Lab 6.1: Directory and file organization

Scenario: Once again files have managed to accumulate in your home directory and you have decided that it is time to organize things. This time you will use your knowledge of the **bash** shell to perform more complex file management tasks.

Deliverable: A more organized home directory, with files placed into the appropriate subdirectories, and some files backed up to `/tmp/archive`.

Instructions:

1. Log in as user `student` with the password *`student`*. If you are using the graphical environment, start a terminal by clicking Applications->Accessories->Terminal

2. Immediately after logging into the system, you should be in your home directory. Verify this with **pwd**.

```
[student@stationX ~]$ pwd
/home/student
```

> **Note:**
>
> You can also tell you are in your home directory by noting the ~ in your command prompt.

3. Use **touch** and curly brace expansion to create some files for use in this sequence.

```
[student@stationX ~]$ touch ↙
 {report,memo,graph}_{sep,oct,nov,dec}_{a,b,c}_{1,2,3}
```

4. Use the **ls** command to examine the results of the last command. You should find that it created 108 new, empty files (you do not need to count) in your home directory. These files represent the data files that you will use in the remainder of this sequence. If for some reason you do not see these files, ask the instructor for assistance; without these files, the remainder of this lab will not work.

5. In order to organize your files you must first create some new directories. Use mkdir to create some subdirectories directly inside your home directory:

```
[student@stationX ~]$ mkdir a_reports
[student@stationX ~]$ mkdir september october november ↙
 december
```

Again, use **ls** to examine your work.

6. Create some additional subdirectories inside one of your new directories using the following commands.

```
[student@stationX ~]$ cd a_reports
```

to change to the directory. Then:

```
[student@stationX a_reports]$ mkdir one two three
```

Use **ls** to verify that you have three new directories named one, two, and three under your `a_reports` subdirectory.

7. Begin by moving all of the "b" reports out of your home directory and grouping them by month. When working with complicated wild card patterns, it is a good idea to pre-verify the operation to ensure you are operating on the correct files. One way to do this is to replace your command with a harmless command using the intended wild card pattern.

```
[student@stationX a_reports]$ cd
[student@stationX ~]$ ls -l *dec_b_?
```

You should see the 9 "december", "b" files listed. Move one of them to the december directory:

```
[student@stationX ~]$ mv graph_dec_b_1 december/
```

Now move the rest of them with:

```
[student@stationX ~]$ mv *dec_b_? december/
```

List the contents of the december subdirectory to verify the move operation was successful:

```
[student@stationX ~]$ ls -l december/
total 9
-rw-rw-r--  1 student  student   0 Oct 16 22:17 ↙
 graph_dec_b_1
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 graph_dec_b_2
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 graph_dec_b_3
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 memo_dec_b_1
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 memo_dec_b_2
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 memo_dec_b_3
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 report_dec_b_1
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 report_dec_b_2
-rw-rw-r--  1 student  student   0 Oct 16 22:16 ↙
 report_dec_b_3
```

8. Move all of the remaining "b" reports into their respective directories:

```
[student@stationX ~]$ mv *nov_b_? november/
[student@stationX ~]$ mv *oct_b_? october/
[student@stationX ~]$ mv *sep_b_? september/
```

9. You will now collect the "a" reports into their respective corresponding numbered directories. Notice the use of ~ as shorthand for "your home directory". The combination of the wild card and the pattern specifies all files that end in _a_1 in your home directory.

```
[student@stationX ~]$ cd a_reports
[student@stationX a_reports]$ mv ~/*_a_1 one/
```

The "september" "a_1" files are old and no longer needed. Use **ls** to make sure you have created a pattern that matches only these files, then delete them, and verify that the other "a_1" files were moved properly:

```
[student@stationX a_reports]$ cd one
[student@stationX one]$ ls *sep*
[student@stationX one]$ rm *sep*
[student@stationX one]$ ls
graph_dec_a_1  graph_oct_a_1  memo_nov_a_1  report_dec_a_1 ↙

report_oct_a_1 graph_nov_a_1  memo_dec_a_1   memo_oct_a_1
report_nov_a_1
```

10. Move the final "a_2" and "a_3" reports into their respective directories. To make life interesting, we will move them from the current directory, using both relative and absolute pathnames. First, use the **pwd** command to identify the current directory:

```
[student@stationX one]$ pwd
/home/student/a_reports/one
```

Verify the pattern that references the "a_2" files with **ls**, then move them using absolute pathnames:

```
[student@stationX one]$ ls /home/student/*a_2*

[student@stationX one]$ mv /home/student/*a_2* ↙
 /home/student/a_reports/two/
```

Even though your current directory is /home/student/a_reports/one, you can move files from /home/student to /home/student/a_reports/two because you specified the files' pathnames - in this case, absolute pathnames.

