COSC 350 System Software: Lab #2

Environment Setting

Configuration Files

The two most important configuration files are **.bashrc** and **.bash_profile** The file **.bashrc** is executed every time you start a subshell. The file **.bash_profile** is executed every time you log in.

Environment Variables

Here are some common environment variables:

Variable	Meaning
EDITOR	User's preferred editor.
HOME	User's home (login) directory.
HOSTNAME	Name of the host machine.
LD_LIBRARY_PATH	Path to search for dynamically loadable libraries.
LESS	Flags to provide to the GNU "less" pager.
MAIL	Location of incoming mail.
MANPATH	Path to search for manual pages.
MORE	Flags to provide to the "more" pager.
PAGER	User's preferred pagination (terminal file display) program.
PATH	The sequence of directory prefixes that bash applies in searching for a file known by an incomplete path name.
PWD	User's current working directory (yes, it's PWD, not CWD).
SHELL	The file name of the user's login shell.
TERM	Terminal type for which output is to be prepared.
USER	The login name of the user.

Task #1:

1. Write down the value of your **PS1** environment variable as well as the file in which it was set.
2. Write down the values of the above environment variables on your Linux machine. Indicate any that have no value.

Setting the prompt

There are a lot of escape sequences offered by the Bash shell for insertion in the prompt. From the Bash 2.02 man page:

When executing interactively, bash displays the primary prompt PS1 when it is ready to read a command, and the secondary prompt PS2 when it needs more input to complete a command. Bash allows these prompt strings to be customized by inserting a number of backslash-escaped special characters that are decoded as follows:

String	Meaning
\a	ASCII bell character
\d	date in "Thu Mar 17" format
\e	ASCII escape character
\H	hostname
\n	newline
\r	carriage return
\s	name of the shell
\t	current time in 24-hour HH:MM:SS format
\T	current time in 12-hour HH:MM:SS format
\@	current time in 12-hour am/pm format
\u	username of current user
\v	bash version (short form)
\V	bash version (long form)
\w	current working directory
\W	basename of current working directory
\!	history number of this command (starting at historic time)
\#	command number of this command (starting at shell creation time)
\$	if effective UID is 0 (root), a #, otherwise a $
\nnn	character corresponding to octal number nnn
\\	a backslash
\[	begin a sequence of non-printing characters. Could be used to embed a terminal or control sequence into the prompt.
\]	end a sequence of non-printing characters

Task #2:

1. Save your prompt. Write down exactly how you did it.
2. Change your prompt so it looks like **[COSC350 basecwd]:**
   where "**basecwd**" means the basename of the current working directory. Write down exactly how you did it.
3. Change your prompt to its previous value. Write down exactly how you did it.

Command-line completion in Linux

You can save a lot of typing by getting bash to complete your commands and file or directory names. Do this by pressing the tab key after enough of the word you are trying to complete has been typed in. If bash can figure out the rest, it will complete the word for you. Otherwise, it's possible that after hitting tab there are multiple possibilities for the completion. Press tab again and it will list the possibilities.

Task #3:

1. Try out command-line completion. Just type the beginning few letters of something and hit the "Tab" key. (Suggestion: from your home directory, type **ls -l D**, then hit Tab.) Play around a little with this feature. Nothing to write down.

Redirecting Standard Input and Output

There are always three default "files" open, **stdin** (directed by default from the keyboard), **stdout** (directed by default to the screen), and **stderr** (error messages; also directed by default to the screen). These, and any other open files, can be redirected. Redirection simply means changing where the output or input is directed.

Redirection Format	Meaning	Example
COMMAND_OUTPUT > FILENAME	Output of the command is redirected to the file. Creates the file if not present, otherwise overwrites it.	**ls -l > foo** Writes a long listing to the file named **foo**, creating the file if necessary.
COMMAND_OUTPUT >> FILENAME	Output of the command is redirected to the file. Creates the file if not present, otherwise appends to it.	**ls -l >> foo** Appends a long listing to the file **foo**, creating the file if necessary.
COMMAND_OUTPUT 2> FILENAME	Error output of command is redirected to the file. Normal output of the command is not redirected.	**nosuchcommand 2> foo** Redirects the error output to the file **foo**.
2>&1	Redirects **stderr** to get sent to same place as **stdout**.	**cmd > foo 2>&1** Redirects normal output to the file **foo** and redirects error output to the same place.
COMMAND < FILENAME	**stdin** is redirected from the file to the command.	**sort < foo** Sends the contents of file **foo** as input to the **sort** command.

