**CSCI.258 (63.258) - Introduction to Operating Systems Using UNIX**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Professor Krishnamoorthy, Computer Science Dept., Framingham State College**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Homework #2 – 150 Points**

**Coverage: Chapters 7, 8, 9, 10, 11, and 13 from Das Textbook**

**Refer Syllabus for due date**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objectives:**

1. Learn about system processes and user processes and process information
2. Learn about pipe
3. Learn to run command in the background
4. Learn about basics of shell, shell variables, customizing environment, and shell programming
5. Learn about shell scripting: read, shift, test, if, for, while, exit, and expr
6. Learn to use two character shell variables
7. Learn to use additional UNIX commands such as head, tail, comm, cmp, diff, sort, uniq, grep, umask, sort
8. Learn about *here* document
9. UNIX has the following three standard devices:
10. standard input device (stdin) with default device sest to keyboard.
11. standard output device (stdout) with default device set to display (monitor), and
12. standard error (stderr) with default device set to display (monitor)

Numerical values are also used to represent these devices: stdin is 0, stdout is 1 and stderr is 2. Sometimes, these numbers are used in shell scripting.

**You have to use the UNIX server at FSU to get output for the commands.**

**How to get the answers to the questions in the homework?**

Use the method explained in Homework #1.

**IMPORTANT NOTE: You must study about the commands needed for each question and know how to use them before you begin answer the question.**

**Q1. Hint: Command to study: grep, wc, echo, pipe ( | ), ps, process states, and man pages for ps options**

**NOTE: Use man command before you start answering this question and before issuing script command to find the options to use with the ps command to find the answers in this question as well as to find the characters representing the various states of a process.**

1. Issue ps command to display your processes, that is, processes for which you are the owner (creator of the process)
2. Display information about your processes using long option
3. With echo command, display the name (command name) and the state character for your shell process
4. Using echo command, answer the process id (pid) of the shell process and the parent pid of your shell process
5. Display all the processes in the system using an appropriate option other than long option

**Pipe takes the standard output of the command preceding it and feeds to standard input of the command following it. The pipe is indicated by | character between the commands as**

**Command1 | Command2**

**The grep command can be used to search for a string in one or more files. When the line with the string is found, it is displayed.**

**The wc command can be used to find the number of lines, number of words and number of character in a file or the input supplied to it.**

Using wc, grep and pipe and do the following:

1. Issue ps command to display all the processes in the system and pipe the output to a grep command to display the line containing the init process.
2. Using the echo command answer the process ID (pid) of the init process.
3. Issue who command and pipe the output to wc command to display the number of lines, word and characters displayed by the who command

**Q2. Hint: Study about: predefined shell variables, and .profile script file, echo**

SHELL, HOME, PATH, MAIL and TERM are predefined shell variables. You can use the value of a shell variable in a shell command putting $ in front of it. For example, to display the value of the HOME directory of the user, specify $HOME in the echo command like echo $HOME.

1. Display the value of SHELL
2. SHELL contains the pathname of the shell program you are using. With echo command, answer what is the name of the shell program
3. Display the value of HOME
4. With echo command, answer what the HOME value stand for
5. Display the value of PATH
6. PATH contains the list of directories shell searches for the nonbuilt-in command (that is, the executable program file with name same as the command you typed). The directories in the list are separated by : character.

With echo command, answer whether the list contains . (dot or period) and whether the shell searches the current working directory.

1. Display the value of MAIL
2. With echo command, answer where you’re the system stores your emails
3. Display the value of TERM
4. With echo command, answer what is the type of the terminal you are using. Note: The combination of keyboard and monitor is called a terminal.
5. The .profile file is in your login directory. It is a startup shell script file containing shell (UNIX) commands. The system executes the UNIX shell commands in this file each time you login to setup your session environment before shell displays the first shell prompt (which is $ by default). Display the content of .profile file
6. With echo command specify the list of predefined shell variables in the .profile file

**Q3. Scripting with if statement: Commands to read: sleep, ps, kill, echo, cat, creating your own shell variable, if statement, and two-character shell variable $! and $?**

1. Exit from script, if is currently enabled; if it is not enabled, skip this item (1)
2. Using the vi editor, create a shell script file to do the following:
3. Issue “*sleep 1500 &”* command to run it as a background process.
4. The two-character shell variable $! contains the process ID (pid) of the above sleep process running in the background. Display the content of the variable $!.
5. Issue ps command to see that the sleep process is running.
6. Store the sleep process ID (pid) in a user defined (your own) shell variable
7. Issue *cat* command with a filename that does not exist
8. The two character shell variable, $? Contains the completion status of the previous cat command. Using an if statement, check the completion status of the above cat command by checking the value of $?. If cat command is successful, then echo *cat command completed successfully* else echo *cat command failed*.
9. Kill the background process with pid stored in your shell variable above as the argument to kill command
10. Issue *ps* command to see that the background process is terminated.
11. Echo, the *script is done* using *echo* command
12. Issue *script* command to capture the output of the items below
13. Add execute permission to the file
14. Display the attributes of the script file
15. Display the content of the script file
16. Run the script file by typing the name of the script file.

