contents of two other files:
part1 and **part2**.
A: $ **cat part[12] > book**

7. The lpr and sort utilities accept input either from a file named on the command line or from standard input.
a. Name two other utilities that function in a similar manner.
A: grep, cat
b. Name a utility that accepts its input only from standard input.
A: tr

8. Give an example of a command that uses **grep**
a. With both input and output redirected.
A: $ **grep \$Name < *.c > name_list**
b. With only input redirected.
A: $ **grep -i aplustudent < demo**
c. With only output redirected.
A: $ **grep asd hong.txt > hong2.txt**
d. Within a pipeline.
A: $ **file /usr/bin/* | grep "Again shell script" | sort -r**
In which of the preceding cases is grep used as a filter?
A: Part **d** uses grep as a filter.

9. Explain the following error message. Which filenames would a subsequent ls command display?
$ **ls**
abc abd abe abf abg abh
$ rm abc ab *
rm: cannot remove 'abc': No such file or directory
A: It is duplicated. “ab*” means any files which starts ab. Hence, rm receives a list of files that includes abc twice. After rm removes abc, it will get an error message when it is asked to remove abc again. After giving the preceding rm command, ls does not list any files.

10. When you use the redirect output symbol (>) on a command line, the shell creates the output file immediately, before the command is executed. Demonstrate that this is true.
A: I will some commands as follows:

$ **ls hong**
ls: hong: No such file or directory
$ **ls id > hong**

ls: id: No such file or directory
$ **ls hong**
hong

The first command shows the file **hong** does not exist in the working directory. The second command uses **ls** to attempt to list a nonexistent file (**id**) and sends standard output to **hong**. The **ls** command fails and sends an error message to standard error. Even though the **ls** command failed, the empty file named **hong** exists. Because the **ls** command failed, it did not create the file; the shell created it before calling **ls**.

11. In experimenting with variables, Max accidentally deletes his **PATH** variable. He decides he does not need the **PATH** variable. Discuss some of the problems he could soon encounter and explain the reasons for these problems. How could he *easily* return **PATH** to its original value?
A: When he deletes it, there is no way to connect users with commands or utilities. Hence, it is very hard to run these commands because Max does not know where the directories or files are. A simple way to return PATH to its original value is to log out and then log back in.

12. Assume permissions on a file allow you to write to the file but not to delete it.
a. Give a command to empty the file without invoking an editor.
A: $ **cat /dev/null > hong_file**
b. Explain how you might have permission to modify a file that you cannot delete.
A: When you have write permission only for this file and execute permission only for the directory, you can modify it but you cannot delete it.

13. If you accidentally create a filename that contains a nonprinting character, such as a CONTROL character, how can you remove the file?
A: To remove it, you should use some wildcard characters to find it. Firstly, use echo to confirm it. Secondly, use rm to delete it. For example, assume we want to remove the file named hongCONTROL-u. The commands are:

$ echo hong?u
hongu
$ rm hong?u

14. Why does the noclobber variable *not* protect you from overwriting an existing file with cp or mv?
A: **noclobber** prevents overwriting a file using redirection (>) but it does work with commands or utilities, such as cp and mv.

15. Why do command names and filenames usually not have embedded SPACEs? How would you create a filename containing a *SPACE*? How would you remove it? (This is a thought exercise, not recommended practice. If you want to experiment, create a file and work in a directory that contains only your experimental file.)
A: SPACE has special meaning to the shell. It separate tokens (elements) on the command line. If you want to create or remove a filename containing a SPACE, you should quote the SAPCE. For instance : touch hong\ zhang.txt; rm hong\ zhang.txt.

16. Create a file named **answer** and give the following command:

$ **> answers.0102 < answer cat**

Explain what the command does and why. What is a more conventional way of expressing this command?

A: this command asks shell to redirect standard output to **answers.0102**, to redirect standard input from **answer**, and to execute the cat utility. More conventionally, the same command is expressed as **cp answer answers.0102**