Task #4:

1. Invoke **ls** with a non-existent filename. You should see the error output on the screen. Do it again, but redirect the error output to a file named **bar**. Write down exactly how you did this.
2. Do it again, but redirect the error output to the "gone forever" file **/dev/null**. Write down exactly how you did this.
3. Create a file named **foo** by echo-ing the numbers 3,5,2,1 into it, one number per line. Write down exactly how you did this.
4. Create a file named **bar** by **cat**-ing **foo** into it. Write down exactly how you did this. (Yes, **cp** would also work, but this lab exercise is about redirection.)

5. Redirect input from **foo** (it contains numbers, right?) to the **sort** function. You should see the sorted numbers on the screen. Write down exactly how you did this. Did the numbers turn out sorted numerically? If not, explain how the sort was done.
6. Do it again, but redirect the output from the screen into the file **bar**. Write down exactly how you did this.

Piping

Bash allows processes to be connected to other processes. When we connect the output of one process to the input of another, we say we are *piping* the first process into the second. You can think of a water pipe connecting the first process to the second; output of the first "travels" through the pipe into the second process. Any number of processes can be piped together.

The pipe symbol is **|** (found on most keyboards above the backslash character). The format for using pipes is **CMD1 | CMD2**.

For example: **ls | less** pipes the output of the command **ls** into the command **less**.

Another (but silly) example is **ls | cat | less**. Here we have three processes piped together. The output of **ls** is piped into **cat** which merely copies it and pipes its output into **less**.

It's important to realize that all of the processes in the pipe chain run concurrently. As soon as one process starts producing output, the next process starts working on it.

Task #5:

1. Create a file named **numbs** that contains the integers 1 through 100, one integer per line. The file will have 100 lines. Write down a short description of how you did this. (You can do it any way you want, including dumb brute force. You might also want to consider the bash **for** loop or a small C++ program.)
2. Run **wc** on the file **numbs**. Write down the output and your explanation of what it means. Check the man page for **wc** if you're not sure.
3. Use pipes and redirection to produce a second file named **somenumbs** that contains lines 25 through 38 of **numbs**. Write down exactly what you did.
4. Run **wc** on the file **somenumbs**. Write down the output and your explanation of what it means.

Task #6: Bash shell script (use one of loop statements)

Write shell script using for loop to print the following patterns on screen

a) task6a.sh

```
1
22
333
4444
55555
```

b) task6b.sh

```
*
**
***
****
*****
****
***
**
*
```

c) task6c.sh: your program ask a number between 5 to 9. If an input is not between 5 and 9, display error message and ask again.

```
      1
     2 2
    3 3 3
   4 4 4 4
  5 5 5 5 5
 6 6 6 6 6 6
7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
```

Task #7: Bash shell scripts

Write a script named task7.sh to calculating factorial of given number by using while loop. The shell accepts one integer argument as a parameter and calculates factorial and display the result.

Task #8: Bash shell scripts

Write a script named "task8.sh" to print given numbers sum of all digit. The shell accept one integer argument as a parameter. Your program must check number of argument is one. If number of argument is not one, your program must display error message and exit.

For example)

```
[separk@sejong] ./sumdigit.sh 345
Sum of digit for number is 12

[separk@sejong] ./sumdigit.sh
You need pass one numerical argument
```

Task #9: Bash shell scripts

Write a Bash shell script named **task9.sh** that searches for a word in a file as follows:

1. Asks user for a directory in which to find the file.
   - If not a valid directory, quits with appropriate error message.
2. Asks user for the name of a readable file in that directory, giving the user three attempts to name one.
   - If no readable file is named, quits with appropriate error message.
3. Asks user for a word to find in the file.
   - If word is in the file, <word> FOUND! is printed
   - Otherwise, <word> NOT FOUND is printed
4. Exit codes are to be:
   0 success
   1 no such directory
   2 no such file (after three attempts)
   3 file is not readable
   4 word not found in the file

What to Hand In

Please organize your materials in the order given below. Clearly identify your materials by task. Write your name on every page. Staple the pages neatly.

Task #1:
- Value of your **PS1**
- Values of the given environment variables.

Task #2:
- Command to save prompt.
- Command to change prompt.
- Command to change prompt to its previous value.

Task #3:
- Nothing required.

Task #4:
- Command to redirect error output to **bar**.this.
- Command to redirect error output to **/dev/null**.
- Command(s) to create **foo** containing numbers.
- Command(s) to cat **foo** into **bar**.
- Command to redirect **foo** into **sort**
- Answer to numeric sort question with explanation.
- Command to redirect above into **bar**.

Task #5:
- Description of how you created **numbs**.
- Output of **wc** on **numbs** and your explanation of what it means.
- Command(s) using pipes and redirection to produce **somenumbs**.
- Output of **wc** on **somenumbs** and explanation.

Task #6, 7, 8, 9: submit by email: cosc350fall20@gmail.com