NOTE: You must submit both the content of the shell script file and the output generated by running the script.

**Q4. Read about *test,* *shift,* *while* loop, *expr*, displaying the value of the two-character variable $1 using *echo* command**

1. Exit from script, if is currently enabled; if it is not enabled, skip this item (1)
2. Using the vi editor, create a shell script file that displays the first 12 command line arguments. You must use a *while* loop, *expr* and *shift* shell commands in the script to do this question.
3. Issue *script* command to capture the output of the items below
4. Add execute permission to the file
5. Display the attributes of the script file
6. Display the content of the script file
7. Run the script file by typing the name of the script file and specifying 14 or more arguments

**The following are helpful hints in creating the shell script file for this question:**

The *expr* command can be used to perform arithmetic operations. For example, the value of a user defined variable, count can be incremented by 1 (that is, add 1 to count) by using the following expr command.

count=`expr $count + 1`

NOTE: (1) there should be no space before and after = when storing a value into a variable, (2) Reverse quote (`) is used and not forward quote (‘) in the above command.

Following are the relational operators that can be used in shell scripting:

**Operator Description Operator in Java and C++**

-eq equal ==

-ne not equal !=

-lt less than <

-le less than or equal <=

-gt greater than >

-ge greater than or equal >=

The *test* command is used to evaluate the value of an expression. Value of an expression with the above relational operators will be a Boolean value, true or false. The following test command checks whether the value of the variable *count* is less than 12:

test $count -lt 12

Alternately, the square brackets [ ] can be used instead of the word *test* to evaluate an expression. For example, the above *test* command can be issued as follows using square brackets:

[ $count -lt 12 ]

Note, there must be space before and after [ and ], before and after lt and after 12 in the above statement.

**Q5. Additional UNIX commands to study: tail, head, cmp, diff, comm, uniq, touch, umask, grep, vi, cat, ls, and echo**

Two files can be compared using *cmp* command and *diff* command depending on requirements.

In order to do this question, create two files with 7 to 10 names with one name in each line. Make sure that some names in the two files are same and some are different. Name the files *hw2test1*, and *hw2test2*. Ensure that there are few consecutive lines with identical names (duplicate lines) in each file. Also, make sure that some of the names are the same in both files.

Create these two files before issuing the script command to answer this question.

1. Display the content of hw2test1
2. Display the content of hw2test2
3. Compare hw2test1 and hw2test2 using *cmp* command
4. Find the differences between hw2test1 and hw2test2 using *diff* command
   1. Notice the differences in the output between *cmp* and *diff* commands.
5. *Sort* hw2test1 and store the result by redirecting the output to hw2test1sorted file.
6. *Sort* hw2test2 and store the result by redirecting the output to hw2test2sorted file.

Two files can also be compared *comm* command. There are differences between *cmp*, *diff* and *comm* commands. Read about them and understand the differences.

1. Compare hw2test1sorted hw2test2sorted using *comm* command
2. Display ONLY the bottom 5 lines of the hw2test1 file
3. Display ONLY the top 10 lines of the hw2test1 file without using any option in the command
4. Issue appropriate command to display the content of hw2test1 removing consecutive duplicate lines
5. Create the file hw2before\_setting\_umask using *touch* command
6. The *umask* command can be used to set or change the default file permissions to be set when a new file is created. Issue *umask 600* command
7. Create the file hw2after\_setting\_umask using *touch* command
8. Display the file permissions set on hw2before\_setting\_umask and hw2after\_setting\_umask
9. Using echo command answer what permissions were removed for the owner due to the above umask command

The grep command can be used to search for a string in one or more files. When the line with the string is found, it is displayed. Check this by issuing the following grep commands.

1. Issue a *grep* command on hw2test1 with a name that exists in hw2test1.
2. Issue a *grep* command on hw2test[12] with a name that exists in both files (hw2test1 and hw2test2).
3. Issue a *grep* command on hw2test[12] with a name that does not exist in both files.
4. What difference you see between (16) and (17) by Observing the output of above three grep commands