Exercise – Using Files And Directories

1. Review questions
2. What is the basic difference between a hard link and a symbolic links?

A hard link points the new file to the old files inode whereas a symbolic link has the inode of the new file point to the old file which in turn directs to the old files inode.

1. What is meant by a 'recursive' operation?

Recursive applies the action in question to all sub directories.

1. What is the purpose of the **/etc** directory?

The etc directory contains sub directories of configuration data.

1. What is an absolute (full) pathname ?

The absolute pathname is the full path taken from the root location to get to your location.

1. Which two commands allow to remove directories ?

Rmdir (Empty Directory)  
Rm –r (Directory has Files/Sub Directories)

1. Scenario: You created a symbolic link file called **symfile** to a text (source) file. Subsequently, you have removed the source file.   
   What will happen if you run **cat** **symfile** command ?

The Symbolic link is removed along with the source file.

1. Navigating the system.

This exercise will help in understanding differences in relative and absolute pathname specification.

Note: This is a ‘pen and paper’ exercise only.

* 1. Assume the following directories exist for a hypothetical user ***qa***:

**/home/qa/project/source/interface**

**/home/qa/project/source/dbstuff**

**/home/qa/project/includes**

First, draw these directories in a tree structure.

* 1. Now, assuming that your current working directory is:

**/home/qa/project/source/interface**

Write down the relative paths to:

**/home/qa/project/source/dbstuff**

**/home/qa/project/includes**

**/home/qa/project**

1. Moving around and listing directory content

This exercise will help your confidence in navigating the system and looking at the contents of a directory. Don’t worry if some of them appear too easy, as this is just the beginning.

1. Identify your current working directory.
2. List the content of your current directory (show filenames only).
3. List the content of your current directory, including files that begin with the dot.
4. List the entire content of your current directory (with their attributes).
5. Change to the **/etc** directory, confirm where you are, then list its contents.
6. Now try the following command and observe the result.   
     
   $ **ls r\***

Is the result what you expected?

1. Use manual pages for **ls** and try to identify the option which would prevent **ls** listing directory contents:
2. Change back to your home directory and confirm where you are.
3. Exploring the attributes of files that have been copied, moved, etc.  
      
   This exercise will also turn your attention to file-access permissions, although we are not going to discuss these here in any detail yet.
4. Copy the /**etc/passwd** file into your home directory, giving it the same name in the destination directory (**passwd**).
5. Do a long listing (**ls -l**) of both files: the original **/etc/passwd** and the newly-created copy and notice the attributes.

Which attributes were retained from the original file, and which have changed?

1. What are the i-node numbers of the two **passwd** files?
2. Rename the newly created **passwd** file to a file called **pass1** (in your home directory).
3. Do a long listing of the newly-created **pass1** and notice the attributes, including the i-node number.

Which attributes were retained from the original (**/etc/passwd**) file, and which have changed?

1. Working with directories.
2. Create two new directories (in ***your*** home directory), called **project1** and **project2**.
3. There is a command called **touch**, whose primary purpose is to change a file's timestamp. For example:

$ **touch lilavati**

would update the timestamp of this file to the current time. A side-effect of the **touch** command is that if the file does not exist, it will be created.

Inside the **project1** directory create a further subdirectory, called **test1**, and within **test1** create two new (empty) files, called **file1** and **file2**.

1. Change into **project2** directory using relative pathname (bear in mind that unless you moved around since the previous exercise, you are currently placed in the **project1** directory).
2. Whilst in **project2** directory, do a long listing of its parent directory (still using the relative pathname).
3. Whilst still in **project2** directory, do a long listing of the **test1** directory (still using the relative pathname).
4. Now return to your home directory and create a further subdirectory, called **test2**, underneath **project2**.   
   The new directory can be created using several distinct methods of pathname specification. How many ways of specifying the command line can you think of to do just this one simple operation?
5. Remove **project1** directory created earlier, but make sure you are prompted to confirm the deletion of each file or directory within.   
     
   Hint: The directory **project1** contains a sub-directory **test1**, and cannot therefore be deleted with **rmdir,** which only deletes empty directories.
6. Working with links
7. In your home directory, create a hard link to the **pass1** file, called **mypass.**
8. Compare the attributes (including i-node numbers of both files (**pass1** and **mypass**). Which attributes are different?
9. Run a command that would attempt to create a hard link to the **project2** directory, called **myproj**.
10. In your home directory, create two soft (symbolic) links: **words**, pointing to an existing file: **/usr/share/dict/words** and **sldoc**, pointing to an existing directory: **/usr/share/doc**.   
      
    Notice that whilst hard link pointing at a directory was not allowed, creation of a symbolic link is fine.
11. Check the attributes of the newly created symbolic link files, including their   
    i-node numbers. (hint: use either **stat** or **ls** **–lid**)
12. Compare the attributes of the symbolic link files with those of the files they point at.