Now move the "a_3" files using relative pathnames. Again, first verify the pattern references the correct files.

```
[student@stationX one]$ ls ../../*a_3*
[student@stationX one]$ mv ../../*a_3* ../three/
```

11. Return to your home directory, and use **ls** to verify that the only files remaining in this directory are the "c" files (i.e., `graph_dec_c_1`, `graph_dec_c_2`, ...)

12. The "`c_1`" and "`c_2`" report files for each month are important, and you want to make a backup of them in another directory:

```
[student@stationX ~]$ mkdir /tmp/archive
[student@stationX ~]$ cp report*[12] /tmp/archive/
```

Additionally, all the report files for the month of December should be backed up to the `/tmp/archive` directory. Note the use of the **-i** option to have **cp** prompt before overwriting any files.

```
[student@stationX ~]$ cp -i report_dec* /tmp/archive/
cp: overwrite `/tmp/archive/report_dec_c_1'? n
cp: overwrite `/tmp/archive/report_dec_c_2'? n
```

13. Now that you have backed up the few "c" files that are important to you, you want to delete all of the files still remaining in your home directory. Examination of the remaining files reveals that the wild card ***c*** will match all of them. However, to ensure that you do not accidentally delete other files, try out the following commands, examining the output:

```
[student@stationX ~]$ ls *c*
...output omitted...
[student@stationX ~]$ ls -Fd *c*
...output omitted...
```

14. Delete the remaining "c" files in your home directory. Once more we'll use **ls** before issuing a destructive command.

```
[student@stationX ~]$ ls *c_[1-3]
...output omitted...
[student@stationX ~]$ rm *c_[1-3]
[student@stationX ~]$ ls
a_reports  december  november  october  Projects september
```

Lab 6.2: Changing your bash prompt

Scenario: You have decided to customize your bash prompt to display the full path of the current working directory and the shell history number, and to make some other cosmetic changes.

Instructions:

1. In a terminal window, display the current value of your primary prompt string. Note the presence of static ASCII characters and backslash-escaped special characters in the prompt.

```
[student@stationX ~]$ echo $PS1
```

2. Change your prompt to print only a static string for testing purposes:

```
[student@stationX ~]$ PS1='Red Hat -> '
Red Hat ->
```

Be sure to include a space after the -> so that commands have padding, that is, so that commands do not appear to touch the prompt (this is almost always desirable with prompts).

3. That prompt is not very useful, so restore the traditional bash dollar-sign, along with the name of the host:

```
Red Hat ->  PS1='\h $ '
stationX $
```

4. Insert the bash history prompt special character `\!` between the hostname and dollar-sign. Make sure you pad it on each side with a space character. This will insert the number of the current command in bash's history. The actual number you see will probably be different than the one shown below.

```
stationX $ PS1='\h \! $ '
stationX 21 $
```

5. Now refer to the **bash** man page to find the special character for the current working directory. (Search the man page for `PROMPTING`, and beware: you want the special character for the full pathname, not the basename as in the default prompt.) Insert that special character into your `PS1` prompt string preceded by a colon.

6. Your customized prompt should appear as shown in the following examples. If not, continue to work on it.

```
station1:~ 21 $ cd /tmp
station1:/tmp 22 $
```

---

7. Edit your new definition of `PS1` into your `.bashrc`, then open a new terminal window to make sure your new prompt is in effect.

8. If you like your new prompt, leave things as they are but be aware that your prompt will differ from the one displayed in subsequent labs. If you prefer, comment out the line you just added in `.bashrc` and start a new shell to go back to the default prompt.